Improving Missing Data Imputation using Generative Adversarial Network-based Methods

Hanna Anderberg, Sofia Wadell

Master's thesis
2023:E23

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematical Statistics

Centrum Scientiarum Mathematicarum

Master's Theses in Mathematical Sciences 2023:E23
ISSN 1404-6342

LUTFMS-3474-2023

Mathematical Statistics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lu.se/

Abstract

In a modern context, organizations increasingly rely on data analysis and the importance of data quality have accordingly become even more crucial. In this context, missing values pose a significant challenge compromising the utility of the data. In an ideal scenario data should be collected in a way so that the missing values are avoided, but practical and cost constraints often make this unfeasible. Consequently, various approaches have been developed to address the issue of missing values. Rather than discarding incomplete observations and compromising the sample size, imputing the missing values has the potential to improve predictions and imputation outcomes. Furthermore, it is a relatively straightforward process in terms of cost and effort.

In addition to this, Generative Adversarial Networks (GANs) have lately gained attention as a recent breakthrough in machine learning, offering novel possibilities for data handling. This study explores two aspects in which GANs can potentially can improve data imputation. Firstly, the performance of an imputation-focused GAN model, GAIN, is compared against other state-of-the-art methods through an extensive evaluation. Secondly, the impact of incorporating synthesized data, generated by a GAN framework named CTGAN, into the training data of imputation models is evaluated.

Our findings reveal that GAIN was outperformed by other data imputation methods. Despite this, its potential is not questioned, as further optimization of hyperparameters and network structure specific to the data set is believed to enhance its performance. The result of this study however emphasizes the clear challenges of the time-consuming training and optimization processes of GANs in general.

Conversely, the additional data generated by CTGAN had a significant positive impact on the result of kNN imputation. Not only does the additional data strenghtens kNN imputation's position as the most prominent method in the study in terms of predictive performance, but it also serves as the most significant contribution from this report as the methodology has not been examined in previous research. Further, the practical feasibility of the method combined with its strong results makes it suitable for practical applications. To sum up, the findings underscore the potential for further enhancements in data imputation using GANs.

Keywords

Missing Values, Data Imputation, Generative Adversarial Network, GAIN, CTGAN

Sammanfattning

I dagens samhälle är organisationer i allt större utsträckning beroende av dataanalys i sin beslutsprocess, vilket resulterat i att datakvalitet blivit ett område som fått allmer uppmärksamhet. Inom detta område utgör saknade värden en betydande utmaning som påverkar datans användbarhet negativt. Idealt bör data samlas in på ett sätt så att saknade värden undviks, men detta är ofta ogenomförbart på grund av praktiska och kostnadsmässiga begränsningar. Som ett resultat har olika metoder utvecklats för att hantera problemet med saknade värden. Istället för att radera ofullständiga observationer och därmed kompromissa med urvalsstorleken är det möjligt att fylla i de saknade värdena, och på så sätt förbättra datans användbarhet och prediktionsförmåga. Att tillämpa denna metod är dessutom en relativt enkel process sett till kostnad och arbetsinsats.

Utöver detta har även Generativa motståndarnätverk (GANs) på senare tid fått ökad uppmärksamhet som ett genombrott inom maskininlärning och erbjuder nya möjligheter inom datahantering. Denna studie undersöker två aspekter där GANs har potential att förbättra ifyllnaden av saknade värden. För det första utförs en omfattande utvärdering där prestandan hos en GAN-modell som fokuserar på att fylla i saknade värden, GAIN, jämförs med andra etablerade metoder. För det andra utvärderas effekten av att addera syntetisk data genererad av en GAN-modell vid namn CTGAN i träningsdatan för metoderna för ifyllnad av saknade värden.

Resultatet visar att andra metoder för ifyllnad av saknade värden presterade bättre än GAIN. GAINs potential ifrågasätts dock inte, eftersom ytterligare optimering av hyperparametrar samt utveckling av en nätverksstruktur som är specifik för datamängden förväntas förbättra dess prestanda. Det är dock viktigt att notera att denna studie visar på utmaningarna i de tidskrävande tränings- och optimeringsprocesserna.

Däremot hade den extra träningsdatan som genererats av CTGAN en signifikant positiv effekt på resultatet för metoden 'kNN imputation'. Inte bara var denna kombination den mest framstående metoden i studien när det gällde prediktionsförmåga, utan den utgör också det mest betydande bidraget från denna rapport då metodiken inte undersökts i tidigare forskning. Dessutom medför metodens praktiska genomförbarhet i kombination med dess starka resultat att den är fördelaktig för praktiska tillämpningar. Sammanfattningsvis understryker resultatet potentialen för ytterligare förbättringar för ifyllnad av saknade värden med hjälp av GANs.

Nyckelord

Saknade Värden, Data Imputation, Generativa motståndarnätverk, GAIN, CTGAN

Acknowledgements

We express our gratitude to our academic mentor Ted Kronvall at the Department of Mathematical Statistics at Lund University, who provided us with insightful guidance and support throughout our research journey. Additionally, we would much like to thank Marcus Zethraeus, our industry mentor from Predli AB, for his contagious enthusiasm and valuable perspectives on the problem at hand. Lastly, we would like to acknowledge the entire team at Predli for their engagement and for introducing us to the subject matter, which ultimately made this thesis project possible.

Contents

1 Introduction

1.1 Problem Statement

1.1.1 The Problem of Missing Values in Data Analysis

As the importance of big data increases for organizations, the complexity of data pipelines has correspondingly grown. High-quality data is crucial for organizations to derive meaningful insights from their data analysis, but ensuring such data standards could be difficult as the data amounts increase. As a result, the challenge of monitoring and improving data quality has become one of the key focus areas in modern database management systems and other data collection applications today, where the goal is to automatically detect and resolve potential data quality errors before the data is further analyzed [53] [28].

By ensuring high data standards within data pipelines, organizations can guarantee that the insights derived from the data are reliable and accurate, thus enabling informed decision-making. Poor data quality can on the other hand imply increased costs for organizations due to incorrect analyses, which could result in missed opportunities for growth and development or misguided decisions [73] [28]. Despite the negative consequences of poor data quality and the recent improvements in data monitoring and improvement tools, not all data quality issues are possible to solve automatically [5]. One of the most common data quality issues in data pipelines today, an issue which also can be difficult to properly solve automatically, is the occurrence of missing values [24] [34].

Missing values in a data set can be defined as values which are expected and that would have added additional information if they were available, yet that are not [61]. The occurrence of missing values in a data set can happen due to a variety of reasons, such as incomplete data transfers, participant non-response in survey data, faulty equipment or any other factors which causes unrecorded data. Regardless of the explanation, missing values present several challenges when it comes to extracting knowledge from a data set. This issue makes analysis more difficult as it limits the use of many machine learning models and estimators, and it can also result in incorrect conclusions [52] [18] [39].

1.1.2 Dealing with the Issue of Missing Values for Tabular Data Sets

Fortunately, there are several ways to handle the issue of missing values. The strategies must generally be adapted to the type of data where the missing values are encountered, and this thesis will thus only refer to missing values in tabular data sets, meaning data structured in rows and columns which is presented in a table. That being said, the ideal way of handling missing data for all data types is by collecting new data and ensuring that no data is missing. However, this is practically impossible in most cases [67].

Another possible way of dealing with missing data in tabular data sets is by applying list-wise deletion. This method implies deletion of the data points for which at least one feature value is missing, and is the most common way to deal with missing values in tabular data sets in practise today [30]. While the approach ensures efficient data pipelines, it typically implies a

loss of information and it may result in ,i.e., inadequate amount of data in order to pursue data analysis. As a consequence, it could hinder the possibility to reach conclusions about the data set [57].

Data imputation, i.e., filling in plausible values for the missing data, is an alternative way of handling missing values which does not reduce the amount of available data and thus have gained interest in recent research [28]. Data imputation done in an appropriate way can both reduce bias as well as improve prediction accuracy [51] [9]. There exists a large number of different imputation methods which differs in efficiency, implementation complexity and accuracy, and getting an overview of which method that is most preferable can thus be somewhat difficult [61].

As generative artificial intelligence and deep learning has increased in popularity within research in general, its ability to perform missing data imputation has also gained traction [74]. More specifically, the ability of Generative Adversarial Networks, GANs, to improve imputation performance has become an area of focus - both in terms of 1) Using GANs for improving the actual imputation [74], but also of 2) Using GANs to increase the training data amount in order to improve the performance of imputation methods [32].

Despite the recent increase in interest, gaining an overview of how GANs can be used in both these ways simultaneously, as well as how the GAN-based imputation method compare to other imputation methods is difficult. This is both due to a lack of research on some aspects of the topic, but also due to a variation of the study setups in the research that has been conducted [74], [69], [28], [32], [47]. This study is aiming to help close this gap in research.

1.2 Previous Research and Related Work

1.2.1 Imputation Methods

As previously mentioned, a wide variety of missing data imputation techniques for tabular data have been developed throughout the years. On a high level, the methods can be split into two categories, discriminative methods and generative methods [69].

Discriminative methods Discriminative imputation methods aim at using information extracted from the available data to predict the missing values. Mean and Median Imputation for continuous variables together with Mode Imputation for categorical variables are generally considered the most simple discriminative methods for imputation. The methods work exactly as their names describes, and replace the missing values of a particular column with the mean, median or mode of the non-existing values in this column. While these methods are easy to implement, they generally underestimate the variance in the data and does not take into account the correlation between features [50].

Nevertheless, there exists a range of other more advanced discriminative imputation methods, which have proved to perform better than Mean, Median, and Mode Imputation [66]. Some examples includes MICE [65]; an iterative imputation method based on a series of regression models, MissForest [59]; a non-parametric iterative random forest-based model and kNN imputation [63]; a method using the values an observation's neighbors to impute the observation's

missing values.

Generative methods Generative imputation methods aim to simulate the missing values by utilizing the joint distribution of observed and unobserved values. Examples of generative methods include algorithms based on Expectation-Maximization [19], DAEs (Denoising Auto-Encoders) [20] and GANs (Generative Adversariral Networks). While all methods have proved to be successful, methods based on GANs have shown superior performance [79] [74], which includes methods such as GAIN (Generative Adversarial Imputation Nets) [74], WGAIN (Wasserstein GAIN) [17] and a number of other extensions based on the original GAIN [47].

The imputation method GAIN was initially presented in the paper "GAIN: Missing Data Imputation using Generative Adversarial Nets" by Yoon et. al in 2018 [74]. The method was then shown to outperform other discrimative and generative state-of-the-art imputation methods, such as MICE, MissForest, Matrix Completion, Denoising Auto-Encoders, and Expectation Maximization, both with regards to imputation accuracy as well as to prediction performance [74]. Another study, Dong *et al.* [12] instead showed that GAIN showed similar performance as MissForest when the simulated miss rate was up to 20%, yet was the best performing model for miss rates above 50%.

Despite the result of the above-mentioned studies, other papers such as "A benchmark for data imputation methods" by Jäger *et al.* [28], have on the other hand shown that GAIN is not the ideal imputation method when analyzing a large number of heterogeneous data sets in a study splitting the data into test and training sets, with evaluation in terms of both imputation performance and prediction tasks. Jäger *et al.* instead conclude that GAIN is outperformed by all other models in the study, among other methods, kNN imputation and Random Forest-based imputation in terms of both imputation and predictive performance, and that the Random Forest-based method showes the overall strongest performance. Part of this result is additionally confirmed by Lalande *et al.*, whom conclude in the study "Numerical data imputation: Choose kNN over deep learning" [36] that kNN imputation outperforms GAIN. To sum up, the opinions regarding the superiorness of GAIN compared to other imputation methods differ.

1.2.2 Other Approaches to Improve Performance of Imputation Methods

A completely different approach of using GANs to improve data imputation, compared to the approach used in GAIN, was presented in the paper "Mixed Data Imputation Using Generative Adversarial Networks" by Khan *et al.* in 2022 [32]. The authors investigated if the Generative Adversarial Network-based method CTGAN, which is a model aiming to synthesize tabular data, could improve standard imputation methods such as MICE, MissForest, and DAEs. The authors proved that increasing the amount of training data using CTGAN improved the imputation performance in terms of imputation accuracy.

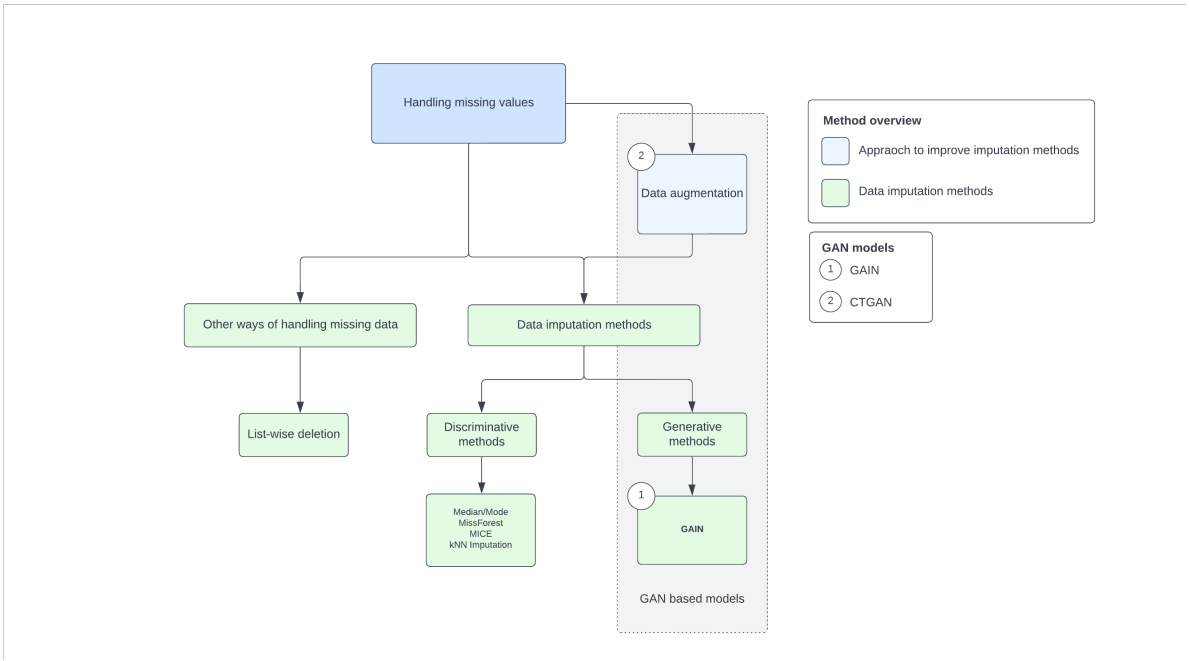

Figure 1: Illustration of how generative models that can improve data imputation, noting that other approaches may also exist.

1.2.3 GANs to Improve Imputation Methods

To conclude, there are different ways in which GANs is suggested to potentially be able to improve missing data imputation, either as 1) A foundation in the imputation method, e.g. as suggested by Yoon *et al.* in GAIN [74], or as 2) A way of increasing training data amount in order to improve other imputation methods, e.g. by using CTGAN as suggested Khan *et al.* [32]. An overview of these different approaches is presented in Figure 1.

Despite the existence of the different methods, there is a lack of an extensive overview of how GANs can help improve data imputation in the current research. Firstly, as presented in Section 1.2.1, the views in literature regarding if GAIN outperforms other imputation methods or not, differ. Secondly, it is not certain whether the two different approaches of using GANs for data imputation presented in Figure 1, can improve imputation performance if being used together, since this question has not been researched in literature. Lastly, the approach of adding additional training data generated by CTGAN deserves a more rigorous examination to establish a more robust comprehension of its potential.

1.3 Thesis Research Questions and Scientific Contributions

1.3.1 Aim of Thesis and Research Questions

The overall aim of the thesis is to explore if, and how, Generative Adversarial Network can improve missing data imputation for tabular data, focusing on two different approaches as well as the relationship between them. These approaches include 1) Using GANs as a foundation

in an imputation method, as in GAIN, and 2) Using GANs for improving the performance of any imputation method, through increasing the amount of training data using CTGAN for synthesizing tabular data. The aim of the thesis will be fulfilled by answering two research questions, each with more specific sub questions.

Research question 1 How does GAIN compare to standard imputation methods when being evaluated in an extensive way?

- Sub question 1a) How does GAIN perform compared to standard imputation methods (Median/Mode Imputation, MICE, kNN imputation, MissForest) in a study in which:
    1. The missing values are introduced in a way that resemble real life situations for both categorical and numerical variables,
    2. A training and test data set is used to train and evaluate the models,
    3. The hyperparameters of GAIN are tuned for each data set using cross-validation,
    4. A variety of data sets with different ratios of numerical and categorical variables is imputed and evaluated,
    5. The models are evaluated in terms of both imputation performance, execution time and predictive performance,
    6. The presented results includes standard deviation (whenever feasible for practical applications[1]), and
    7. The models are compared to list-wise deletion, the most common way to handle missing data in practise today
- Sub question 1b) How does the original GAIN perform compared to a version of GAIN more adapted to categorical data, and how does this new version of GAIN in turn perform compared to other imputation methods when being evaluated in an extensive way?

Research question 2 How can the approach of increasing training data using CTGAN improve imputation methods?

- Sub question 2a) Which imputation method (Median/Mode Imputation, MICE, kNN imputation, MissForest, GAIN) exhibit the greatest improvement in imputation accuracy and predictive performance when the amount of training data has been increased using CTGAN?
- Sub question 2b): How does the specific amount for which the training data is increased with CTGAN impact the imputation accuracy and predictive performance?

[1]This will be the case for all methods with variability except MICE, due to considerably longer execution times than other methods.

1.3.2 Scientific Contributions

The scientific contributions of the thesis can be structured based on the two research questions.

Research question 1 This study aims to provide an extensive evaluation of how GAIN compare to other standard imputation methods. As mentioned, there have been other studies aiming to compare GAIN to other imputation methods but there are clear differences in the setups between the studies. A selection of these studies is presented in Table 1. The selection of these four studies is based on their coverage of a wide range of imputation methods, their representation of diverse study setups found in current literature, and their specific focus on tabular data, which aligns with the main objective of this study.

Study setup component	**Color coding explanation**		**This thesis**	**Yoon *et al.* [74]**	**Dong *et al.* [12]**	**Jäger *et al.* [28]**	**Lalande *et al.* [36]**
Models compared			Listwise deletion, Median/Mode, kNN imputation, MICE, MissForest, GAIN	MICE, MissForest, Matrix Completion, DAE, EM, GAIN	MICE, MissForest, GAIN	Median/Mode, kNN imputation, Random Forest, Discriminative Deep Learning, VAE, GAIN	kNN imputation, misGAN, GAIN
Conclusion about GAIN compared to other methods				GAIN is performing better	GAIN is performing better	GAIN is performing worse	GAIN is performing worse
Data sets used	*Even split of categorical / numerical features*	*Majority of numerical features*					
Use of training and testing data	*Yes*	*No*					
Downstream prediction after imputation	*Yes*	*No*					
Hyperparameter tuning based on used data sets	*Yes*	*No*					
Way of introducing missing values for categorical variables	*After one hot encoding*	*Before one hot encoding*					
Cross-entropy loss function for categorical variables in GAIN	*Yes*	*No*					

Table 1: Overview of four studies comparing GAIN with other imputation methods

As seen in Table 1 the studies conducted by Jäger *et al.* [28] and Lalande *et al.* [36] claims GAIN is outperformed by other standard imputation methods. These studies distinguishes themselves on several factors compared to the studies conducted by Yoon *et al.* [74] and Dong *et al.* [12], which claim that GAIN is superior. In the first mentioned two studies, the authors does not fully optimize the GAIN model, either by not tuning the hyperparameters for each data set or by not introducing a cross-entropy loss function. Further, they evaluate GAIN in a more realistic real life setting by introducing missing values before one hot encoding the categorical variables in the data sets, and in addition they are also either using a wider range of different types of data sets or using train and testing data for their model evaluations. All these are factors which could be the explanatory reasons as to why GAIN performs worse in the two studies.

To conclude, the scientific contribution of research question 1a) is A) Providing an extensive evaluation of GAIN compared to other imputation methods as well as B) Conducting a study with realistic conditions. A) will be fulfilled by conducting a study in which train and testing data is used, the hyperparameters for each data set are tuned and where cross-entropy is included in the loss function. B) will be fulfilled by using data sets with a variety of numerical and

categorical features, and by introducing missing values in a realistic way before one hot encoding the categorical variables. Conducting the study would fill a gap in literature since no previous study comparing GAIN with other imputation methods study has fulfilled both A) and B).

Lastly, as presented in Table 1, the studies suggesting GAIN is performing worse are differing on aspects related to the handling or inclusion of categorical features, compared to the research papers concluding the opposite. Thus, by answering research question 1b) this study is also suggesting one version of GAIN which is more adapted to categorical data. This extension will include an additional activation function, a tuning parameter for the categorical loss as well as an updated rounding function. Comparing this version with the original GAIN model is another scientific contribution of this study.

Research question 2 The impact of using additional synthetic training data generated by CTGAN for data imputation accuracy has only been investigated by Khan *et al.* in 2022 [32]. The study has several research gaps that require further investigation and attention, gaps which this study is aiming to fill. Firstly, the research does only evaluate performance in terms of RMSE (Root Mean Square Error) and not by prediction accuracy. Secondly, neither kNN imputation or GAIN are part of the imputation method alternatives in the study, and the effect of increased training data for the generative network based imputation method GAIN has therefore never been evaluated. Using GANs with limited data or small data sets typically leading the training to diverge [31]. Thus investigating the influence of augmented training data on GAIN presents an intriguing avenue for further exploration. Lastly, the amount of additional data is limited to 100% additional data, leaving the impact of incorporating higher amounts of additional data unexplored. The first two aspects will be answered by research question 2a) and the last aspect will be answered by research question 2b).

1.4 Structure of Thesis

The remaining part of the thesis is structured as follows. Section 2 provides an overview of the theoretical background required to understand the research questions and the proposed methodology. On a high level, this includes an overview of mechanisms of missing values in tabular data sets, an introduction to GANs in general, a deep dive of the imputation methods used in this thesis as well as an overview of the data synthetization method CTGAN. The methodology of the study including the data and evaluation criteria used is presented in Section 3. Section 4 presents the results of the study, and in Section 5, those results are discussed and analyzed. Lastly, in Section 6 the conclusions of the study are presented together with suggestions of future work.

2 Background

This chapter will cover the theoretical background of this thesis. Firstly, the theory of topics related to missing values in tabular data will be presented. This includes an introduction to tabular data, the theory behind the different missing data mechanisms for tabular data as well as an introduction to tabular data imputation and synthesis. After this, an introduction to GANs will be presented. Here, the prerequisites needed to understand both the imputation method GAIN as well as the synthetization method CTGAN, which will be covered in later sections, will be covered. This leads to a section presenting the theoretical background to the data imputation methods used in the empirical study of this thesis. This includes an introduction to the discriminative methods such as Median/Mode Imputation, MICE, MissForest, and kNN imputation, as well as to the generative GAN-based method GAIN. In the final section, the theoretical background of the method for generating synthetic tabular data, CTGAN, will be presented.

2.1 Missing Values in Tabular Data

2.1.1 Introduction to Tabular Data

Tabular data refers to data structured in rows and columns, presented together in a table. Every record, i.e., every observation, is stored as a row while feature characteristics for a specific observation are stored as column values on the particular row. The dimensions are the same throughout all rows so that each observations have the same number of feature characteristics, with the columns in the same order.

2.1.2 Handling of Different Tabular Data Types

A table can contain data of various data types, yet a particular column only contains values of the same datatype. The different data types includes categorical data, meaning that every value can only take on values from a category, or numerical data, where the numerical data can be classified as either continuous or discrete data [11]. The numerical data can generally be analyzed after only minor data pre-processing such as for example normalization, however for categorical variables more drastic transformations such as encoding to numerical values is usually required [10].

Normalization of numerical variables Normalization implies scaling all values of a column in a data set to a particular range, usually to the range [0,1] or to the range [-1,1] [1]. The process of normalizing is required to achieve accurate results when applying statistical or machine learning methods when there are significant differences in the ranges of the values of different feature variables [1].

Encoding of categorical variables Encoding of categorical variables is generally required as many statistical and machine learning method only can handle numerical values. There are

a number of ways in which the encoding can be executed [10]. Two common ways, which will be used in this thesis are label encoding and one-hot encoding. To illustrate, $\boldsymbol{X}$ is assumed to be an $(n \times p)$-dimensional data matrix, where X_{is} is the value of the i^{th} observation's value of the the s^{th} feature. This leads to that $\mathbf{X}_i = (X_{i1}, \cdots, X_{ip})$ represents one observation. The two mentioned encoding techniques can then be described as follows:

- **One-hot encoding.** One-hot encoding implies transforming each categorical column to several binary columns, each binary column representing one category alternative of the initial category. A value of 1 represents the presence of the particular category alternative, while a value of 0 represents the absence [10]. Formally, this means that given that $\boldsymbol{X}$ has a categorical feature for index s, the value X_{is} for observation i is then transformed into a dummy vector $X_{is} = (X_{is1}, X_{is2}, ..., X_{isd_s})$ where d_s is the number of possible alternative that the categorical feature s can take, and the variables X_{isd_s} are binary.

  Example: A dummy vector X_{is} for a feature s with 3 possible categories can be expressed as $X_{is} = (1, 0, 0)$ if X_{is} takes the value with an index 1. Overall, all possible choices for X_{is} for the 3 categories can be summarized according to:

category	X_{is1}	X_{is2}	X_{is3}
1	1	0	0
2	0	1	0
3	0	0	1

  (1)

  where a 1 in the column X_{isd_s} indicate that the observation i takes the value with an index d_s for the feature s, and a 0 in the column means that the observation does not take the value. This means that the column of the feature s is now transformed to 3 columns for all observations [16].

- **Label encoding.** Label encoding implies that each categorical alternative within a certain category is transformed to the integer. If the categories are ordered, the integer sequence should reflect this ordering [10]. Formally, this means that an observation $\mathbf{X}_i = (X_{i1}, \cdots, X_{ip})$ keeps the same dimensions, although the value X_{is} is transformed to an integer if s is a categorical feature.

2.1.3 Missing Data Mechanisms for Tabular Data

Missing values in a data set can be defined as values which are expected and that would have added additional information if they were available, yet that are not [61]. Real-world tabular data sets often contains missing values and the handling of these is a crucial challenge for data scientists [15]. There are a number of reasons as to why data might be missing, for instance reasons such as improper data entries, data unavailability, and data collection issues [46]. Data can be missing according to three missing data mechanisms defined by Rubin in 1976 [54].

Missing completely at random (MCAR) Data is considered MCAR when the probability of missing data on a variable X is independent of the variable X itself and independent of other variables in the data set. In a modern context and in practical situations, MCAR is often considered unrealistic.

Missing at random (MAR) Data is considered MAR when the probability of missing data on a variable X is not dependent on the variable X itself, but dependent on other variables in the data set. The standard assumption is that the data is missing according to the MAR mechanism [15].

Missing not at random (MNAR) Data is considered MNAR when the probability of missing data on a variable X is dependent on the variable X itself.

2.1.4 Approaches for Handling of Missing Values in Tabular Data

There are three main approaches encountered in literature for handling missing values in data sets. The first method, referred to as list-wise deletion, is a frequently used technique that involves the removal of rows containing at least one missing value. The second is to adapt the specific data task to tolerate missing values, however this approach will be out of scope for this thesis. The third approach is to impute the missing data [52] [15].

Listwise deletion Listwise deletion is the most common way to handle the issue of missing values in real-world applications today, and implies deletion of every observation for which at least one feature value is missing [30]. The approach is a default method for many standard data analysis packages, such as by the *lm* and the *glm* command in R as well as the *regress* function in Stata. However, the approach comes with the negative aspect of often significantly reducing the sample size, omitting important information. This is especially the case when the number of features is large, since the probability of one observation having at least one missing value then potentially increases [68].

Imputation of missing values Imputing missing values involves replacing absent data with approximated or inferred values using a statistical technique [15]. When selecting an imputation method, knowing the missing data mechanism is crucial. An incorrect assumption whether data is missing at random can affect the proportions of certain groups in the data and introduce bias [60]. A more detailed introduction to different imputation methods will be covered later in this thesis.

2.1.5 Synthesizing Tabular Data

The utilization of synthetic data origins from the inception of computer vision in the 1960s, but the common problem of insufficient amount of data in modern AI has increased the degree of interest in the subject today. Especially deep-learning models are in many cases hindered by

the lack of large enough training data sets, an issue which synthetic data can provide a solution to [44].

Synthetic data is artificially produced data and could either be created from scratch based on descriptions of the data distributions, or based on current data to produce more novel and diverse training samples. This approach can not only extend the data set in size, but also filling in gaps in the data distribution, supplementing situations that are missing or occurs rarely in the original data set [48].

Some issues with synthesizing tabular data include that tabular data sets often contains feature characteristics with various data types that needs to be handled in different ways during synthesizing, as well as that the continuous values of a tabular data set often have complex probability distributions, which may be difficult to sample new synthetic data from. The latest research in the field of data science have led to the development of synthetic data generation techniques that involve treating each column in a table as a random variable. A joint multivariate probability distribution is modeled based on the data and the sampling is subsequently performed from that distribution [72].

2.2 Generative Adversarial Networks

This section covers the fundamental theory of GANs. The GANs presented in this section will not be used in the empirical studies of this thesis, however the theory presented includes knowledge prerequisites for the subsequent sections, which will more in detail explain the imputation method GAIN and the synthesis method CTGAN, which are both based on GANs.

2.2.1 Introduction to GANs

A Generative Adversarial Network (GAN) is an architecture used to generate data samples with the same statistical distribution as the input data. The framework was originally introduced by Ian Goodfellow *et al.* in 2014 [21]. The model consists of two neural networks, one generator network, G, and one discriminator network, D. The aim of the generator network, G, is to generate data that the discriminator network, D, will classify as real data. The aim of the discriminator network, D, is to correctly classify real data and generated data [21].

2.2.2 Objective and Loss Functions

The initial input for the generator network, denoted as G, will be a noise variable $\boldsymbol{z}$ from a prior distribution $p_{\boldsymbol{z}}(\boldsymbol{z})$, which usually is a uniform or Gaussian distribution. A mapping to a data space is represented as $G(\boldsymbol{z}; \theta_g)$, resulting in samples $\hat{\boldsymbol{x}}_{\boldsymbol{d}}$. The generator G is a multi layer perceptron neural network and thus uses the parameters θ_g to produce a set of outputs based on a corresponding set of inputs. To trick the discriminator network, G tries to make p_g, i.e., the distribution from which the data points $\hat{\boldsymbol{x}}_{\boldsymbol{d}}$ are sampled, resemble the unknown distribution p_d of the real data $\boldsymbol{x}$. More specifically, G aims to minimize the probability that D succeeds in guessing that $\hat{\boldsymbol{x}}_{\boldsymbol{d}}$ is drawn from p_g rather than p_d, which implies an objective function that minimizes its loss function, described as:

$$\min_G \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[\log(1 - D(G(\boldsymbol{z})))] = \min_G \mathbb{E}_{\hat{\boldsymbol{x}}_{\boldsymbol{d}}\sim p_g}[\log(1 - D(\hat{\boldsymbol{x}}_{\boldsymbol{d}}))] \tag{2}$$

The discriminator network, D, is also a multi layer perceptron neural network represented as $D(\boldsymbol{x}; \theta_d)$, where θ_d are the parameters of the network and $\boldsymbol{x}$ representing either a real data sample or data generated by G. The discriminator network, D, outputs a single scalar and $D(\boldsymbol{x})$ represents the probability that $\boldsymbol{x}$ comes from real distribution p_d and not from the distribution from which the generator samples, p_g. The generator's loss function (2) is minimized by making the discriminator assigning a probability of $D(G(\boldsymbol{z}))$ as close to 1 as possible. The optimal discriminator will give a high probability when $\boldsymbol{x} \sim p_d$ and a low probability when $\boldsymbol{x} \sim p_g$. The discriminator therefore has an objective function which is maximizing its loss function, described as

$$\begin{aligned} &\max_D \mathbb{E}_{\boldsymbol{x}\sim p_d}[\log(D(\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{z}\sim p_z}[\log(1 - D(G(\boldsymbol{z})))] = \\ &\max_D \mathbb{E}_{\boldsymbol{x}\sim p_d}[\log(D(\boldsymbol{x}))] + \mathbb{E}_{\hat{\boldsymbol{x}}_{\boldsymbol{d}}\sim p_g}[\log(1 - D(\hat{\boldsymbol{x}}_{\boldsymbol{d}}))] \end{aligned} \tag{3}$$

The networks are simultaneously playing a minimax game of a value function $V(G, D)$, leading to the combined objective function defined as

$$\min_G \max_D V(D, G) = \min_G \max_D \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{4}$$

The above reasoning leads to the loss function $\mathcal{L}_d$ for the discriminator D and $\mathcal{L}_g$ for the generator G

$$\begin{aligned} \mathcal{L}_d &= \log D(\boldsymbol{x}) + \log(1 - D(G(\boldsymbol{z}))) \\ \mathcal{L}_g &= \log(1 - D(G(\boldsymbol{z}))) \end{aligned} \tag{5}$$

At convergence, the generator network G will sample from the true data distribution, $p_d = p_g$. If this state is reached, $D(\boldsymbol{x}; \theta_g) = \frac{1}{2}$ since the discriminator then is guessing whether the data sample $\boldsymbol{x}$ is real or generated [21].

2.2.3 Training of GANs

Stochastic Gradient Descent Neutral networks are trained using gradient back-propagation in an iterative manner. Gradient back-propagation involves the process of computing the gradients of the loss functions using the chain rule of differentiation, which implies that the gradients can be propagated back through the network from the final loss to the initial inputs. This implies that the discriminator D is trained as a first step to update the parameters θ_d by using the gradient $\nabla_{\theta_d}\mathcal{L}_d(\theta_g, \theta_d)$. The generator G is subsequently trained to update θ_g using the gradient $\nabla_{\theta_g}\mathcal{L}_g(\theta_g, \theta_d)$ [21].

Stochastic gradient descent is generally the standard optimization algorithm for GANs, which updates the parameters according to:

$$\theta_{n+1} = \theta_n - \gamma \nabla_{\theta_n} f\left(\mathbf{x}; \theta_n\right), \tag{6}$$

where γ is a learning parameter which determines the step-size and $f\left(\mathbf{x}; \theta_n\right)$ represents the loss functions of G and D. To reduce the computational complexity, mini-batch stochastic variation of gradient descent is commonly used, where the samples are split into mini-batches of size m. The parameters are then updated based on the gradients of the mini batches according to [21]:

$$\theta_{n+1} = \theta_n - \gamma \nabla_{\theta_n} \frac{1}{m} \sum_{i=1}^{m} f\left(\mathbf{x^{(i)}}; \theta_n\right) \tag{7}$$

It can be proved that by using the batch stochastic gradient descent algorithm, GANs theoretically converges to where $p_d = p_g$ and $D\left(\boldsymbol{x}; \theta_g\right) = \frac{1}{2}$, which corresponds to sampling from the true distribution [21].

Adam Optimizer Algorithm The Adam algorithm, used to optimize stochastic objective functions, is widely used in training GANs. Adam was introduced to fit large data sets with sparse gradients. Estimates of first and second moment of the gradients are used to apply adaptive learning rates for the different parameters of the generator and discriminator network [33].

Batch Normalization Batch normalization is a normalization technique that is applied between the layers of a neural network. It has shown to speed up training with higher learning rates and less concern about initial parameter initialization. For a mini-batch of size m with values $\mathcal{B} = \{x_1, ..x_m\}$, the output y_i is defined as

$$y_i \leftarrow \gamma \left(\frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \beta \equiv \mathrm{BN}_{\gamma,\beta}\left(x_i\right) \tag{8}$$

where γ and β are the model parameters and ϵ a constant added for numerical stability. [27].

Activation Functions An activation function is used in a neural network to introduce non-linearity into the output of a neuron. Common activation functions include the `sigmoid`, `ReLU`, `Leaky ReLU`, `tanh`, and `softmax` function. The `softmax` activation function has traditionally been used to output categorical or discrete values, by interpreting the values as probabilities and then use the `argmax` function to determine the label with highest probability. One issue with this approach is that the `argmax` function can not be differentiated and therefore no back-propagation can be done to train a neural network using this activation function.

Gumbel-Softmax The Gumbel-Softmax distribution was introduced by Jang *et al.*[29] and Maddison et al [40] to allow for differentiable sampling from a categorical distribution, and thus solves the issue previously mentioned. Both authors proved that the Gumbel-Softmax

outperforms other estimators for categorical variables. If the distribution of a categorical feature X is represented by π, the samples are drawn from

$$x_k = \frac{\exp\left(\left(\log\left(\pi_k\right) + G_k\right)/\tau\right)}{\sum_{i=1}^{n} \exp\left(\left(\log\left(\pi_i\right) + G_i\right)/\tau\right)} \text{ for } k = 1, 2, \ldots, n \tag{9}$$

where π_k is the probability that $X = X^k$, G_k is an independently drawn sample from the Gumbel distribution, and where τ represents the hyperparameter temperature. As $\tau \longrightarrow 0$, the sampling will approaches a discrete `argmax` sampling, namely a one-hot representation.

2.2.4 GAN Convergence Issues

Several issues has been reported when making the original GAN framework converge to $p_d = p_g$, which corresponds to sampling from the true distribution.

- **Mode collapse** Mode collapse can occur when dealing with multi-modal data distributions and is causing the generator to only produce samples from one of the categories or modes.
- **Vanishing gradient** The issue of vanishing gradients arises when the discriminator D reaches a high level of proficiency in determining a real or generated sample from G, which fails in producing sufficiently convincing samples, leading to a near-zero gradient. If the gradient is small, it implies that the weights and biases of the early layers will not be updated efficiently. This situation blocks the networks from further learning [70].

Because of these issues, there are many more stable variants of GANs proposed since the original paper was released, including the WGAN [3], WGAN-GP [23], and CTGAN [72] networks described in later sections.

2.2.5 Challenges in Using GANs for Modeling Tabular Data

Further, there are several properties that make GANs in particular challenging to apply in tabular data generation tasks, listed by Xu *et al.* in the paper "Modeling Tabular data using Conditional GAN", presenting an extension of the GAN model called CTGAN [72].

- **Variety of data types.** Tabular data sets usually consists of a mix of discrete and continuous columns, with a variety of data types such as integers, floats, booleans, and strings. There is accordingly a need for different activation functions in the output layer of the network. Categorical or discrete values are most often one-hot-encoded and using the activation function `softmax`, which leads to that the generator produces a probability for each category. The issue is that this probability is most often continuous and not an exact 0 or an exact 1, which makes it very easy for the discriminator to separate real data from generated data. This since the real data is represented as a proper one-hot vector, i.e, an exact 0 or an exact 1, while the generated data is a continuous probability value between zero and one.

- **Non-Gaussian distributions.** In the image data set usually modelled by the original GAN, each data point takes a value between 0 and 255 and follow a Gaussian-like distribution which can be normalized to $[-1, 1]$, usually using a `tanh` function. However, continuous values frequently found in tabular data usually follow a non-Gaussian-like distribution, leading to the issue of vanishing gradients described in 2.2.4.

- **Multi-modal distributions.** Tabular data tend to be highly multi-modal, having several modes in the distribution. Srivastava et al [58] proved that the original GAN framework was not able to model all modes on a simple 2D data set, indicating that a more advanced technique is needed to model complex continuous distributions.

- **Imbalanced categorical columns.** Categorical columns often exhibit a high degree of imbalance, wherein a specific value occurs more frequently and thus becomes over represented. This results in that if missing a minor category the data distribution is not notably affected, making which is hard for the discriminator to notice the minor categories as well as poor training possibilities for these categories. However, maintaining these categories in the synthesized data can still be important.

2.2.6 The Importance of Abundant Training Data

GANs have been shown to perform poorly in the case of limited amount of training data. With a small amount of data during training, the discriminator often ends up over-fitting the data, leading the training to diverge. To combat this, data augmentation can be used, with the aim to use more augmented data while maintaining the sampling from the true original distribution. One solution presented by Zhao et al. is Differentiable Augmentation (DiffAugment) which generated differentiable augmentations on both real and fake samples, instead of directly augment the training data [77]. Another solution proposed by Karras et al. is an adaptive discriminator augmentation mechanism, shown to significantly stabilizes training in limited data contexts [31]. NT Tran et al present a framework termed Data Augmentation Optimized for GAN (DAG) [62] to enable the use of augmented data in GAN training.

2.3 Data Imputation Methods

This section will cover the theoretical foundation to the imputation methods used in the empirical study of the thesis. The methods covered includes the discriminative methods Median/Mode Imputation, MICE, MissForest, and kNN imputation as well as the generative GAN-based method GAIN.

2.3.1 Discriminative Methods

2.3.1.1 Mean, Median, and Mode Imputation Mean, median, and mode imputation are frequently used so-called single imputation methods, which means that a single value is used to replace the missing values. Mean imputation implies replacing the missing values in one feature column of a tabular data set with the mean of the non-missing observations in this

column. The process is performed for all columns, replacing each missing value in the data set with the mean of the corresponding column. Median imputation works very similarly, but the missing values are imputed with the median of the non-missing observations in the associated column [42].

Mean imputation ensures that the mean of the distribution remains the same before and after imputation. However, it have been shown that the imputation method typically distorts other characteristics of the distribution since they ignore relationship between variables [76]. Median imputation could be a more appropriate method in case of a distribution with many outliers, however the imputation method tends to bias variances and co-variances towards zero as it reduces the variability between observations [56] [42].

Mode imputation implies imputing the most frequent value of the non-missing observation of the corresponding column. It is typically used for missing nominal and categorical data [56].

2.3.1.2 MICE MICE (Multivariate Imputation by Chained Equation) has emerged as a well-used method for handling missing data [65]. The method is based on multiple imputation, a Monte Carlo-based technique to deal with incomplete data sets in which several values are generated for each missing value. In other words, m different completed data sets are formed in which the missing values are imputed based on the distributions of the variables in observed data set, as well as the relationship between them. These m data sets are then analyzed and the results are combined into one imputed data set [55].

The high-level algorithm of MICE consists of a number of steps:

- Step 1. Missing values are replaced with initial placeholder values (usually the mean or mode of the variable column).
- Step 2. The first variable for imputation is chosen. All the placeholder values for this variable are set back to missing.
- Step 3. The missing values of the one variable chosen in Step 2 are estimated using a regression model, with all the other variables used as independent features in the model.
- Step 4. The missing values are replaced with the predictions from the regression model.
- Step 5. Steps 2-4 are iterated for each variable with missing data.
- Step 6. Steps 2-5 are repeated for a fixed number of cycles or until conversion.

The regression model in Step 3 is chosen based on the distribution of the variables. Different software packages have slightly different default methods, but continuous variables are typically modelled by a form of linear regression, binary categorical variables by logistic regression and other categorical variables by multinomial logit models [65] [4].

2.3.1.3 kNN Imputation The main idea of K Nearest Neighbor (kNN) imputation is that missing values are imputed based on an aggregation of the values of the K nearest neighbors in space. This entails that if a value of a particular feature is missing for one observation, this value is imputed based on the corresponding values of this particular feature for the observation's K nearest neighbors. The neighbors are determined based on the distance between observations, which in term is based on the similarity of the feature values between the different observations, for all features. This similarity is determined in various ways depending on if the feature is categorical or numerical [43] [16].

The overall procedure of kNN imputation can be described as follows [43]:

1. Choose metric for calculating distance between observations.

   The standard way of determining the distance between observations is usually by using the Euclidean distance, which also will be used in this study. The distance consider all neighbors to a particular observation equally important, and is used as standard in libraries such as Scikit-learn's kNNImputer and MATLAB's knnimpute [49] [26].

2. Calculate distance between observations.

   To illustrate, $\boldsymbol{X}$ is assumed to be an $(n \times p)$-dimensional data matrix where X_{is} is the value of the i^{th} observation's value of the the s^{th} feature. This leads to that $\mathbf{X}_i = (X_{i1}, \cdots, X_{ip})$ represents one observation. The formula of calculating the Euclidean distance between two observations $\mathbf{X}_i$ and $\mathbf{X}_j$ can for numerical variables then be found as [43]:

   $$d_{ij} = \text{dist}\left(\mathbf{X_i}, \mathbf{X_j}\right) = \sqrt{\sum_{s=1}^{p} \left(X_{is} - X_{js}\right)^2} \tag{10}$$

   where p is the total number of features.

   Categorical variables must be transformed into binary variables before calculating the distance between observations. This is typically done through one-hot encoding [16]. As described in Section 2.1.2 this entails that given that s is a categorical feature, a variable X_{is} is then transformed into a dummy vector $X_{is} = (X_{is1}, X_{is2}, ..., X_{isd_s})$ where d_s is the number of possible alternatives that the categorical feature s can take. The Euclidean distance between the observations can be then calculated according to a variation of equation (10), where an additional summation is added to sum over all possible feature values d_s, according to [25]:

   $$d_{ij}^{Cat} = \text{dist}^{Cat}\left(\mathbf{X}_i, \mathbf{X}_j\right) = \sqrt{\sum_{s=1}^{p} \sum_{c=1}^{d_s} \left(z_{isc} - z_{jsc}\right)^2} \tag{11}$$

3. Find optimal number of neighbors K.

   The optimal number of neighbors K for a data set is usually determined through performing cross-validation on the training data set [25] [64]. Several papers, such as Duda and Hart

[14] and Maier *et al.* [41] have proposed that the number of neighbors K should be smaller than the square root of the number of observations. One important aspect of kNN imputation to note is that the choice of K is critical for the accuracy of the method [75].

4. Estimate and impute the missing values.

   After determining the K nearest neighbors for each observation, the missing values can be calculated as the average of the neighbor's corresponding values [43]. One observation i with a missing value for feature s can be imputed as $\hat{X}_{is}$ according to:

$$\hat{X}_{is} = \frac{\sum_{k=1}^{K} X_k}{K} \tag{12}$$

$$X_k = X_{i=1\ldots N,s} \mid d_i \in \{d_1, d_2, \ldots, d_K\} \tag{13}$$

   where K is the optimal number of neighbors, d_1 is the shortest distance of neighbor, d_i accordingly the i'th rank in distance of neighbor and X_k the value of the feature s of the neighbors of observation i.

2.3.1.4 MissForest MissForest is another imputation method suited for mixed-type data sets with complex or non-linear relationships between the variables. The method is based on multiple imputation using the Random Forest algorithm and was introduced by Stekhoven and Buhlmann in 2012 [59].

The approach of the missForest algorithm can be described using an n x p-dimensional data matrix $X = (X_1, X_2, ..., X_p)$, where X_k represent an arbitrary feature. Initially, all the missing values in the data matrix are replaced by a placeholder value, usually using the mean of the other values of the respective features. All features X_k with missing values are then sorted based on the amount of missing values, starting with the feature with lowest miss rate, and are then one by one imputed by fitting a random forrest aiming to predict the feature, using the observed values of the other features as predictors.

More formally, one can assume that a feature X_k in the data set have missing values at the indices $i_{mis}^{(k)}$. This implies that the feature X_k will be the target variable that the random forest aims to predict, and the data matrix can accordingly be split into four different sets of observations with different notations.

1. $y_{obs}^{(k)}$ represents the non-missing values of the feature X_k.
2. $y_{mis}^{(k)}$ represents the missing values of the feature X_k.
3. $x_{mis}^{(k)}$ represents all the values with indices $i_{mis}^{(k)}$ of features other than X_k.
4. $x_{obs}^{(k)}$ represents all the values with indices $i_{obs}^{(k)} = \{1, ..., n\} \setminus i_{mis}^{(k)}$ of features other than X_k.

Thus, when predicting the missing values of a feature X_k, this means that a random forest model is initially trained on the data consisting of the target variable $y_{obs}^{(k)}$ with the predictors $x_{obs}^{(k)}$. This model is subsequently used to predict the missing values $y_{mis}^{(k)}$ using the predictors $x_{mis}^{(k)}$. This

Algorithm 1 MissForest algorithm

Require: Matrix X size $n \times p$, stopping criterion γ
  Sort the features in X based on miss rate, starting with feature with lowest miss rate. S $\leftarrow$ sorted vector of indices
  Do initial guess for missing values
  while not γ **do**
    $X_{old}^{imp} \leftarrow$ save matrix of imputed values last iteration
    for k in S **do**
      Fit a random forest with target variable $y_{obs}^{(k)}$ and predictors $x_{obs}^{(k)}$
      Predict $y_{mis}^{(k)}$ using $x_{mis}^{(k)}$
      $X_{new}^{imp} \leftarrow$ update the imputed matrix with the predicted values $y_{mis}^{(k)}$
    end for
    update γ
  end while
  return The imputed matrix X^{imp}

sub-process is repeated for all features containing missing values, and the overall process is then repeated until a converging criteria is met. According to Stekhoven and Buhlmann who first introduced the algorithm, the pseudo algorithm can be described as follows [59]:

2.3.2 GAIN

Generative Adversarial Imputation Nets (GAIN), originally proposed by Jinsung Yoon *et al.* in 2018 [74], is an generative imputation method that employs the GAN framework to impute missing values. The GAN structure with a generator network and a discriminator is used but the generator aims correctly impute values while the discriminator aims to classify imputed data and observed data.

We define the data vector, a random variable, as $\mathbf{x} = (x_1, \ldots, x_d)$. The mask vector $\mathbf{m} = (m_1, \ldots, m_d)$ is used to indicate which of the values of $\mathbf{x}$ that are observed and which that are missing. The mask vector $\mathbf{m}$ is taking values in $\{0, 1\}^d$ with a missing value is being represented as a zero and an observed value being represented by a one. The random variable $\tilde{\mathbf{x}} = (\tilde{x}_1, \ldots, \tilde{x}_d) \in \tilde{\mathcal{X}}$ is defined as

$$\tilde{x}_i = \begin{cases} x_i, & \text{if } m_i = 1 \\ *, & \text{otherwise} \end{cases} \tag{14}$$

.

The generator G has three inputs, realizations of $\tilde{\mathbf{x}}$, $\mathbf{m}$ and a d-dimensional noise, $\mathbf{z} = (z_1, \ldots, z_d)$. It outputs a vector of imputed values defined as

$$\bar{\mathbf{x}} = G(\tilde{\mathbf{x}}, \mathbf{m}, (\mathbf{1} - \mathbf{m}) \odot \mathbf{z}) \tag{15}$$

where the imputed values are generated by conditioning on the observed values. The completed data vector is obtained by combining the observations in $\tilde{x}_i$ with the imputed values of $\bar{\mathbf{x}}$ and is defined as

$$\hat{\mathbf{x}} = \mathbf{m} \odot \tilde{\mathbf{x}} + (\mathbf{1} - \mathbf{m}) \odot \bar{\mathbf{x}} \tag{16}$$

Rather than identifying an entire vector as real or generated, as in the ordinary GAN framework, the discriminator D in GAIN aims to identify specific components of the vector that are observed or imputed. The aim of the discriminator is accordingly to predict the components of the mask vector $\mathbf{m}$. The input to the discriminator D is the imputed values seen in equation (16) as well as a random hint vector $\mathbf{h}$. The hint mechanism is introduced to help D with finding $\mathbf{m}$ since the authors prove that there could be multiple distributions that D would consider optimal. This suggests that the hint vector additionally guarantees that the generator improves its capability to fill in missing data according to the true distribution of the observed data. In order to define $\mathbf{h}$, we first introduce the random variable $\mathbf{b} = (b_1, .., b_d) \in \{0, 1\}^d$. We sample k samples uniformly at random from $\{1,...d\}$ and then set

$$b_j = \begin{cases} 1 \text{ if } j \neq k \\ 0 \text{ if } j = k \end{cases} \tag{17}$$

The hint vector $\mathbf{h}$ is then defined as, given $\mathbf{m}$,

$$\mathbf{h} = \mathbf{b} \odot \mathbf{m} + 0.5(\mathbf{1} - \mathbf{b}). \tag{18}$$

.

By defining $\mathbf{h}$ in different ways it is possible to adapt the knowledge which the discriminator receives about $\mathbf{m}$. The discriminator will output $D(\hat{\mathbf{x}}, \mathbf{h})$, where the i-th element is the probability that that the i-th component of the completed data vector, $\hat{\mathbf{x}}$, was observed and not imputed conditional on and $\hat{\mathbf{X}} = \hat{\mathbf{x}}$ and $\mathbf{H} = \mathbf{h}$. We refer to the likelihood of forecasting the mask $\mathbf{m}$ as $\hat{\mathbf{m}}$.

2.3.2.1 Objective and Loss Functions

The loss function of the discriminator D is defined in (19) loss and the network is trained to minimize its negative value as seen in (20).

$$\begin{aligned} \mathcal{L}_D(\mathbf{m}, \hat{\mathbf{m}}, \mathbf{b}) = \sum_{i:b_i=0} & [m_i \log(\hat{m}_i) \\ & + (1 - m_i) \log(1 - \hat{m}_i)] \end{aligned} \tag{19}$$

$$\min_D - \sum_{j=1}^{k_D} \mathcal{L}_D(\mathbf{m}(j), \hat{\mathbf{m}}(j), \mathbf{b}(j)) \tag{20}$$

The loss of the generator G is consists of two loss functions. The loss function applied to the missing values seen in (21) and the reconstruction loss function applied to the observed values seen in (22).

$$\mathcal{L}_G(\mathbf{m}, \hat{\mathbf{m}}, \mathbf{b}) = - \sum_{i:b_i=0} (1 - m_i) \log(\hat{m}_i) \tag{21}$$

$$\mathcal{L}_M\left(\mathbf{x}, \mathbf{x}'\right) = \sum_{i=1}^{d} m_i L_M\left(x_i, x_i'\right) \tag{22}$$

where

$$L_M\left(x_i, x_i'\right) = \begin{cases} \left(x_i' - x_i\right)^2, & \text{if } x_i \text{ is continuous} \\ -x_i \log\left(x_i'\right), & \text{if } x_i \text{ is binary} \end{cases} \tag{23}$$

.

The generator will be trained according to minimise the sum of the two losses, with α as a hyperparameter to weigh the reconstruction loss, according to

$$\min_G \sum_{j=1}^{k_G} \mathcal{L}_G(\mathbf{m}(j), \hat{\mathbf{m}}(j), \mathbf{b}(j)) + \alpha \mathcal{L}_M(\tilde{\mathbf{x}}(j), \hat{\mathbf{x}}(j)) \tag{24}$$

.

The aim of the discriminator D is maximize the probability of finding the mask vector $\mathbf{m}$ while the aim of the generator G is to minimize the probability of D accurately predicting $\mathbf{m}$. The minimax game leads to the objective function

$$\begin{aligned} \min_G \max_D \mathbb{E}_{\hat{\mathbf{x}},\mathbf{m},\mathbf{h}} \big[& \mathbf{m}^T \log D(\hat{\mathbf{x}}, \mathbf{h}) \\ & + (\mathbf{1} - \mathbf{m})^T \log(\mathbf{1} - D(\hat{\mathbf{x}}, \mathbf{h})) \big] \end{aligned} \tag{25}$$

The optimization problem is solved in a iterative manner. Using mini-batches of size n_d, the discriminator D is optimized using a fixed generator G. The generator is then optimized using mini-batches of size n_g with a fixed discriminator.

2.4 Generating Data

The aim of this section is to provide the theory behind the GAN-based data synthesis method CTGAN. In the first subsection, the model WGAN is presented, an extension of GAN from which CTGAN is developed. The second subsection will provide a deep dive of the theory behind CTGAN.

2.4.1 WGAN

In order to understand the distinctions between CTGAN in comparison to the conventional GAN model, it is essential to provide an explication of the WGAN model.

2.4.1.1 WGAN Model and Algorithm Arjovsky *et al.* introduced the Wasserstein GAN (WGAN) in 2017 [3] with the primary aim to address the vanish gradient problem. In the vanilla GAN, the *Jensen-Shannon* (JS) divergence is used to measure the distance between two distributions P_r and P_g. In WGAN, the distance is instead expressed as the *Earth-Mover* (EM) distance or Wasserstein-1, defined as

$$W\left(\mathbb{P}_r, \mathbb{P}_g\right) = \inf_{\gamma \in \Pi\left(\mathbb{P}_r, \mathbb{P}_g\right)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|], \tag{26}$$

where $\Pi\left(\mathbb{P}_r, \mathbb{P}_g\right)$ denotes all the set of all joint distributions $\gamma(x, y)$ with marginals $\mathbb{P}_r$ and $\mathbb{P}_g$. The EM distance could be interpreted as the cost of transforming a distribution $\mathbb{P}_r$ into a distribution $\mathbb{P}_g$ and $\gamma(x, y)$ as a specific transportation plan.

The aim for the generator G in WGAN is to minimize the Wasserstein divergence seen in (26). The infinimum, the greatest lower bound, is unfortunately highly intractable, since there might be an infinite number of transportation plans. The authors are therefore rewriting (26) using the Kantorovich-Rubinstein duality [8]. The Wasserstein divergence between p_d and p_g that the generator G aims to minimize can therefore be expressed as

$$W\left(p_d, p_g\right) = \frac{1}{K} \sup_{\|f\|_L \leq K} \mathbb{E}_{x_d \sim p_d}\left[f\left(x_d\right)\right] - \mathbb{E}_{\tilde{x}_d \sim p_{gen}}\left[f\left(\tilde{x}_d\right)\right] \tag{27}$$

where the supremum is taken over all K-Lipschitz function $f : \mathcal{X} \to \mathbb{R}$. A K-Lipschitz function is defined as

$$\left|D_w\left(x_1\right) - D_w\left(x_2\right)\right| \leq K\left|x_1 - x_2\right| \quad \forall\left(x_1, x_2\right) \tag{28}$$

for a positive real constant K. In our context, this means the norm of the gradient of the discriminator network must be at most K.

The authors proposing WGAN are naming the discriminator D the term 'critic' instead of the term 'discriminator' used in the original GAN paper since it is providing feedback rather than fully discriminating. However, in this paper, we will follow the naming of the discriminator network D to remain consistent. Replacing f with D in equation (27) updates the minimax game between D and G to

$$\min_G \max_{\theta_d \in \mathcal{W}} \mathbb{E}_{x_d \sim p_d}[D(x; \theta_d)] - \mathbb{E}_{\mathbf{z} \sim p_\mathbf{z}}\left[D\left(G\left(z; \theta_g\right); \theta_d\right)\right] \tag{29}$$

where all functions $\{D_{\theta_d}\}_{\theta_d \in \mathcal{W}}$ are all K-Lipschitz for some K. Note that to ensure the K-Lipschitz constraint, the weights θ_d will have to lie in a compact space $\mathcal{W}$ after each gradient update, something that is solved by introducing weight clamping to a fixed range $[-c, c]$. Selecting a large value of c could result in an extensive training duration, whereas selecting a small value could induce the occurrence of vanishing gradients.

2.4.1.2 WGAN-GP The weight clamping solution in WGAN was seen to show undesired behaviour, and Gulrajani *et al.* introducted gradient penalty (WGAN-GP) [23] to combat this issue. To ensure the Lipschitz contraint, an alternative approach was to add a gradient penalty to the loss function for random samples $\mathbf{z} \sim p_\mathbf{z}$. The updated objective function for WGAN-GP can be described as

$$\begin{aligned}\max_{\theta_d \in \mathcal{W}} \mathbb{E}_{x_d \sim p_d}[D(x;\theta_d)] - \mathbb{E}_{\mathbf{z} \sim p_\mathbf{z}}\left[D\left(G\left(z;\theta_g\right);\theta_d\right)\right] \\ +\lambda \mathbb{E}_{\mathbf{z} \sim p_\mathbf{z}}\left[\left(\left\|\nabla_{\theta_d} D\left(G\left(z;\theta_g\right);\theta_d\right)\right\|_2 - 1\right)^2\right]\end{aligned} \tag{30}$$

where the penalty coefficient is λ set to 10 in the original paper.

2.4.2 CTGAN

The Conditional Tabular GAN (CTGAN) is a variation of a GAN first introduced by Lei Xu and Kalyan Veeramachaneni in 2019, with the aim to successfully model and synthesize tabular data [72].

2.4.2.1 Data Pre-processing

Categorical values CTGAN represents categorical values as a one-hot-vector as described in Section 2.1.2, where each vector comprises zeros and a single one, located at the position denoting the mode. The one-hot vector $\mathbf{c}_{i,j}$ for a sample $c_{i,j}$ has $|C_i|$-dimensions, the number of possible categories for the categorical column C_i.

Numerical values To deal with the complex distribution and multi-modality of numerical values, CTGAN uses *mode-specific* normalization. The aim is to express a given value by a one-hot vector β representing the mode and a scalar normalized value α. The method consists of three steps.

1. A variational Gaussian mixture model (VGM) is used to find the number of modes, m_i, for each numerical column N_i. The resulting Gaussian mixture can be expressed as $\mathbb{P}_{N_i}(n_{i,j}) = \sum_{q=1}^{N} \omega_q \cdot \mathcal{N}\left(n_{i,j}; \mu_q, \sigma_q\right)$ for a value $c_{i,j}$ where ω_q , μ_q and σ_q are respectively the weight, the mean and the standard deviation of a specific mode q.

2. The probability density for each value $n_{i,j}$ coming from each mode q is computed as $\rho_q = \omega_q \cdot \mathcal{N}\left(n_{i,j}; \mu_q, \sigma_q\right)$ for a value $n_{i,j}$.

3. From the computed probability density, one mode q^* is sampled which is used to normalize the value. Each $n_{i,j}$ is represented as a one-hot vector $\beta_{i,j}$, filled with zeros except for a one at the position q^*, and a scalar $\alpha_{i,j}$ computed as $\alpha_{i,j} = \frac{n_{i,j} - \mu_{q^*}}{4\sigma_{q^*}}$.

Numerical and categorical columns are concatenated to reach the representation of a row j:

$$\mathbf{r}_j = \alpha_{1,j} \oplus \beta_{1,j} \oplus \ldots \oplus \alpha_{N_n,j} \oplus \beta_{N_n,j} \oplus \mathbf{d}_{1,j} \oplus \ldots \oplus \mathbf{d}_{N_c,j} \tag{31}$$

,

where N_n and N_c represent the total number of numerical and categorical columns respectively.

2.4.2.2 Solutions to Deal with Class Imbalance As described in Section 2.2.5, the original GAN framework, the training does not account for imbalanced categorical columns. This is causing minor categories to not exposed sufficiently during training since the generator becomes biased towards the major categories. The authors proposing CTGAN are suggesting two solutions to combat this issue, a *conditional generator* and *training-by-sample*.

Conditional vector The aim of introducing a condition into the framework is to sample $\mathbf{x}_d$ from $p_d(\mathbf{x}_d \mid \mathbf{c})$ and not from the unconditioned $p_d(\mathbf{x}_d)$. The vector $\mathbf{c}$, a conditional vector, is introduced in both the generator and the discriminator. We can ensure equal representation of all categories in the training data by sampling a batch of conditional vectors $\mathbf{c}$. A mask vector is introduced to create $\mathbf{c}$. Every one-hot vector $\mathbf{d}_i$ is associated with a mask vector $\mathbf{m}_i$ defined as

$$\mathbf{m}_i^{(k)} = \begin{cases} 1 & \text{if } i = i^* \text{ and } k = k^* \\ 0 & \text{otherwise.} \end{cases} \tag{32}$$

The conditional vector $\mathbf{c}$ is a concatenation of the total number N_c of mask vectors

$$\mathbf{c} = \mathbf{m_1} \oplus ... \oplus \mathbf{m}_{N_c} \tag{33}$$

To put it in practice, consider a simple example of two categorical columns $C_1 = \{A, B, C\}$ and $C_2 = \{D, E\}$. If we desire to generate a sample with the value C, the first value of the second column, $\mathbf{c}$ the mask vectors would be expressed as $\mathbf{m_1} = \{0, 0, 0\}$ and $\mathbf{m_2} = \{1, 0\}$. The conditional vector $\mathbf{m_2}$ would be computed as $\mathbf{c} = \mathbf{m_1} \oplus \mathbf{m_2} = \{0, 0, 0, 1, 0\}$. To condition on the vector in the framework, the initial input of the generator includes both the noise variable z and the conditional vector c.

Training-by-sample Training-by-sample is introduced in CTGAN to sample the conditions leading to the generation of a proper conditional vector $\mathbf{c}$. The training-by-sample technique for each batch of observations can be described in six steps.

1. Create N_d zero-filled mask vectors m_i.
2. Select, with equal probability, a categorical column C_i. The index of the selected column is represented as i^*.
3. Compose a PMF (probability mass function) for the values in C_i, such that the probability mass of each category is the logarithm of its frequency in D_i.
4. According to the PMF, select a category k^*.
5. Set the k^*th component of the mask vector of C_i to one.
6. Compute $\mathbf{c}$ according to $\mathbf{c} = \mathbf{m_1} \oplus ... \oplus \mathbf{m}_{N_c}$

2.4.2.3 Network architecture Both the generator and discriminator consists of two fully-connected hidden layers, each with size 256. The loss function from the WGAN-GP framework as seen in (30) is used to train the networks.

Generator structure The conditional vector c is fed into the generator in addition to the noise variable z as the initial input. The generator uses batch-normalization and `ReLU` as the activation function. The activation functions that are used in the output layer, are `tanh` for the scalar values α_i and `Gumbel softmax` for β_i and $\mathbf{d}_i$. The full network structure for the conditional generator can be described as

$$h_0 = z \oplus c \tag{34}$$

$$h_1 = h_0 \oplus \texttt{ReLU}(\texttt{BN}(\texttt{FC}_{|\mathbf{c}|+|\mathbf{z}|\to 256}(h_0)) \tag{35}$$

$$h_2 = h_1 \oplus \texttt{ReLU}(\texttt{BN}(\texttt{FC}_{|\mathbf{c}|+|\mathbf{z}|+256\to 256}(h_1)) \tag{36}$$

$$\hat{\beta}_i = \texttt{gumbel}_{0.2}(\texttt{FC}_{|\mathbf{c}|+|\mathbf{z}|+512\to |m_i|}(h_2)), \quad i = 1, 2...N_c \tag{37}$$

$$\hat{\mathbf{d}}_i = \texttt{gumbel}_{0.2}(\texttt{FC}_{|\mathbf{c}|+|\mathbf{z}|+512\to D_i}(h_2)), \quad i = 1, 2...N_d \tag{38}$$

$$\hat{\alpha}_i = \texttt{tanh}(\texttt{FC}_{|\mathbf{c}|+|\mathbf{z}|+512\to 1}(h_2)), \quad i = 1, 2...N_c \tag{39}$$

To enforce the generator to produce samples that match the condition given by the conditional vector an additional loss term, the cross-entropy between $\mathbf{m}_{i^*}$ and $\hat{\mathbf{d}}_{i^*}$, is added to the generator loss. For a batch of n samples, the additional loss is expressed as

$$\mathcal{L}_{\mathbf{c}} = \frac{1}{n} \sum_{i=1}^{n} \left(CE\left(\mathbf{m}_{i^*}^{(i)}, \hat{\mathbf{d}}_{i^*}^{(i)} \right) \right) \tag{40}$$

Discriminator structure The discriminator uses `Dropout` and `LeakyReLU` as activation functions instead of `ReLU` used in the generator. To prevent the issue of mode collapse, the discriminator uses the PacGAN framework [38]. The input h_0 to the first layer is a concatenation of 10 samples and the corresponding conditional vectors. This permits the discriminator to examine multiple samples concurrently. The full network structure for the discriminator can be described as

$$h_0 = \mathbf{r}_1 \oplus ...\mathbf{r}_{10} \oplus \mathbf{c}_1 \oplus \mathbf{c}_{10} \tag{41}$$

$$h_1 = \texttt{drop}(\texttt{leaky}_{0.2}(\texttt{FC}_{10|\mathbf{r}|+10|\mathbf{c}|\to 256}(h_0)) \tag{42}$$

$$D = \texttt{drop}(\texttt{leaky}_{0.2}(\texttt{FC}_{256\to 256}(h_1)) \tag{43}$$

$$\hat{\beta}_i = \texttt{FC}_{256\to 1}(h_2)) \tag{44}$$

$$\tag{45}$$

The loss function of the discriminator is the loss function used in a conditional WGAN-GP. CTGAN also implements the Adam optimizer with learning rate $2 \cdot 10^{-4}$.

3 Methodology

This chapter presents a detailed description of the experimental methodology undertaken to explore the research question. The data used in the study is described, as well as the data pre-processing methods and overall experimental approach. Finally, the evaluation framework is presented.

3.1 Data

To evaluate the different imputation techniques, five real world data sets from different domains commonly used within machine learning were selected. Collected from UCI Machine Learning Repository [13] were *Default of Credit Card Clients*[2], *Bank Marketing*[3], *Online News Popularity*[4], and *Letter Recognition*[5], while *Mushroom Classification*[6] was retrieved from Kaggle. To simplify the namings onwards, we will refer to the data sets as *Mushroom*, *Letter*, *Bank*, *Credit*, and *News*. The data sets were selected due to their differences in number of observations as well as in number of numerical and categorical columns, in order to thoroughly evaluate the imputation methods and their performance in different contexts. Worth noting is that all numerical columns are treated as continuous in the study, while all categorical are considered to be discrete and unordered. Further, the original data sets contains no missing values. A brief description of the data sets is presented below and a summary can be found in Table 2.

- **Mushroom**: The *Mushroom* data set contains samples from 23 species of gilled mushrooms. The goal is correct classify weather the mushroom is edible or poisonous. The data set only contains categorical features.
- **Letter**: The *Letter* data set aims to classify one of the 26 capital letters in the alphabet based on 16 primitive numerical attributes.
- **Bank**: The *Bank* data set contains data from a direct marketing campaign where the aim is to classify weather customer will subscribe a term deposit. The full additional data set was used with all 41188 samples and 20 inputs.
- **Credit**: The *Credit* data set contains 23 explanatory variables that aims to predict if an individual will default or not.
- **News**: The *News* data set consists of features of articles published under a two year period. The aim is classification of the popularity of the articles, measured by number of shares of the article in social networks. The data set mostly contains numerical features but 3 categorical features are found, originally one-hot-encoded.

[2]https://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients
[3]https://archive.ics.uci.edu/ml/datasets/Bank+Marketing
[4]https://archive.ics.uci.edu/ml/datasets/online+news+popularity
[5]https://archive.ics.uci.edu/ml/datasets/letter+recognition
[6]https://www.kaggle.com/datasets/uciml/mushroom-classification

Data set	#Train	#Test	#N	#C	Task	Target	Target class
Mushroom	6499	1625	0	23	Classification	Class	Binary
Letter	16000	4000	16	0	Classification	Letter	Multi-class
Bank	32950	8238	10	10	Classification	Term deposit	Binary
Credit	24000	6000	14	9	Classification	Default	Binary
News	31715	7929	44	3	Regression	# of shares	Continuous

Table 2: Summary of data sets used in the study. The notations #N and #C represent the number of numerical and categorical columns.

3.1.1 Data Pre-processing

Data removal The first step in the data pre-processing was to remove identifier variables and non-predictive variables. This included removing the feature *ID* from the *Credit* data set and removing the features *url* and *timedelta* from the *News* data set.

Encoding of categorical features For Median/Mode imputation and MICE, label encoding as described in Section 2.1.2, was performed to represent categorical features, thus representing each categorical alternative in one column as a number from 1 to the N number of possible values in the particular column. For kNN imputation, MissForest and GAIN imputation, the technique of one-hot encoding was used for the categorical features, also as described in Section 2.1.2. If a feature value was missing in the data set before performing one-hot encoding for one particular observation, then all subcategories associated with that feature were considered missing for this observation in the resulting one-hot encoded representation.

Normalization The numerical features in all data sets was normalized to the interval [0, 1]. Before imputation the training data set with missing values was normalized, and the same normalization parameters were used for the test data set. The motivation behind this approach was to mimic real-world scenarios where complete data sets without missing values are not available during imputation. Thus, normalization must be performed using the data sets with missing values. For the training data sets containing additional training data generated by CTGAN, the normalisation parameters based on the training data set with missing values including additional CTGAN generated data were used.

To perform evaluation of the imputation performance, the normalization parameters based on the complete training data set with no missing values were used. These normalisation parameters were applied on the complete test data and the imputed test data for evaluation. These normalization parameters were also used for the cases when the model was trained with additional CTGAN data. The motivation behind this is that the imputation performance is based on a comparison between the imputed and the original complete data sets, and the normalization should accordingly also be based on the original data sets.

The above-mentioned approach for normalization was applied to the numerical features for all data sets and all imputation methods.

3.2 Experimental methodology

3.2.1 Overall experimental setup

Training and testing division and introduction of missing values Data was split into training and testing with ratio 80% train data and 20% test data. For all five data sets, missing values were introduced according to the MCAR mechanism. The miss rates used were 10%, 30%, and 50%. For higher miss rates than 50%, data imputation methods might not be a feasible solution in practical applications. Further, few studies have been conducted using higher miss rate than 50% [46].

Synthetic data generation To evaluate the performance of research question 2, i.e., the data imputation methods with an increased amount of training data, which going forward will be called the augmented training data, the original training data set with missing values was split into two data sets: Data Set A), consisting of observations with missing values and Data Set B), formed of observations with only observed values and no missing values. Data Set B) with only observed values was subsequently used to train a CTGAN model. The CTGAN model was used to generate Data Set C), i.e., new additional synthetic data. All three data sets, i.e., A), B) and C) were concatenated to create a training data set with an increased number of rows. The study was conducted in two different ways, by creating Data Set C) both 50% the size and 100% the size of the total training data size, i.e., of the joint amount of Data Set A) and Data Set B). It should be noted that for certain data sets with significant levels of missing data, it was not possible to perform the methodology due to the absence of complete rows. These include the News data set with 30% and 50% missing values, as well as all other data sets with 50% missing values.

Study setup With the augmented training data sets and the non-augmented training data set, data imputation methods were used to impute the missing values. The methods used in the study were Median/Mode Imputation, kNN imputation, MICE, MissForest, and GAIN. An alternative to imputation, list-wise deletion of incomplete rows, were also done and evaluated as further described in Section 3.4. The full methodology can be seen in Figure 2.

Figure 2: Overview of experimental methodology of study.

3.3 Model Implementation, Adjustments and Tuning

3.3.1 Model Implementation in Practice

List-wise deletion In the case of list-wise deletion, all incomplete rows in the test data set were removed and the remaining rows were then used to predict the target. For smaller data sets and higher miss rates, this meant no complete rows were remaining and thus the methodology could not be pursued.

Median and Mode Imputation The Median and Mode Imputation models were implemented using the SimpleImputer[7] library from Scikit-Learn. For continuous features, Median Imputation was chosen over Mean Imputation, as the method is less sensitive to outliers [35].

[7]https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html

MICE MICE was implemented using the R package mice: Multivariate Imputation by Chained Equations [8], developed by Van Buuren and Groothuis-Oudshoorn, the authors of the original paper in which the model was first presented in 2011 [65]. The model was implemented using the default settings for imputation methods, which implies using predictive mean matching for numeric features, logistic regression imputation for binary features, polytomous regression imputation for features with unordered categorical data with more than two features, and lastly proportional odds model for ordered categorical data with more than two features. The model was only run once due to the long execution time compared to other models. Thus, a seed was used to ensure reproducability of the study.

kNN Imputation The kNN imputation method was implemented using the kNNImputer[9] package from SciKit Learn in Python. The Euclidean distance metric was used to determine the distance between observations and the neighbor's values were weighted uniformly to impute the missing values. The model was only run once since it generates in the same imputed values on every iteration.

missForest missForest was implemented using the Python library missingpy developed by Ashim Bhattarai, a Python implementation of the original missForest model first introduced by Bühlmann and Stekhoven [59]. The model is based on the RandomForestRegressor[10] and RandomForestClassifier[11]. The model was run 10 times and the average results as well as the standard deviations were reported.

GAIN The GAIN method was obtained from its Github repository[12]. Two versions of the GAIN method were employed in the study, one model following the original code without any modifications which in this thesis will be referred to as GAIN v1, and one model were several suggestions were applied to the original code as described in Section 3.3.2. The latter model will be referred to as GAIN v2. Both models were run 10 times and the average results as well as the standard deviations were reported.

CTGAN CTGAN was implemented using Synthetic Data Vault (SDV)'s model CTGAN[13] based on the original paper. The suggested model parameters, found in the paper where CTGAN was introduced [72], were used.

Hardware versions In order to ensure that the execution time was calculated in a correct and comparable way, all imputations and evaluations have been run locally on the same machine, a Macbook Pro (2021) with 32 GB memory, Apple M1 Pro chip, and 1 TB flash storage.

[8] https://cran.r-project.org/web/packages/mice/mice.pdf
[9] https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html
[10] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
[11] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[12] https://github.com/jsyoon0823/GAIN
[13] https://sdv.dev/SDV/user_guides/single_table/ctgan.html

3.3.2 Model Adjustments

kNN Imputation modifications When imputing categorical variables in a one-hot encoded data set, it is important that only one category per original feature is imputed, i.e., that one of the categories for each observation takes the value 1 while the other categories for this observation is set to 0 [16].

To illustrate what this means in practice, $\boldsymbol{X}$ is assumed to be an $(n \times p)$-dimensional data matrix, where X_{is} is the value of the i^{th} observation's value of the the s^{th} feature. This leads to that $\mathbf{X}_i = (X_{i1}, \cdots, X_{ip})$ represents one observation. If s is a categorical feature, a one-hot encoded variable X_{is} is represented by a dummy vector $X_{is} = (X_{is1}, X_{is2}, ..., X_{isd_s})$, where d_s is the number of possible alternative that the categorical feature s can take. With this notation, the statement that only one category per original feature is imputed implies that only one value in the dummy vector X_{is} is allowed to be 1 for each observation i, representing the imputed category, while the other values should be set to 0.

When applying kNN imputation, the statement that only one category per original feature should be imputed is not always true. To ensure that this is the case, a similar strategy to the one implemented by Faisal and Tutz [16] is used in this thesis. Faisal and Tutz presents a variant of kNN imputation, which, with the notation introduced above and where K is the number of optimal neighbors, initially represents each categorical value c within a feature s as a probability:

$$\widehat{\pi}_{isc} = \sum_{j=1}^{\mathrm{K}} w\left(\mathbf{X}_i, \mathbf{X}_{(j)}\right) X_{(j)sc} \tag{46}$$

where $w\left(\mathbf{X}_i, \mathbf{X}_{(j)}\right)$ is a weighting function determining the importance of each neighbor j.

In this thesis, all neighbors are equally important, and the probability value of one category value c within a feature s is represented as the average of the neighbor's corresponding values $X_{(j)sc}$, setting $w\left(\mathbf{X}_i, \mathbf{X}_{(j)}\right) = 1/K$. To ensure that only one value in the dummy vector X_{is} is set to 1 for each observation i, the value X_{is} is determined based on the index with highest probability [16]:

$$X_{isc} = \begin{cases} 1 & \text{if } \widehat{\pi}_{isc} = \operatorname{argmax}_{q=1}^{d_s} \widehat{\pi}_{isq} \\ 0 & \text{otherwise} \end{cases} \tag{47}$$

GAIN modifications To adapt the original GAIN version to better deal with categorical variables, three modifications were made to the original code implementation from the original paper. The model based on the original code implementation will be referred to as GAIN v1 in this thesis, while the model that includes the modifications will be referred to as GAIN v2.

Firstly, with the aim of achieving the imputation of only one category per original feature in the one-hot encoded data set, as described above for kNN imputation, a new rounding function was implemented. This rounding function operates with the same goal as the extension for kNN described in Section 3.3.2. This results in that it ensures that the one-hot encoded vector of

observation i and feature s, $X_{is} = (X_{is1}, X_{is2}, ..., X_{isd_s})$, is determined based on the category with index c, $c = 1, ..., d_s$, for which the initially imputed value X_{isc} is closest to 1, which implies X_{isc} is set to 1. All other elements in the vector $X_{is} = (X_{is1}, X_{is2}, ..., X_{isd_s})$ is set to 0.

The second modification is that a different activation functions was used in the output layer in the generator for different variable types, instead of using the same activation function for all variable types. Due to the limitations of the `softmax` activation function, which traditionally has been used to output categorical or discrete values but cannot be differentiated and does not enable back-propagation, the `Gumbel softmax` was implemented for categorical variables. For numerical variables, the `sigmoid` activation function was used as in the original implementation. Additionally, this involved the inclusion of another hyperparameter, τ.

Lastly, the generator loss function (24) was updated to weigh the loss of categorical variables separately from the loss of numerical variables, similar to done in [12]. The binary cross-entropy loss is used to compute the categorical loss and the numerical loss remains computed as the mean square error, as seen in equation (49) and (48), respectively, where $\mathbf{x}_n$ represents a numerical variable and $\mathbf{x}_c$ represents a categorical variable, such that

$$\mathcal{L}_{M1}(\mathbf{x}_n, \mathbf{x}'_n) = \sum_{i=1}^{d_n} m_i \left(x'_{i,n} - x_{i,n}\right)^2 \tag{48}$$

$$\mathcal{L}_{M2}(\mathbf{x}_c, \mathbf{x}'_c) = -\sum_{i=1}^{d_c} m_i \left(x_{i,c} \log\left(x'_{i,c}\right) + (1 - x_{i,c}) \log\left(1 - x'_{i,c}\right)\right) \tag{49}$$

With an additional hyper parameter β, the updated generator loss function becomes

$$\min_G \sum_{j=1}^{k_G} \mathcal{L}_G(\mathbf{m}(j), \hat{\mathbf{m}}(j), \mathbf{b}(j)) + \alpha \mathcal{L}_{M1}(\tilde{\mathbf{x}}_n(j), \hat{\mathbf{x}}_n(j)) + \beta \mathcal{L}_{M2}(\tilde{\mathbf{x}}_c(j), \hat{\mathbf{x}}_c(j)) \tag{50}$$

In summary, the model GAIN v2 corresponds to the GAIN model presented in the original paper [74], GAIN v1, with these three modifications.

3.3.3 Model Hyperparameters and Tuning

To obtain the optimal results for the methods which requires determination of hyperparameters, a process of hyperparameter tuning was conducted using cross-fold validation. In this thesis, this is the case for kNN imputation and GAIN, and the selected parameters for each imputation method per data set can be found in Table 25. The performance score used for both kNN imputation and GAIN was prediction accuracy in terms of AUROC and MSE, further described in Section 3.4.2. Since the augmented data sets with additional data generated by CTGAN do not contain true labels, prediction accuracy could not be performed and thus the optimal hyper parameters for the corresponding data set with no extra data were used.

kNN Imputation kNN imputation only have one parameter to optimize, which is the optimal number of neighbors, K. A five-fold cross validation on the training data set was performed in

order to determine the optimal value of this parameter, measured by predictive performance. The value of K was restricted with a maximum value as the square root of the number of observations for each data set.

GAIN GAIN on the other hand have several parameters to optimize, and the value of one parameter is likely to affect the optimal value of another. Thus, the optimal parameter selection for every data set and miss rate was determined through cross-fold validation with three folds and 3000 iterations on the training data. The parameter search grid consisted of *batch size*:$\{64, 128, 256\}$, *hint rate*:$\{0.1, 0.5, 0.9\}$, and *alpha*:$\{0.5, 1, 2, 10\}$, found in studies of a similar nature [74] [47] [7] [12] [6]. The updated version of GAIN contains two additional hyperparameters. The optimal values for the hyperparameters, *batch size*, *hint rate*, and *alpha* were selected for the updated version through optimization using the original version of GAIN. The parameter search grid for the updated version then consisted of *beta*: $\{0.1, 0.5, 1, 10, 50\}$ and *tau*: $\{0.1, 0.5, 5, 10\}$.

3.4 Evaluation Framework

The evaluation of imputation performance is done based on two criteria, direct imputation performance and prediction performance.

3.4.1 Direct Imputation Performance

According to a systematic review conducted by Thomas and Rajabi [61], which examined 2883 research papers comparing missing value imputation techniques from 2010 to 2020, Root Mean Squared Error (RMSE) and Percentage of Correct or False Prediction (PCP or PFP) are widely used evaluation metrics for numerical and categorical values. Consequently, these metrics will be used in the current study as well.

Root Mean Square Error Numerical value imputation is evaluated using the Root Mean Square Error (RMSE) between observed and imputed values [37].

$$\text{RMSE}_{num} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2} \tag{51}$$

where x_i is the observed value, $\hat{x}_i$ is the imputed value and n are the total number of missing values.

Proportion of Falsely Classified Categorical value imputation is evaluated using the the Proportion of Falsely Classified (PFC) entries. This is the complementary to the Proportion Correctly Classified (PCC) [37].

$$PFC = \left(\frac{\sum_{i=1}^{n}\text{count}\left(y_i \neq \hat{y}_i\right)}{\sum_{i=1}^{n}\text{count}(y_i)}\right) \times 100 \tag{52}$$

Modified Root Mean Square Error To enable a concatenated evaluation of the categorical values and the numerical values, a measurement refered to as modified RMSE (mRMSE) is used, found in [22]. The categorical RMSE is calculated as

$$\text{RMSE}_{cat} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} 1_{\{\widehat{x}_i \neq x_i\}}} \tag{53}$$

To form the modified RMSE, (51) and (51) are concatenated according to

$$\text{mRMSE} = \sqrt{(\text{RMSE}_{num})^2 + (\text{RMSE}_{cat})^2} \tag{54}$$

Execution time To determine which imputation methods that offer greater practical benefits, the execution time is measured for every imputation method.

3.4.2 Prediction Performance

Prediction models The prediction models used in the study were LinearRegression[14] for data sets with regression task and the KNeighboursClassifier[15] for data sets with classification task. The task for every data set can be seen in Table 2. For the KNeighboursClassifier, StandardScaler[16] was used to standardize features and cross-validation was used to determine the optimal value for the number of neighbors. For both models, data was split into 80% training and 20% testing.

Prediction evaluation for classification task To measure the prediction performance for the imputed values for classifiers the accuracy score[17] and the Area under the Receiver Operating Characteristic Curve[18] (AUROC) were used. These metrics are two of the most common evaluations metrics according to [46]. To define the metrics, the notations found in Table 3 are applied.

		Predicted condition	
	Total population = P + N	Predicted Positive (PP)	Predicted Negative (PN)
Actual condition	Actual Positive (P)	True Positive (TP)	False Negative (FN)
	Actual Negative (N)	False Positive (FP)	True Negative (TN)

Table 3: Confusion matrix to define TP, TN, FN and FP.

Accuracy is defined as the ratio of correct predictions to the total number of predictions.

[14] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
[15] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
[16] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
[17] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
[18] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html

$$Accuracy = \frac{TP + TN}{P + N} \tag{55}$$

The Receiver Operating Characteristic (ROC) curve is plotted with True Positive Rate (TRP) on the vertical axis and False Positive Rate (FPR), defined as

$$TPR = \frac{TP}{TP + FN} \tag{56}$$

$$FPR = \frac{FP}{FP + TN} \tag{57}$$

The Area Under the Receiver Operating Characteristic Curve (AUROC) is used as a summary of the ROC curve. The values ranges from 0 to 1 where a higher value indicates a better predictor. For data sets with multiclass target columns, the one-vs-rest approach is used, computing the AUROC of each class against the rest and then finding the unweighted average, since it is considered necessary to prioritize the performance of all classes equally.

Prediction evaluation for regression task To measure the prediction performance for the imputed values for regression tasks, we use the Mean Square Error[19] (MSE) defined as

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{58}$$

[19] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html

4 Results

This section presents the most important results from the study. The full results are found in the appendix.

4.1 Comparison of GAIN to Standard Imputation Methods

This section will present the results of 'Research Question 1: How does GAIN compare to standard imputation methods when being evaluated in an extensive way?'.

4.1.1 Imputation Results

The imputation performance is determined based on the evaluation metrics presented in Section 3.4.1. The full results are found in the appendix.

Best performing method Figure 3 shows the best performing method statistically significant at 95% confidence level amongst the different imputation methods for different direct imputation evaluation metrics.

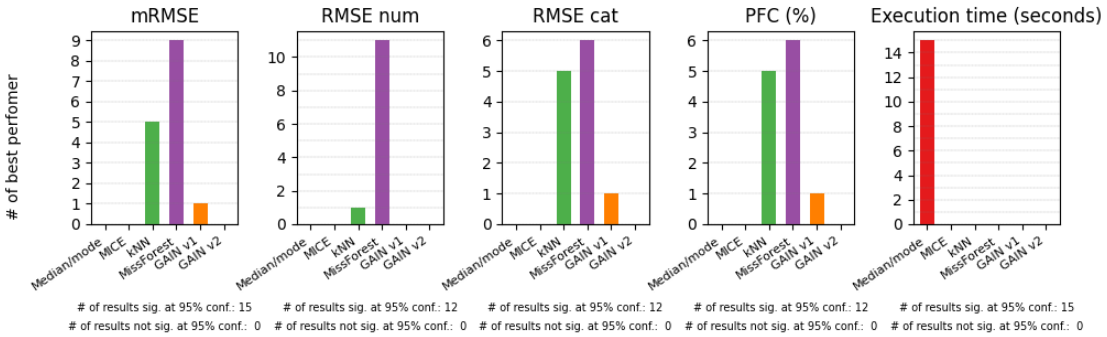

Figure 3: Number of best performer statistically significant at 95% confidence level amongst the different imputation methods for different direct imputation evaluation metrics.

mRMSE per data set and miss rate Figures 4 to 8 present the performance in terms of mRMSE of different methods per data set and miss rate.

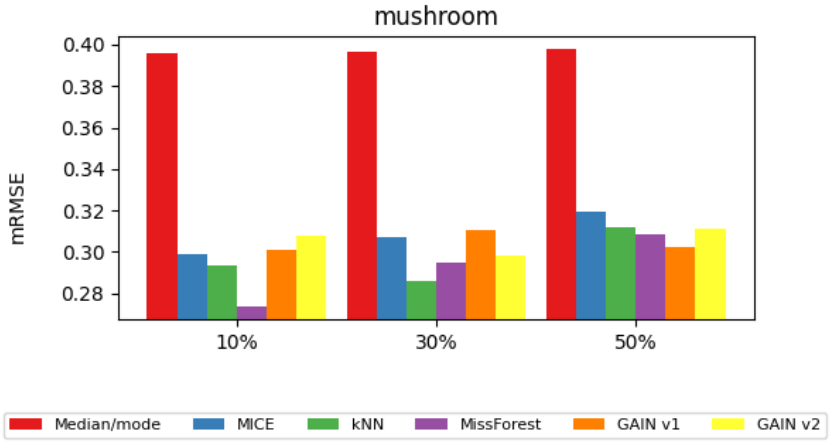

Figure 4: Evaluation in terms of mRMSE for the different imputation methods for different miss rates for the mushroom data set.

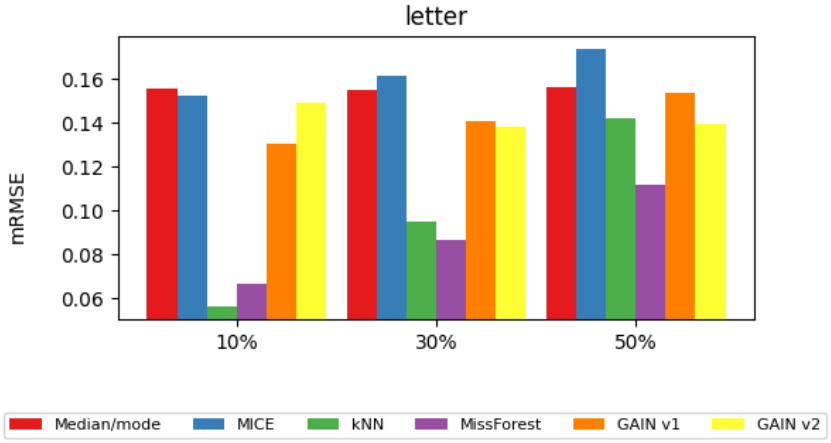

Figure 5: Evaluation in terms of mRMSE for the different imputation methods for different miss rates for the letter data set.

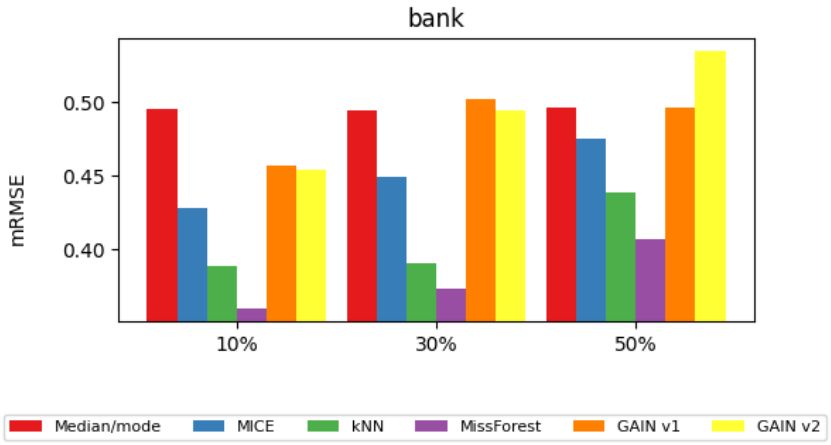

Figure 6: Evaluation in terms of mRMSE for the different imputation methods for different miss rates for the bank data set.

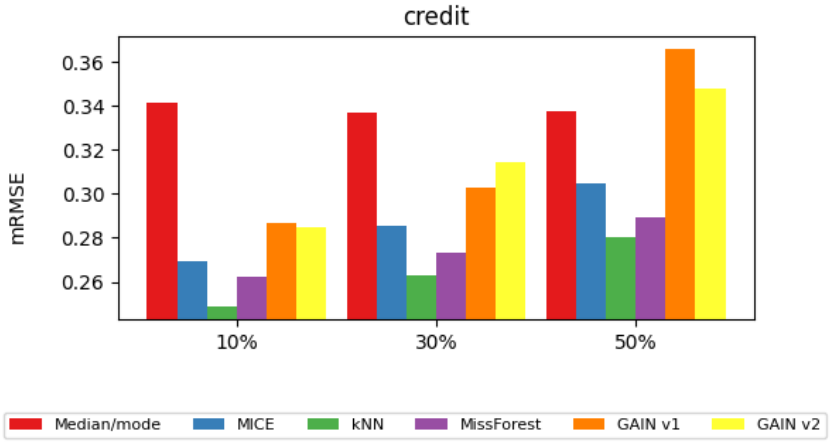

Figure 7: Evaluation in terms of mRMSE for the different imputation methods for different miss rates for the credit data set.

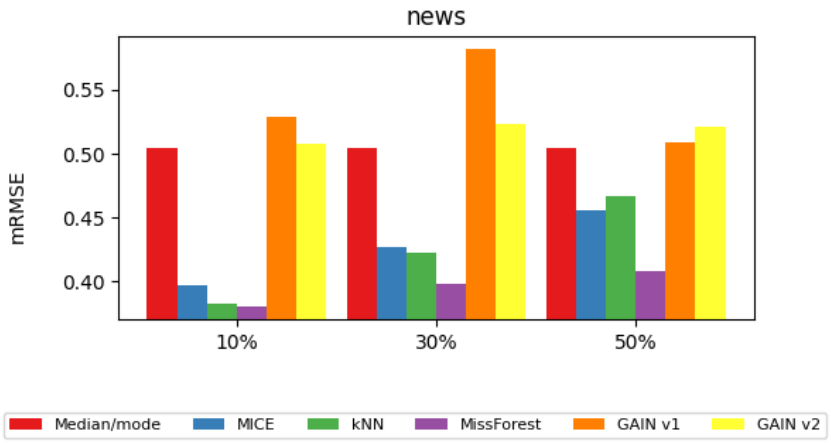

Figure 8: Evaluation in terms of mRMSE for the different imputation methods for different miss rates for the news data set.

Comparison of GAIN v1 to GAIN v2 Figure 9 presents the best performing method of the two GAIN versions statistically significant at a 95% confidence level for different imputation evaluation metrics. Figure 19 presents the best performing method of the two GAIN versions for different imputation evaluation metrics, not taking the confidence interval into consideration.

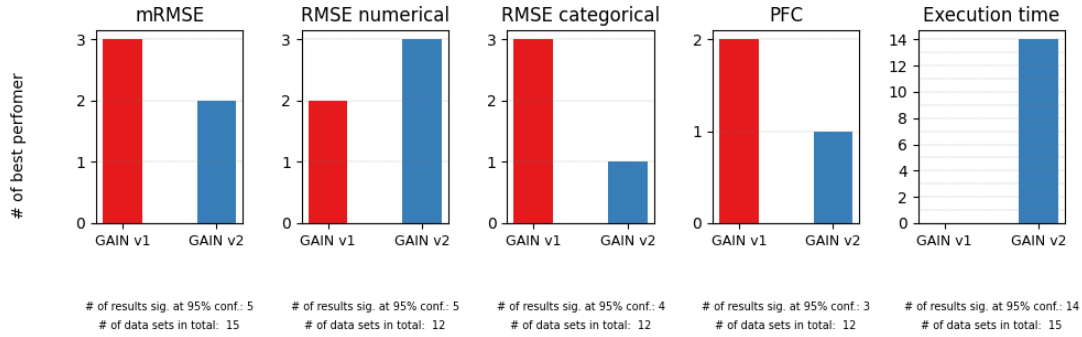

Figure 9: Number of best performer amongst the different GAIN versions statistically significant at a 95% confidence level for different direct imputation evaluation metrics.

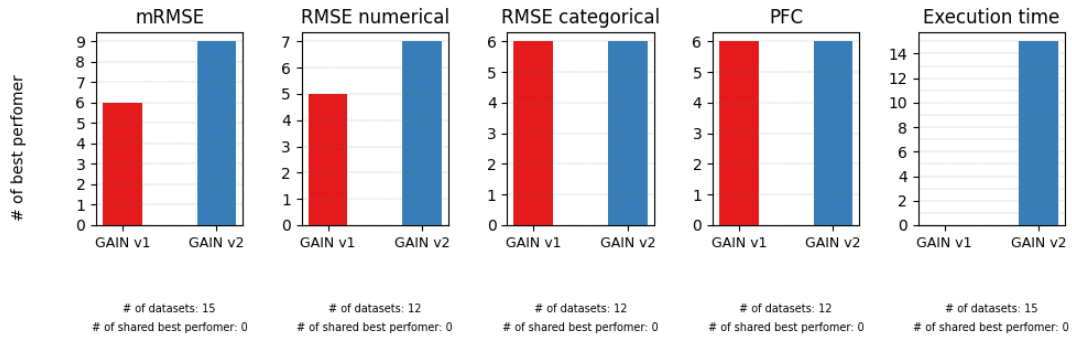

Figure 10: Number of best performer amongst the different GAIN versions for different direct imputation evaluation metrics.

4.1.2 Prediction Result

The prediction performance will be determined in terms of AUROC and accuracy for all data sets for which the main goal is classification, which is the case for all data sets except for News, and MSE for the data sets which aims to use regression to predict the target variable. The full results can be found in the appendix.

Best performing method Figure 11 shows the best performing method statistically significant at 95% confidence level amongst the different imputation methods for different direct imputation evaluation metrics. Figure 12 shows the best performing method amongst the different imputation methods for different direct imputation evaluation metrics when considering the average values, not taking the confidence intervals into consideration.

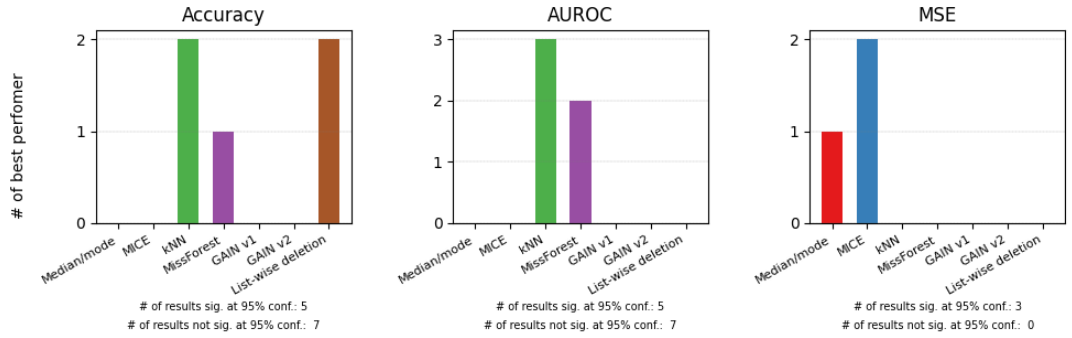

Figure 11: Number of best performer statistically significant at 95% confidence level amongst the different imputation methods for different prediction evaluation metrics.

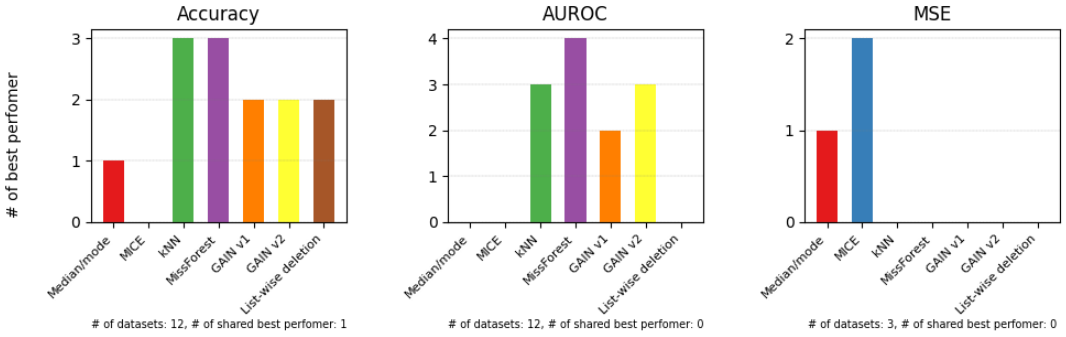

Figure 12: Number of best performer in terms of average value amongst the different imputation methods for different direct prediction evaluation metrics.

AUROC per data set and miss rate Figures 13 to 17 present the performance in terms of AUROC of different methods per data set and miss rate.

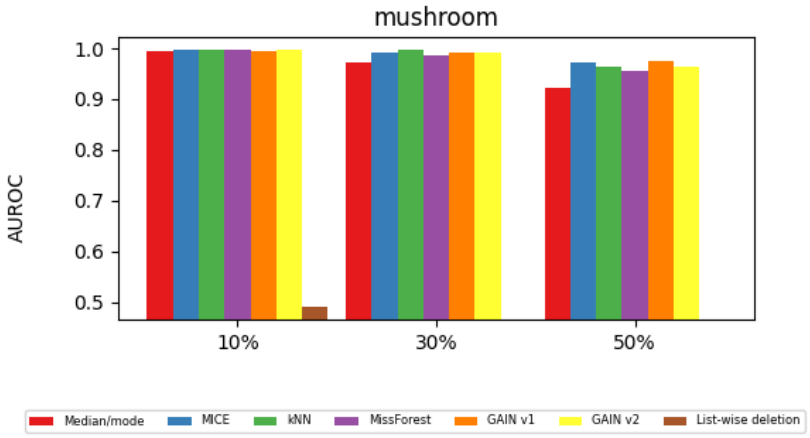

Figure 13: Evaluation in terms of AUROC for the different imputation methods for different miss rates for the mushroom data set.

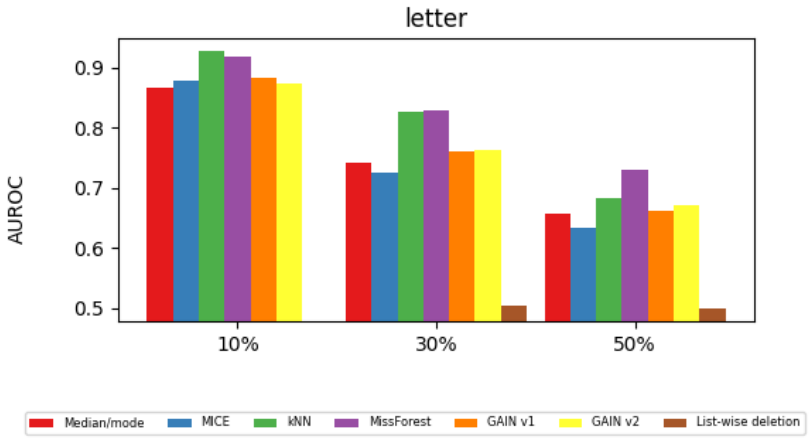

Figure 14: Evaluation in terms of AUROC for the different imputation methods for different miss rates for the letter data set.

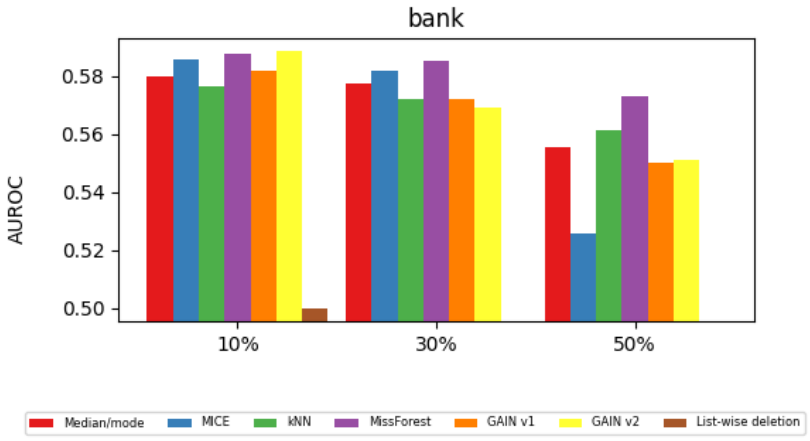

Figure 15: Evaluation in terms of AUROC for the different imputation methods for different miss rates for the bank data set.

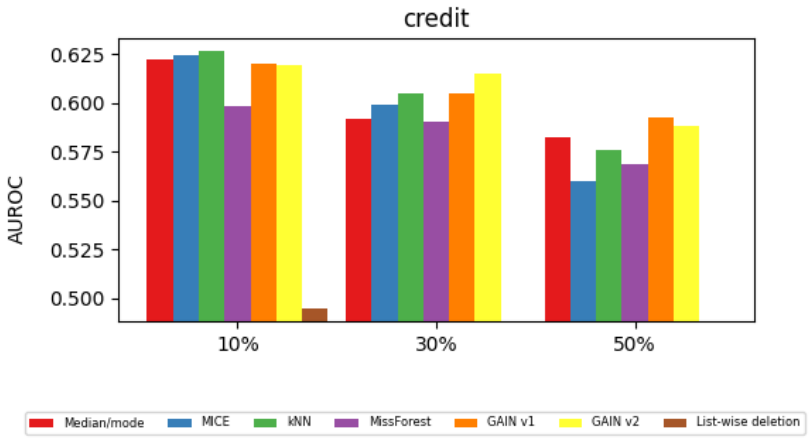

Figure 16: Evaluation in terms of AUROC for the different imputation methods for different miss rates for the credit data set.

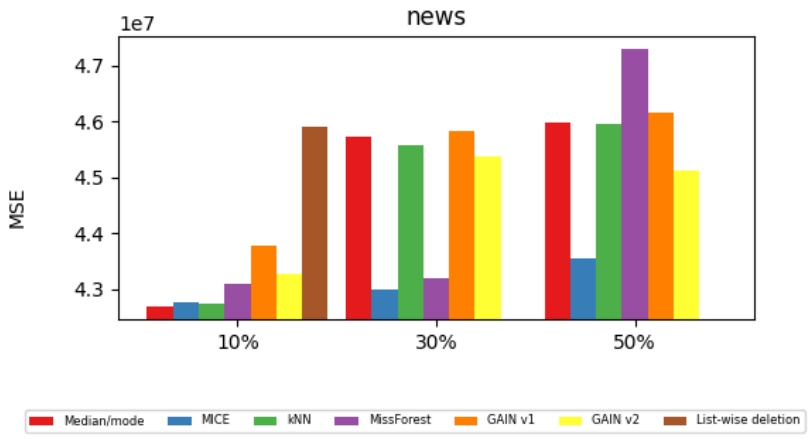

Figure 17: Evaluation in terms of MSE for the different imputation methods for different miss rates for the news data set.

Comparison of GAIN v1 to GAIN v2 Figure 18 presents the best performing method of the two GAIN versions statistically significant at a 95% confidence level for different prediction evaluation metrics. Figure 19 presents the best performing method of the two GAIN versions for different prediction evaluation metrics, not taking the confidence interval into consideration.

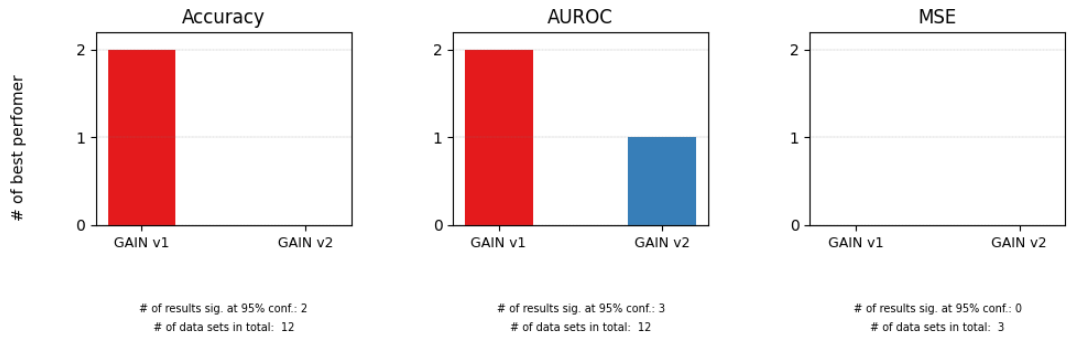

Figure 18: Number of best performer amongst the different GAIN versions statistically significant at a 95% confidence level for different prediction evaluation metrics.

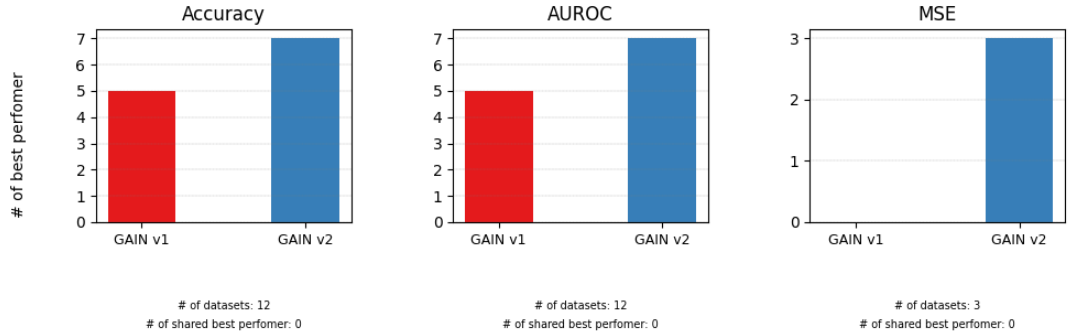

Figure 19: Number of best performer amongst the different GAIN versions for different prediction evaluation metrics.

4.2 Impact of Additional Training Data using CTGAN

This section will present the results of 'Research Question 2: How can the approach of increasing training data using CTGAN improve imputation methods?'.

4.2.1 Imputation Results

mRMSE improvements with CTGAN Figure 20 shows number of data sets that improved statistically significant at a 95% confidence level with different amounts of additional data per every imputation method. Figure 21 shows the average number of data sets that improved with different amounts of additional data per every imputation method where confidence intervals are not considered. There were nine of data sets in total in the comparison. If one two data shows exactly the same performance with and without the augmented data, it will not be counted as an improvement nor a reduction of result in the figures.

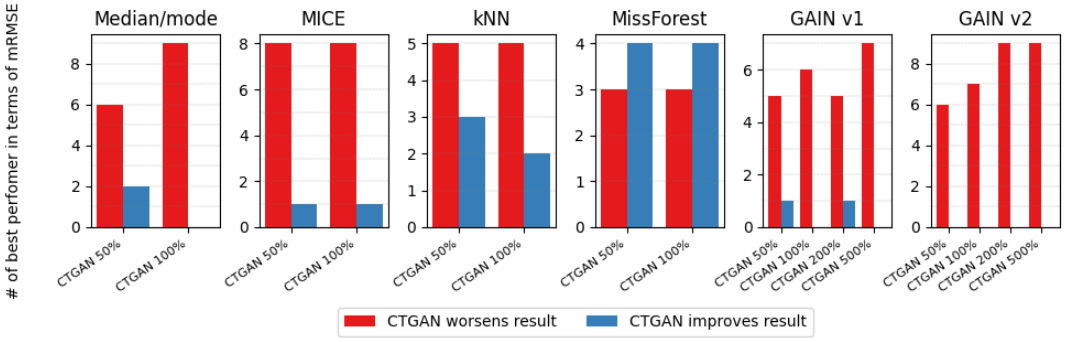

Figure 20: Number of data sets that improved statistically significant at a 95% confidence level with amounts of additional data per every imputation method in terms of mRMSE.

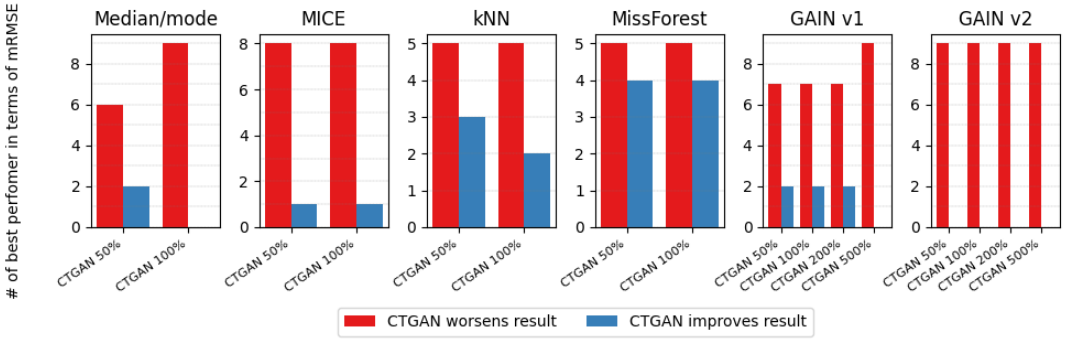

Figure 21: Number of data sets that improved with different amounts of additional data per every imputation method in terms of mRMSE, not taking confidence interval into consideration.

Best performing method in terms of mRMSE Figure 22 shows number of best performer in terms of mRMSE for every approach, different imputation methods with different amounts of additional data, statistically significant at a 95% confidence level. Figure 23 shows number of best performer in terms of mRMSE for every approach, different imputation methods with different amounts of additional data, not taking the confidence interval into consideration. There were nine number of data sets in total in the comparison.

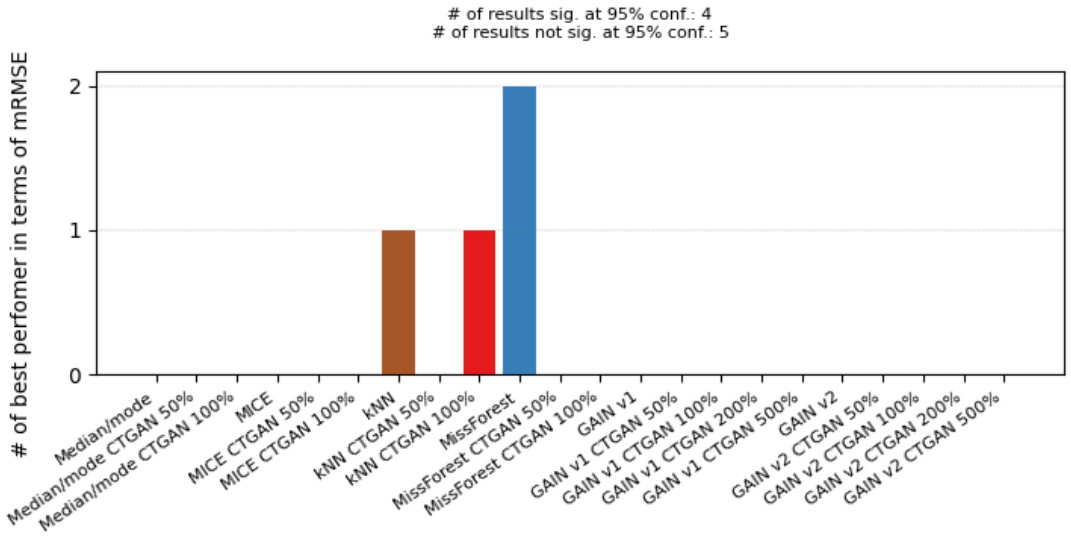

Figure 22: Number of best performer statistically significant at 95% confidence level amongst all the different imputation methods including additional data in terms of mRMSE.

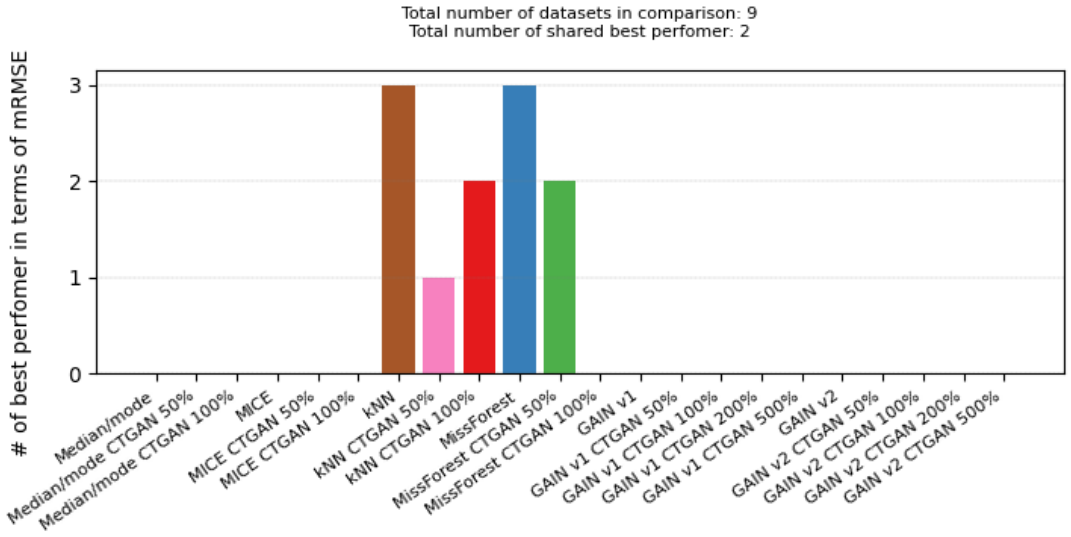

Figure 23: Number of best performer amongst all the different imputation methods including additional data in terms of average mRMSE, not taking confidence interval into consideration.

CTGAN impact per method and data set in terms of mRMSE Figures 24 to 29 present the impact of additional training data per method and data set in terms of mRMSE.

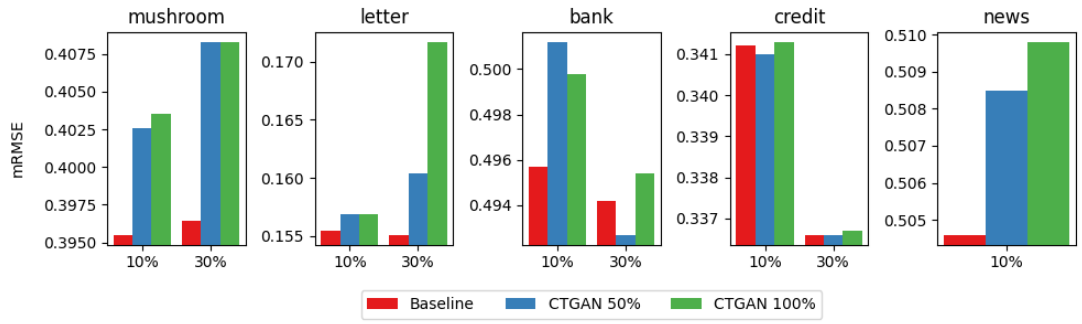

Figure 24: Impact of additional CTGAN data for Median/Mode in terms of mRMSE.

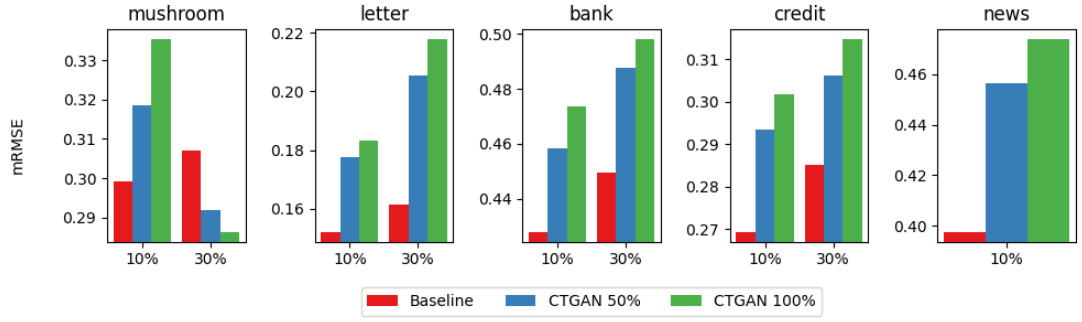

Figure 25: Impact of additional CTGAN data for MICE in terms of mRMSE.

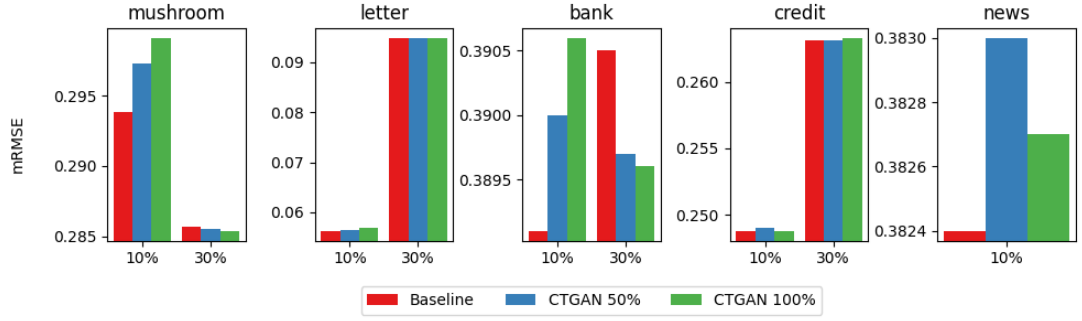

Figure 26: Impact of additional CTGAN data for kNN imputation in terms of mRMSE.

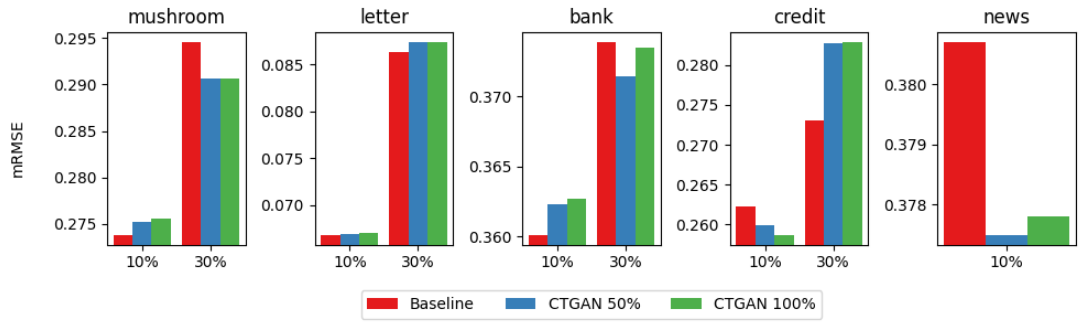

Figure 27: Impact of additional CTGAN data for MissForest in terms of mRMSE.

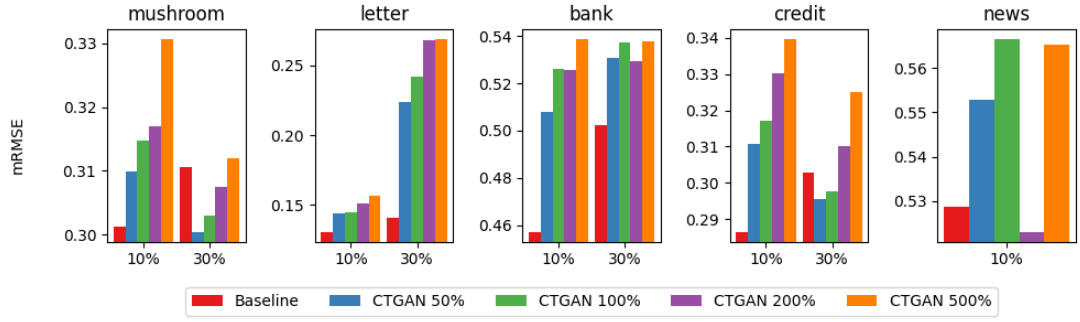

Figure 28: Impact of additional CTGAN data for GAIN v1 in terms of mRMSE.

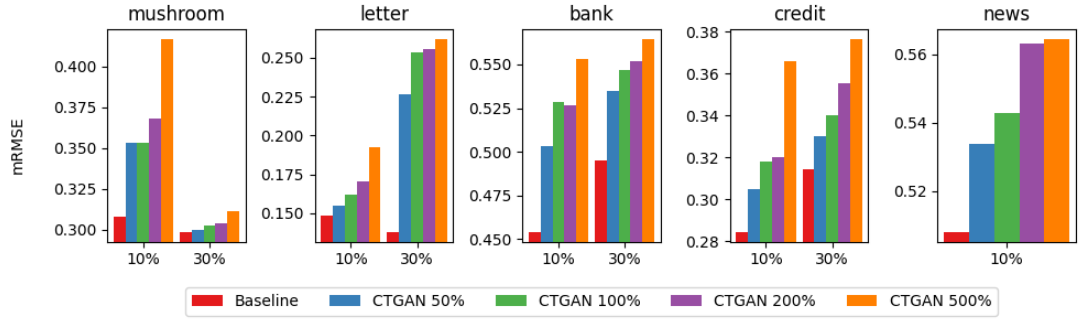

Figure 29: Impact of additional CTGAN data for GAIN v2 in terms of mRMSE.

4.2.2 Prediction Result

AUROC improvements with CTGAN Figure 30 shows number of data sets that improved statistically significant at a 95% confidence level with different amounts of additional data per every imputation method. Figure 31 shows the average number of data sets that improved with different amounts of additional data per every imputation method where confidence intervals not considered. There were eight number of data sets in total in the comparison. If one two

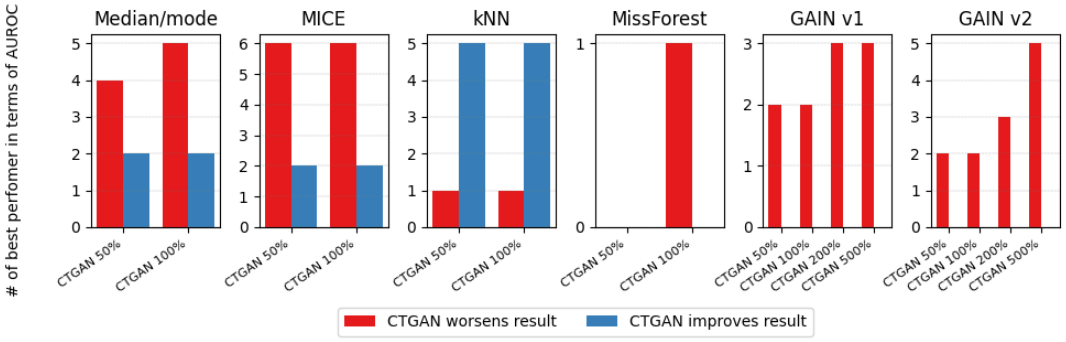

Figure 30: Number of data sets that improved statistically significant at a 95% confidence level with different amounts of additional data per every imputation method in terms of AUROC.

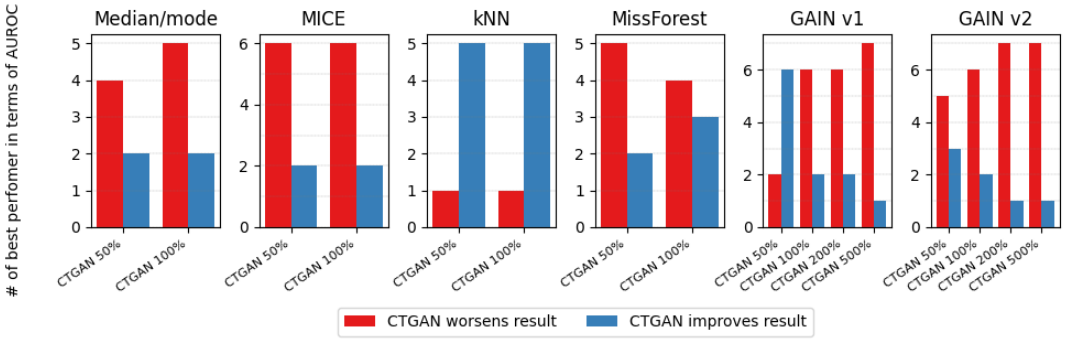

Figure 31: Number of data sets that improved with different amounts of additional data per every imputation method in terms of AUROC, not taking confidence interval into consideration.

data shows exactly the same performance with and without the augmented data, it will not be counted as an improvement nor a reduction of result in the figures.

Best performing method in terms of AUROC Figure 32 shows the number of best performer in terms of AUROC for every approach, different imputation methods with different amounts of additional data, statistically significant at a 95% confidence level. Figure 33 shows the number of best performer in terms of AUROC for every approach, different imputation methods with different amounts of additional data, not taking the confidence interval into consideration. There were nine number of data sets in total in the comparison.

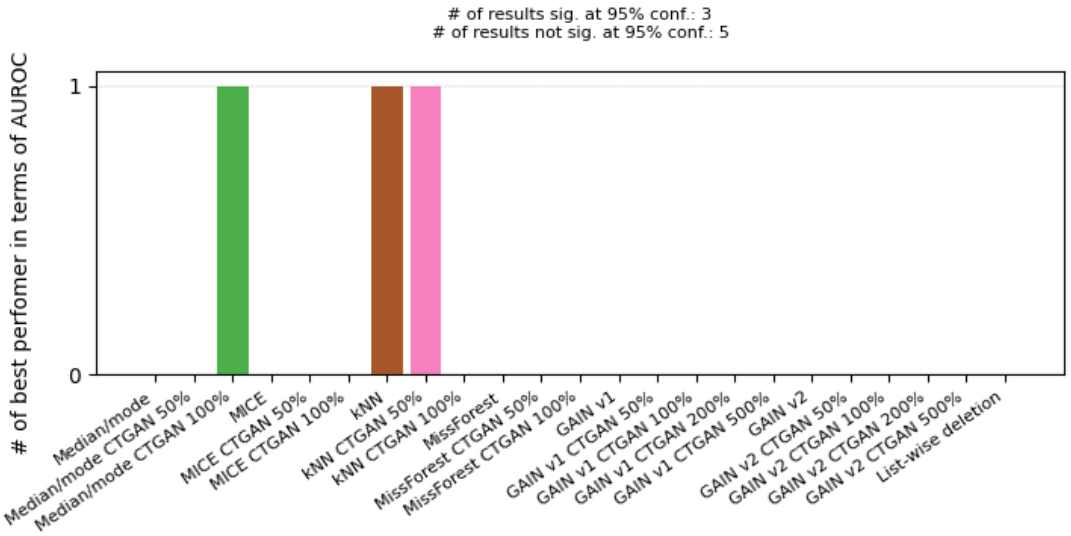

Figure 32: Number of best performer statistically significant at 95% confidence level amongst all the different imputation methods including additional data in terms of AUROC.

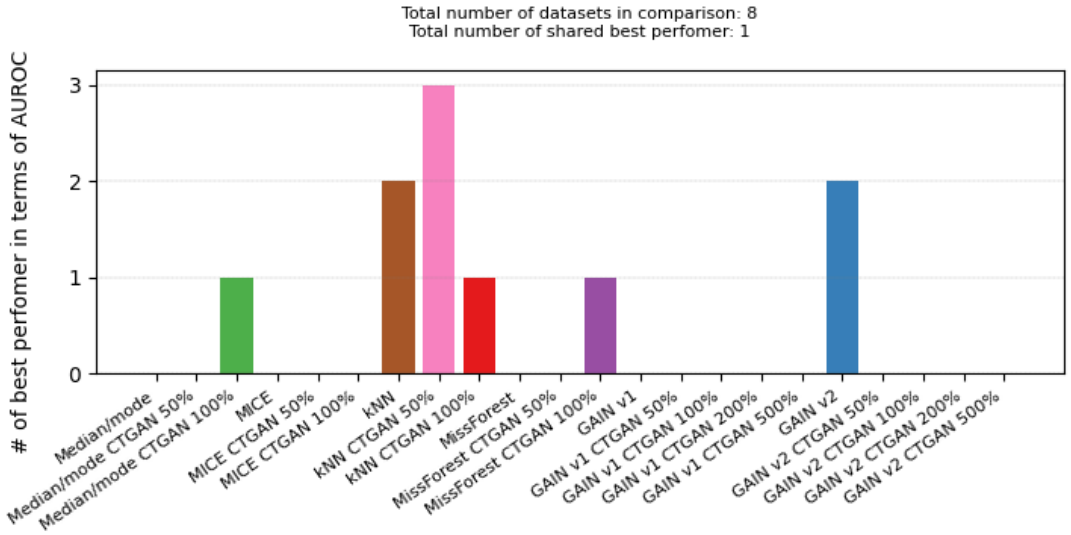

Figure 33: Number of best performer amongst all the different imputation methods including additional data in terms of average AUROC.

CTGAN impact per method and data set in terms of AUROC Figures 34 to 39 present the impact of additional training data per method and data set in terms of AUROC.

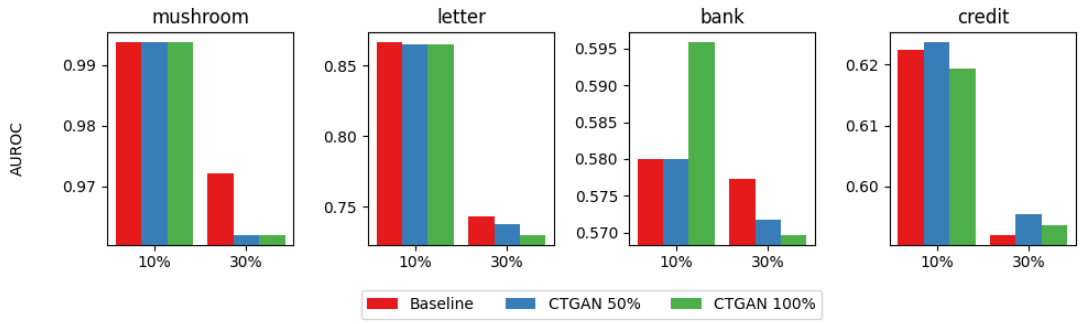

Figure 34: Impact of additional CTGAN data for Median/Mode in terms of AUROC.

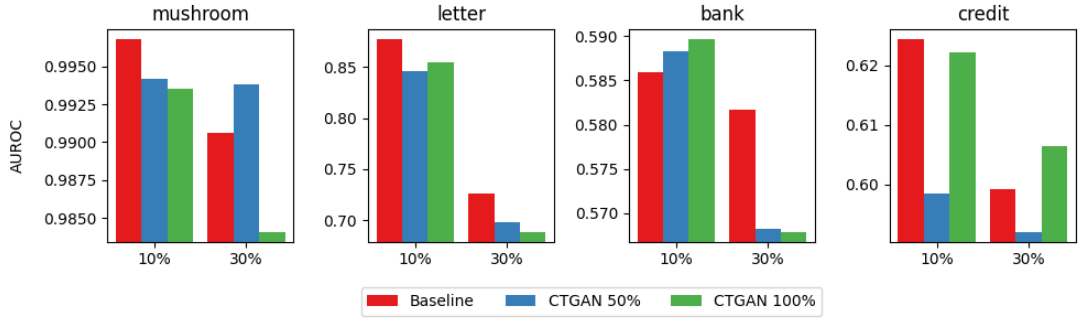

Figure 35: Impact of additional CTGAN data for MICE in terms of AUROC.

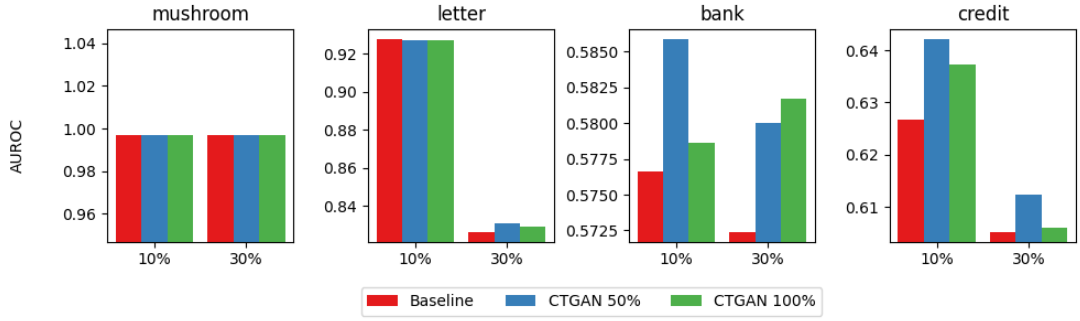

Figure 36: Impact of additional CTGAN data for kNN imputation in terms of AUROC.

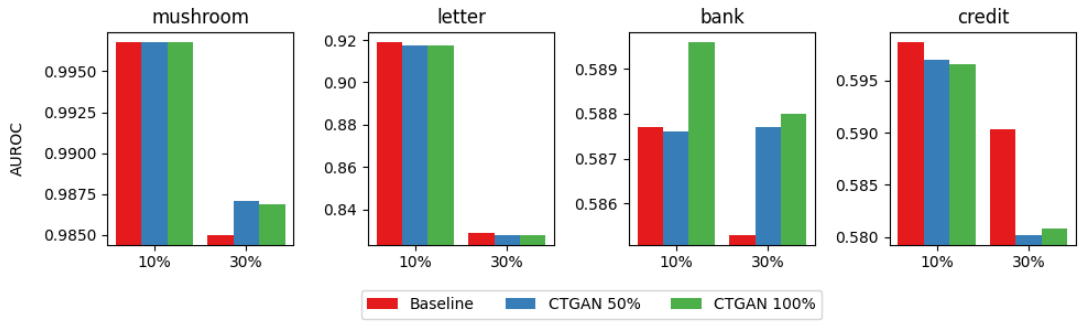

Figure 37: Impact of additional CTGAN data for MissForest in terms of AUROC.

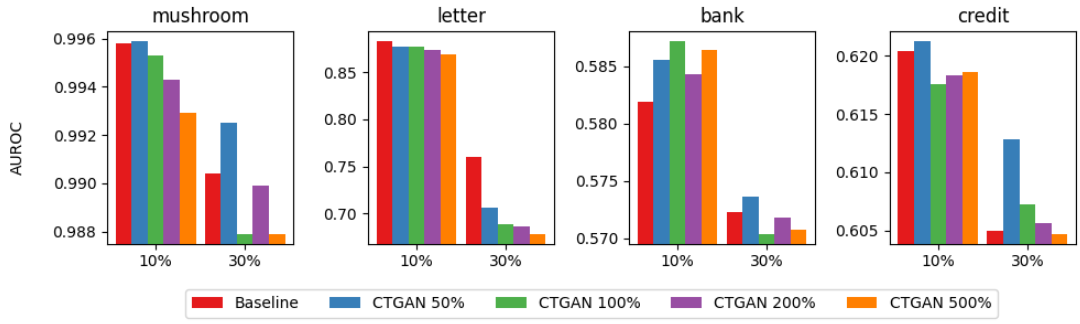

Figure 38: Impact of additional CTGAN data for GAIN v1 in terms of AUROC.

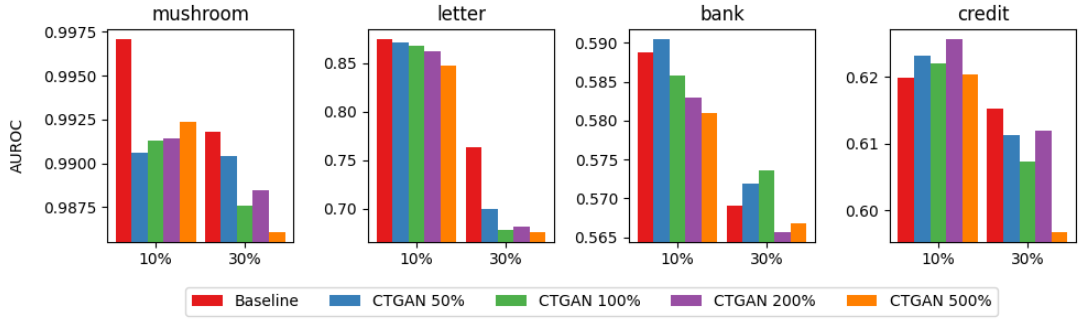

Figure 39: Impact of additional CTGAN data for GAIN v2 in terms of AUROC.

5 Discussion

5.1 Notations

To enable clear referencing of a specific data set in this section, we adopt a standardized notation denoted by *"Data set name, Miss rate %"*. For instance, the *Mushroom* data set with 10% miss rate will be represented as *"Mushroom, 10%"*. While Table 4 presents an overview of the naming convention for the *Mushroom* data set, the same approach is employed for all other data sets.

Data set	Miss rate %	Notation
Mushroom	10	*Mushroom, 10%*
	30	*Mushroom, 30%*
	50	*Mushroom, 50%*

Table 4: Explanation of the notations used for a particular data set.

5.2 Comparison of GAIN to Standard Imputation Methods

In this section, the results of the first research question will be presented where the aim is to compare GAIN to standard imputation methods.

5.2.1 List-wise Deletion

Deleting incomplete rows as an alternative to data imputation. In this study, applying the approach and then performing predictions proved to be impractical for the majority of data sets. This was due to the fact that in these cases no complete rows remained, either due to high miss rates or due to a large number of features, since the missingness was introduced at random across the features. For *Mushroom, 10%*, list-wise deletion only did yield 0.4908 in AUROC, comparing to all other methods acheiving a higher score than 0.99, as been shown in Figure 13. For the only data sets with 30% miss rate that there were complete rows remaining, *Letter* and *Bank*, the accuracy score was 0. Interestingly, for *Bank, 10%* and *Credit, 10%*, the method did yield highest accuracy score out of all methods, although the AUROC score was far from being the best performer, as presented in Figure 11. This means that the model was good at predicting the correct target value, however at the expense of a large number of false positives.

Overall, this demonstrates that data imputation methods are generally preferred over list-wise deletion, particularly when dealing with higher rates of missing data, a data set with many feature variables or when false positives must be avoided.

5.2.2 Median/Mode Imputation

The Median/Mode method showed superior performance in terms of execution time, yet was the weakest method considering imputation performance. The predictive performance was slightly better, yet still not competitive compared to the best models. Some aspects to note include:

- **Practically feasible** Median/Mode showed superior performance in terms of execution time, as presented in Figure 3. The data set with the shortest execution time was im-

puted in 0.0088 seconds, and the data set with the longest time in 0.1543 seconds with Median/Mode imputation. This can be compared to the second best method in terms of execution time, kNN imputation, which ranged from 13.0536 seconds to 1,152.8814 seconds. To conclude, Median/Mode imputation was the fastest method across all imputation methods compared in this study. The easy implementation and computational efficiency are also additional factors which can explain why this method is very popular in for practical use in industry today [45].

- **Worst imputation performance of all methods in terms of mRMSE** For seven out of 15 data sets, the method showed the highest mRMSE, indicating that Median/Mode was the poorest performing method. The model did not perform best in terms of mRMSE for any data set, as presented in Figure 3.

- **Insensitive to higher miss rates** The method exhibited consistent imputation performance, as measured by both mRMSE and AUROC, across all levels of missing data, as shown in Table 5 and Table 8 in the Appendix. This is in contrast to other imputation methods, where the mRMSE or AUROC tended to increase/decrease significantly for higher miss rates. For the highest miss rate, 50%, the Median/Mode method only showed the worst performance in terms of mRMSE or AUROC for one data set, *Mushroom, 50%*.

  The reason for this is likely that regardless of miss rates, the median or mode value is remaining similar. As a consequence, the mRMSE or AUROC will be roughly the same for all miss rates, and the Median/Mode method accordingly is better for higher miss rates compared to other methods, whose performance deteriorates for as the miss rate increases.

- **Better imputation performance for categorical variables than numerical** For the *Bank*, *Letter* and *News* data set with 10% and 30% miss rate, the RMSE for numerical variables was approximately twice as high for the Median/Mode method as the corresponding value for the MissForest method with the strongest performance, as presented by Table 5 and Table 6 in the Appendix section. However, the difference in PFC for categorical variables was not as significant, where the value for MissForest was approximately 65-75% of the value of the corresponding PFC value for Median/Mode.

- **Relatively weak predictive performance** As presented in Figure 13 to Figure 16, the Median/Mode method showed the worse performance in terms of AUROC or MSE in four out of the 15 data sets in the study. It is accordingly clear that the predictive performance for Median/Mode was not as poor as the imputation performance, where as mentioned the Median/Mode method was the worst method for seven out of 15 data sets. This observation suggests that using the median or mode value for imputation still adequately captures the overall characteristics of the complete data set, thereby enabling the predictive model to produce a satisfactory outcome, at least in relation to the imputation performance.

  As described by Figure 12 the Median/Mode method even demonstrates the lowest MSE score for one data set, *News, 10%*, as well as the highest accuracy average for one data

set, *Bank, 50%*. However, the accuracy outcome lacks statistical significance at a 95% confidence level, since GAIN v2 reports the same result. Further, Median/Mode does not show the best average performance for any data set with regards to AUROC, suggesting that the model can relatively accurately predict the correct target value, however at the expense of a higher share of false positives.

5.2.3 MICE

The MICE model demonstrated inferior performance relative to alternative imputation models with regards to both execution time and the effectiveness of imputation and prediction. Noteworthy considerations include:

- **Extended execution times for large data sets** The MICE model performed much worse than all the other models in terms of execution times, likely making it inappropriate for practical purposes. As presented by Table 5, the MICE model exhibited the longest execution time for the data set *News, 10%*, taking 121,295.4 seconds. In contrast, the second slowest model, MissForest, demonstrated significantly improved efficiency, completing the imputation process in 4,563.4597 seconds, which was more than 26 times faster than MICE.

  The reason to the extended execution times is likely that MICE is based on an iterative fitting of a regression model between the feature variables, where the number of iterations required to reach convergence can be high. Further, the model is based on *Multiple Imputation* which implies fitting a number of models in each iteration, making it computational expensive.

- **Weak imputation performance** MICE was not the best performing model on any data set in terms of imputation performance, as shown in Figures 3 and 32. Further, it was the worst model with regards to average mRMSE on two data sets, *Letter, 30%*, *Letter, 50%*, even worse than the simplistic Median/Mode method.

  The model struggled with imputing data sets with high miss rates and one possible explanation for this difficulty lies in the methodology employed by MICE with creating regressive models between the feature variables, and with high miss rates those models are likely to not succeed in representing the data in a correct way.

- **Slightly better prediction performance, yet still not competitive** MICE performed slightly better with regards to prediction performance than imputation performance, compared to the other imputation models. Especially for the *Mushroom* data set with only categorical variables MICE showed strong performance, presented in Figure 13. The model shared best performance with kNN imputation and MissForest on *Mushroom, 10%* in terms of both accuracy and AUROC, with the prediction metric presenting up to four decimal places.

  MICE also performed as the best model with regards to MSE on two data set with regression task *News, 30%* and *News, 50%*, as presented in Figure 17. However, the model was

the worse model with regards to accuracy for three data sets (*Letter, 30%*, *Letter, 50%*, *Bank, 50%*), and four data sets with regards to AUROC (*Letter, 30%*, *Letter, 50%*, *Bank, 50%* and *Credit, 50%*), as presented by Table 8.

- **Potential improvement after model tuning** Stef van Buuren, one of the authors of the paper introducing MICE in "mice: Multivariate Imputation by Chained Equations" [65], states that the performance of MICE can be improved by adapting the regression models for the different variable types. In particular for data sets with many categories there is a risk of that the number of fitted parameters grows too large in relation to number of observations. Consequently, the use of the more robust regression models that are included in the MICE package in R other than the default methods, such as cart (Classification and Regression Trees) or rf (Random Forest Imputations), can lead to improved performance.

  Nevertheless, improving MICE was out of scope for this study, and also practically infeasible due to the long execution times of the model. To exemplify this issue, it was not feasible due to practical limitations to perform the imputation of MICE 10 times and find an average and a standard deviation in this thesis, and a seed accordingly had to be used.

In conclusion, the MICE model showed worse performance compared to alternative methods used in the study such as kNN imputation and MissForest. One potential reason is that MICE has troubles handling non-linear relationship between feature variables, since it is based on fitting regression models between the features in order to impute the missing values. Further, the extended execution times of MICE leads to that the usability of the model for applications in industry can be questioned.

5.2.4 kNN Imputation

The kNN imputation model was the best model considering predictive performance, and performed second best after MissForest with regards to imputation performance as presented by Figures 3 and 12. Additionally, it is generally considered intuitive and was considerably faster than MissForest, leading to that it could be the most suitable method for practical applications. A few aspects to consider include:

- **Strong imputation and predictive performance** kNN imputation was the second best model in terms of imputation performance after MissForest, and particularly strong for categorical variables. The model was the best performer for five out of the 15 data sets when evaluating in terms of mRMSE, one data set when evaluating in terms of RMSE for numerical variables and five data sets with regards to PFC, as presented in Figure 3.

  When evaluating the predictive performance, kNN imputation was the best model with regards to both accuracy and AUROC. As presented in Figure 11, it was the best model in terms of accuracy for two data sets and of AUROC for three data sets when evaluating the statistically significant results at a 95% confidence level. However, when examining the average values, MissForest was the best method in terms of AUROC as presented by

Figure 12. It can accordingly be concluded that the two methods were both strong and performed very similarly.

- **No variability** A notable advantage of the kNN imputation model is that once the number of neighbors k have been determined, there is no variability in the imputation results. Additionally, considering that the execution time of kNN imputation is generally shorter compared to MissForest, which produced the highest result, the model becomes beneficial for practical applications as it only has to be run once.

- **Hyperparameter sensitivity** As presented in Section 2.3.1, kNN imputation is sensitive to the choice of number of neighbors k. The selection of the optimal value for k in this study is determined through cross-fold validation with the chosen k corresponding to the number yielding the highest AUROC score for the validation data set. Nonetheless, there are a number of ways found in literature to determine k, and examining how different values of k would impact both the prediction and the imputation result would require substantial efforts, being out of scope of this study. Nevertheless, it is evident that the value used for each data set in this study impacts the result, which can be seen as a disadvantage of the method, as it accordingly requires tuning.

There are several compelling reasons that make kNN an advantageous imputation method for practical applications, despite showing the second best imputation performance after MissForest. Firstly, kNN is acknowledged for its simplicity and ease of understanding, as the imputed values are intuitively derived from the most similar observations. Secondly, the strong result is likely a consequence of that kNN can handle complex relationships between variables in a robust way, since the imputations are only based on the neighbor's values. Lastly, the method is fast and is only run once, being an important aspect in practical applications.

5.2.5 MissForest

The MissForest model showed the strongest performance with regards to imputation performance of all models, and it was the second best model with regards to imputation performance measured by AUROC as presented by Figures 3 and 11. There are several noteworthy aspects to consider, including:

- **Strongest imputation performance** MissForest showed the strongest imputation performance with regards to both mRMSE, RMSE for numerical variables as well as PFC for categorical variables at a 95% confidence level, as is shown in Figure 3. The model was particularly superior for numerical variables, showing the lowest RMSE for numerical variables for all data sets except one, the *Letter, 10%* where kNN imputation performed better. When evaluating the imputation models in terms of PFC, MissForest was not as superior, yielding the best result for six out of 12 data sets followed by kNN imputation which was the best performer for five out of 12 data sets, yet still at a 95% confidence level.

- **Strong prediction performance for classification task** Of the 12 data sets which aimed to classify a target variable, MissForest showed best performance in terms of AUROC for two data sets and best performance in terms of accuracy for one data sets, presented in in Figure 11. It was accordingly the second best method with 95% certainty in terms of number of data sets for which it performed best regarding prediction performance, as kNN imputation was slightly better. In contrast, when observing only at the average AUROC and accuracy values, MissForest was the best method as presented by Figure 12. It was then the best method for three out of 12 data sets considering accuracy performance, and four out of 12 data sets considering AUROC performance. This means that especially kNN imputation and MissForest both showed strong performance and performed very similarly.

  Despite this, when considering the *News* data set with a regression task, MissForest did not show as strong performance as it was not the best model for any miss rate.

- **Impact of use of standard parameters** There are a number of parameters that can be tuned when implementing MissForest, such as maximum numbers of iterations and number of trees in the forest of the random forest models. Tuning the model was out of scope for this study, indicating a potential for further performance enhancement of MissForest.

There are several possible reasons as to why MissForest showed such strong performance, both with regards to imputation and prediction abilities. Firstly, as opposed to MICE assuming a linear relationship, MissForest is known to be able to handle and capture complex relationships between feature variables. The reason to this could be the fact that MissForest is an ensemble learning based method that is formed by multiple decision trees, making the method able to preserve the original distributions when imputing the data, which presumably yields better both imputation and prediction performance. The ability of MissForest to handle large data sets is also an advantage, even though the method is slower in regards to execution time than alternatives such as kNN imputation.

5.2.6 GAIN

Both versions of GAIN were significantly outperformed by other methods across the different data sets and miss rates in terms of imputation performance. The predictive performance was comparatively better but still not as much as affecting the ranking of the imputation method compaared to the alternatives methods in the study.

Poor imputation performance The two versions were even performing worse in terms of mRMSE than the basic Median/Mode imputation for six out of 15 data sets including *Bank, 30%*, *Bank, 50%*, *Credit, 50%*, *News, 10%*, *News, 30%*, and *News, 50%*, as presented in Table 5 and Table 7 in Appendix. As seen in Figure 3, GAIN v2 was not the best performer for any data set in terms of any evaluation metric. GAIN v1 was only the best performer for one data set in terms of mRMSE and one data set in terms of RMSE cat and PFC.

Enhanced predictive performance Despite the insufficient imputation efficiency, the predictive performance was comparatively better. Either GAIN v1 or GAIN v2 were better than the Median/Mode method in terms of accuracy for all data set except for two. Further, when examining the average values both GAIN versions were the best method out of all imputation methods in terms of accuracy for two different data sets out of 12, as presented in Figure 12. GAIN v1 was the superior model for *Mushroom, 50%* and *Credit 50%*, and GAIN v2 for *Mushroom, 10%* and *Bank, 10%*. Both GAIN versions were the best model for these same two data sets when evaluating based on AUROC as well, and in addition to this GAIN v2 was also the best model for *Credit, 30%*. In total, one of the two GAIN versions were the best method out of all imputation methods for five out of 12 data sets when evaluating based on AUROC, evaluating based on the average values. However, the variability of GAIN was comparatively high, and it was accordingly not possible to state that a GAIN model was the best models for a data set with 95 % certainty, explaining the result in Figure 32 where no GAIN version is the best performer.

Challenge to converge GANs are known for their challenging training process, where both the hyperparameters and model architecture are significantly impacting performance [2]. Making a GAN network converge has proven to be difficult due to its structure with two neural networks, since enhancing the performance of one network often comes at the expense of the other. To exemplify this in the case of GAIN, the objective of the generator is to impute samples that are so realistic that the discriminator can only classify if they are real or imputed with 50% accuracy. However, if the GAIN framework continues training when the generator performs much better than the discriminator, the discriminator will provide random feedback which lead to that the overall quality of the network may collapse. Conversely, if the discriminator performs too well, the generator may struggle to leverage the feedback effectively, hindering its ability to discern the necessary improvements required for generating data that the discriminator believes are sampled from the true distribution. This, in turn, can also contribute to a significantly decreased quality of the network.

In this study, there are several possible explanatory reasons for the overall performance of GAIN, namely:

- **No convergence** As seen in the Appendix, the test loss did not converge when training the GAIN model and the loss function is behaving unstable. This entails that the model is likely to not generalize well on unseen data, and since test and training data sets are used in this study, this could be one explanatory reason to GAIN's overall poor performance. This is further strenghtened by the fact that for several papers showing solid GAIN performance, no training and test split is done as seen in Table 1.

- **Potential use of non-optimal hyperparameters** The hyperparameters in this study were selected based on cross-fold-validation, but optimally a larger grid search could have been used to further optimize the parameters per data set. Additionally, the number of training iterations is a parameter which have not been tuned in this study for GAIN v1,

and instead the standard number of iterations from the original GAIN paper has been used. To conclude, the fact that it is not certain that the global optimal hyperparameters were found could potentially explain the poor performance of GAIN in this study.

- **Model not fully adjusted to categorical data** As the original code implementation does not adapt an activation function in the output layer appropriate for categorical data, the GAIN v1 model was expected to show poor categorical imputation performance. This situation occured since GAIN v1 demonstrated poor overall imputation performance, including for categorical variables. However, despite the fact that the model GAIN v2 was developed with an adapted activation function it did still not show performance improvement for categorical variables. This implies that GAIN's ability to handle categorical variables likely could be optimized further, and can be one reason to the models' poor performance.

- **Variations in study set-up** Compared to previous studies which have proven that GAIN performs better than other imputation methods, this study introduces missing values among categorical variables in a more realistic way. This implies that missing values are introduced before the data set has been one-hot encoded just as in real life scenarios when some values are missing, rather than after. This variation in study set up can also be a potential explanatory reason as to why GAIN demonstrates worse performance than other imputation methods in this study.

Potential of GAIN With this reasoning in mind it is essential to highlight that despite the findings of this study, it is not certain that GAIN can not outperform other imputation methods. There are a number of potential improvements of GAIN that can be investigated, and some of them are likely to yield an improved result compared to the original version. Firstly, as with all GAN networks it is possible to adapt the GAIN architecture to fit each data set. This for example includes adding or removing number of layers in the generator's or discriminator's networks, or adapting the size of the already existing layers. Secondly, it is possible to add, remove or adapt different component of the network to mitigate the impact of the model's weaknesses. The caveats of the original GAN framework mentioned in Section 2.2.4 are applicable for the GAIN model as well, and are the motivation behind introductions of more sophisticated models such as SGAIN, WSGAIN-CP, and WSGAIN-GP [47]. These have shown to outperform the original GAIN model and could therefore be interesting to further investigate.

Despite the possible improvements that have not been implemented in this study, it clearly shows the difficulties associated with training GANs, and more specifically GAIN. In particular, the fact that this study tuned the hyperparameters through an extensive optimization using cross-fold validation, as well as suggested several improvements presented as GAIN v2 emphasizes the challenges in training the GAIN model, especially for it to reach a level that outperforms other state-of-art imputation methods.

5.3 Comparison of GAIN v1 to GAIN v2

Similar in regards to imputation performance GAIN v2 was modified with the aim to improve the handling of categorical variables. The *Mushroom* data set only contains categorical variables and was anticipated to exhibit the most notable performance enhancement for GAIN v2. In terms of the evaluation of PFC on the *Mushroom* data set, GAIN v2 was only superior in terms of average PFC for miss rate 30%, as presented by Figure 4, yet it was still not possible to state that GAIN v2 was better than GAIN v1 for this data set with 95% certainty. On the other hand, it was possible to state that GAIN v1 was better than GAIN v2 in terms of PFC with a confidence level of 95% for two data set, while it was only possible to state the opposite for one data set. To sum up, GAIN v2 did not handle categorical variables better than GAIN v1.

When considering the statistically significant result presented in Figure 9, it is clear that regarding imputation performance the two methods showed very similar results, where GAIN v1 was superior for three data sets in terms of mRMSE, while the corresponding number of data sets for GAIN v2 was two. For the majority of the data sets it thus was not possible to state which method that was superior with 95% certainty, due to the wide and overlapping confidence intervals. Despite this, when considering the average mRMSE values as presented in Figure 10 GAIN v2 performs better than GAIN v1 in terms of mRMSE for nine out of the 15 data sets.

Similar in regards to predictive performance When considering prediction performance, the result was very similar to the imputation performance result and it was accordingly difficult to statistically significantly state which method of GAIN v1 and GAIN v2 that was superior. In terms of AUROC, GAIN v1 outperformed GAIN v2 with 95% certainty for two data set, and the corresponding number for GAIN v2 was one data sets as presented by Figure 18.

Observing the average values, GAIN v2 is superior than GAIN v1 for all miss rates for the *News* data set. For the *Mushroom* data set, it performs better than GAIN v1 for lower miss rates, 10% and 30%. In summary, when measuring the average AUROC score or MSE score, GAIN v2 performs better than GAIN v1 for 10 out of the 15 data sets in the study, as presented by Table 10. However, it is not possible to state this result at a 95% confidence level.

Conclusion and further improvements Considering the significance and emphasis placed on hyperparameter optimization, it becomes evident that there is potential for further improvement for GAIN v2. This arises from the fact that the optimal values for *batch-size*, *alpha*, and *hint-rate* were directly inherited from the GAIN v1 version, with only the additional hyperparameters *beta* and *tau* being optimized. In this study, GAIN v1 and v2 performs very similarly and the development of GAIN v2 does not alter the ranking of the model when compared to other imputation methods.

However, GAIN v2 is outperforming GAIN v1 with regards to imputation time, since the optimum for the model was found for approximately 3000 iterations instead of 10000 iterations as the original GAIN implementation. This is a strength if applying GAIN since both of the

models experience comparatively high variability of the results, and running the imputation several times accordingly seems beneficial.

5.4 Impact of Additional Training Data using CTGAN on Different Imputation Methods

In this section, the results of the second research question will be presented where the aim is to analyze the impact of additional training data for the different imputation methods.

5.4.1 Median/Mode

Mainly reduced performance when adding training data For Median/Mode imputation, additional data only improved the direct imputation performance in terms of mRMSE for two data sets, *Bank, 30%* and *Credit, 10%* as presented by Figures 20 and 24. For both of these, 50% additional data improved the result while 100% additional data worsens the result compared to no additional data. For the remaining data sets, the addition of training data resulted in a worse result in terms of mRMSE. In terms of predictive performance, both 50% and 100% additional data exhibit improvements on two distinct data sets as described by Figure 34 and 30. Specifically, the performance of *Credit, 10%* and *Credit, 30%* is enhanced by 50% additional data, while 100% additional data yields improved results for *Credit, 10%* and *Bank, 10%*. For the remaining data sets, the addition of data had a negative effect on the predictive performance.

Potential reasons for negative impact of augmented training data Considering CTGAN's objective to preserve the distribution of the initial training data set during data synthesis, it may appear somewhat odd that the introduction of additional training data affects the outcome. One plausible explanation for this could be that the CTGAN model fails to capture the true underlying distribution of the initial training data. Adding more observations thus leads to that the mode or the median of each feature variable changes, which leads to that the imputed values with CTGAN are not the same to the imputed values without the augmented data. When the median or mode changes for one or more columns the imputation and predictive performance will undoubtedly also change.

The fact that adding CTGAN training data had a negative impact on the direct imputation and predictive performance for the majority of the data sets indicates that the augmented training data accordingly has a different distribution than the original training data. As the imputed values stems from another distribution than the true distribution, the predictive performance reduces, and they are accordingly not as likely to predict the target variable.

5.4.2 MICE

Mainly reduced performance when adding training data MICE aims to capture the relationships in the data for the observed values, and with more data the model could learn from a larger sample size, and thus potentially improve the task. However, in the study, MICE

does not achieve better imputation nor prediction performance with additional training data generated by CTGAN for the majority of the data set, as seen in Figures 20, 25, 30 and 35. The *Mushroom, 30%* is the only data set that is improved in terms of both mRMSE and AUROC. The *Bank, 10,%* and *Credit, 30, %* data sets are the only data sets that improved in terms of prediction performance. For the two latter, 100% additional data is superior than adding 50% data.

Potential reasons for negative impact of augmented training data There are several possible explanations to why adding training data using CTGAN has a negative impact on the performance of MICE. One possible is the aspect already discussed for Median/Mode Imputation, namely that the CTGAN model fails to accurately capture the underlying distribution. For MICE, this indicates that the regression model that aims to predict the missing values of a particular feature is impacted by data which does not fully represent the true relationship between the features, resulting in a model which does not reflect the true data. This is especially the case if the generated data contains outliers, as these observations are more likely to impact the full model. Consequently, this situation increases the probability of incorrect imputed values.

Another aspect worth mentioning is that even if the training data were to enhance the performance of MICE, it is likely not feasible from a practical standpoint. With 50% additional data, the execution time almost doubled for some data sets. Without the augmented data, MICE was the significantly slowest method in terms of execution time, and it is therefore to justify a decision that makes it even slower.

5.4.3 MissForest

Various impact on direct imputation performance The MissForest algorithm shows various outcomes regarding direct imputation performance for additional data. Out of the nine data sets examined, it can be stated with 95% certainty that adding the augmented data improves the result for four data sets, while it has a negative impact on the mRMSE result of three data set. When examining the average values, the augmented data improves the result of four data sets and worsens the result on five data set, presented in Figure 27. To sum up, it is not clear whether the augmented data improves the result or has a negative impact, and the conclusion can thus be made that it does not impact the imputation performance of MissForest drastically.

The data sets that are improved includes *Mushroom, 30%*, *Bank, 30%*, *Credit, 10%*, and *News, 10%*, and no correlation between a specific miss rate and improvement from additional data can accordingly be found. Further, it is not possible to state if 50% or 100% additional data impacts the result the most in terms of direct imputation performance and mRMSE, since this differ between the data sets for which the performance is improved.

Limited impact on predictive performance Similarly to the imputation performance, it is not possible to state with 95% certainty that the additional data improves the result of any data set when evaluating the AUROC performance. It is only possible to state with 95% certainty

that the 100% additional data has a negative impact on the result for one data set, as presented in Figure 30.

When examining the average AUROC values, adding 50% of additional data implies that AUROC is reduced for five out of nine data sets, while the result is identical for two data set and higher for two data sets as presented by Figure 31. The corresponding value when adding 100% is four out of nine improving, three is negatively impacted and two data sets showing the same performance with or without CTGAN data.

Potential reasons for limited impact of augmented data The result of MissForest when adding CTGAN data is comparatively vague, and adding the data overall does not have a significant impact on the result for the majority of the data sets. In some situations the augmented data has a slightly negative impact, which can likely be explained with a similar reasoning as for Median/Mode and MICE, that the CTGAN model fails to fully capture the true underlying distribution of the original data.

The reason to MissForest appearing less impacted compared to Median/Mode and MICE could be that MissForest does not directly use the full distribution of the data when fitting the random forests and thus estimating the imputed values. Instead, the model chooses subsets of features and observations to train each decision tree in the model through random feature selection and random sampling. The slightly negative impact of adding more data could be a result of the fact that if the distribution of the augmented data is altered, the distribution of the subset is also likely to be slightly affected.

5.4.4 kNN Imputation

Enhanced result with additional data The kNN imputation shows significant improvement with additional data. The performance improvement with the augmented data is most significant when examining the prediction result, as seen in Figure 30. Adding 50% data improves the prediction result in six out of nine data sets, and remains the same for two out of the nine data sets. The *Letter, 10%* is the only data set for which adding 50% data has a negative impact, but this difference is considered very small in terms of magnitude, corresponding to a decrease in AUROC of 0.06%. When adding 100% data, the prediction result improves in five out of eight data sets, and remains the same for two out of the nine data sets. It is only the *Letter, 10%* and *News, 10%* that performs worse with 100% additional data, again with the observed difference considered to be minimal. For direct imputation performance and for a low miss rate, 10%, none of the data sets gets improved in terms of mRMSE with additional data. However, for 30% miss rate, four out of the five data sets improves or remains the same with additional training data.

The additional data strengthens kNN Imputation's position as a leading performer When assessing the performance of all methods in terms of direct imputation performance and mRMSE, Figure 22 demonstrates that kNN imputation and MissForest without additional data are the best performers. Introducing additional data to these methods have improved the

results for certain data sets, however not for all and both kNN imputation and MissForest are still considered to be the best method in terms of imputation performance.

After comparing all imputation methods in terms of predictive performance and AUROC with the inclusion of additional training data, it is evident from Figure 32 that kNN imputation consistently remains one of the top-performing methods. However, it is important to note that it is only possible to determine with 95% confidence which method is superior for a few data sets. This implies that multiple methods perform similarly well. Despite this, if one were to select a single imputation method as the best in this study, kNN would be the preferred choice.

Furthermore, it is worth mentioning that if both kNN imputation without additional data and kNN imputation with additional data exhibit the same best AUROC value in Figure 32, neither of them is considered statistically significantly superior, even though they might outperform all other imputation methods. This suggests that the method has more instances of being the best performer than what is depicted in Figure 5, and it is therefore beneficial to consider the average best values when determining the optimal method.

When examining the average AUROC values, any kNN method is the best performing method for six out of the eight data sets in comparison as presented in Figure 33. In the study without additional CTGAN data, kNN imputation was only the best method for three out of 12 data sets as presented by Figure 12. The additional training data thus strengthens kNN imputation's position as s top performer in terms of predictive performance in this study.

Potential reasons for positive impact of augmented data The discovery that incorporating training data through CTGAN substantially enhances predictive performance for kNN imputation is an interesting founding, particularly since the utilization of augmented training data in combination with the other methods indicates that CTGAN fails to fully capture the underlying distribution of the original data in the generated data. Additionally, it is worth noting that the model with additional data has improved despite the fact that the number of neighbors, k, has been optimized using cross-validation on the data set without augmented data. This approach was chosen since determining a k for the increased training data is not possible without synthesizing a target variable.

To understand why CTGAN improves kNN imputation, it is crucial to fully comprehend the basic nature of kNN imputation. The method imputes the missing values on the basis of the non-missing values of the k nearest neighbors in the feature space. Adding more data is likely to add more neighbors to an observation, resulting in that all observations likely gets surrounded by more observations which are similar to themselves. An observation which already is surrounded by many other observations, i.e., an observation that is not an outlier, is likely to not be as impacted by the additional data when its missing values are imputed. However, the augmented data could result in that even the outliers are surrounded by more observations which are similar to themselves. When imputing the value of an outlier with the mean or mode value of the non-missing corresponding values of the k nearest neighbors, it is then likely that this value is closer to the true value, since the k neighbors now resembles the observation more. This reasoning could be the explanation as to why the predictive performance is clearly improved

with the augmented data.

One last notable conclusion is that augmenting more data does not necessarily lead to improved performance. Surprisingly, the addition of 50% additional data outperforms the results obtained by adding 100% data, although this difference is minimal. One possible reason for this is that when there has been enough data generated so that the majority of all observations to have k close neighbors, the result will not be further improved. At this point, more observations generated by CTGAN is likely to have a negative impact, since the probability that the close neighbors of one observation are sampled from this distribution and not the true distribution then increases. Since it is likely that the distribution generated by CTGAN is to be differing from the original data distribution, this will lead to that the added neighbors are not an as good approximation of the missing values as the original neighbors.

5.4.5 GAIN

An increased amount of training data did not improve performance significantly for neither of the GAIN versions even though GANs have previously seen to benefit from a larger amount of training data. Further, adding larger amounts of data, i.e., adding 500% instead of 50%, had a more negative impact on the result.

GAIN v1 There are only a few data sets for which the additional CTGAN data improved the imputation performance for GAIN v1. Instead, for the majority of the data sets there seems to be a decreased direct imputation performance the more additional data that was added as presented by Figure 20 and 21. Specifically, adding 500% data is performing the worst for all data sets except for *News, 10%* where it is the second worse performer. The data sets *Mushroom, 30%*, *Credit, 30%*, and *News, 10%* are the only data sets that benefit from additional data in terms of average lower mRMSE, but only for additional amounts 50%, 100%, and 200%. It was only possible to state that adding CTGAN improved the result with 95% certainty for one data set when adding 50% and 200% CTGAN data, as presented in Figure 20. For 100% and 500% additional data, it was not possible to reach this level of certainty for any data set. It was on the other hand statistically significant that adding CTGAN decreases imputation performance in terms of mRMSE for five out of nine data sets when adding 50% of data, six out of nine data sets for 100%, five out of nine data sets for 200%, and lastly seven out of nine data sets for 500%.

In contrast, when considering the average AUROC or MSE in terms of predictive performance, adding 50% CTGAN data improves the average value seven out of nine data sets. It is only the *Letter* data set for all miss rates that does not improve with this amount of additional data as presented by Figure 38 and Tables 15 and 16 in the Appendix. The amount 50% data is however the only extra amount of CTGAN data that seems to have a positive impact on the average AUROC or MSE. Adding 100%, 200%, or 500% either has a negative, or only a small positive impact, as presented by Tables 14 to 19. Despite the increase in average AUROC for GAIN v1 when adding 50% additional data, it was not possible to state with 95% certainty that the result was improved. As seen in Figure 30, there was no data set for which it is possible to

state that the AUROC value was improved with 95% certainty for any amount of CTGAN data. The reason for this is likely the comparatively wide confidence intervals of the GAIN method.

GAIN v2 Regarding imputation performance, none of the data sets are showing lower mRMSE for any amount of additional training data for GAIN v2 as presented in Figure 20. All of the data sets show the same pattern of demonstrating a reduced performance the more additional data that is added.

A few data set show higher average AUROC score with additional data, as seen by Figure 31. The *Bank* data set improves with additional data for both 10% and 30% miss rate and the best performer for the *Credit* data set is the *Credit, 10%, 200%*. Although these are exceptions and the majority of data sets does not improve in terms of prediction. Further, as presented in Figure 30, it can not be stated with 95% certainty that any of the data sets show an increase in performance.

Potential reasons for negative impact of augmented training data One potential reason to model's poor performance is likely due to its nature, making GAIN is very sensitive to changes in the true distribution. In the GAIN model, the goal of the generator is to learn the true data distribution and it is therefore crucial that the augmentation by CTGAN data have minimal impact on the overall data distribution. This finding further reinforces the hypothesis that CTGAN failed to fully capture the true distribution, consequently adversely impacting the performance of the GAIN model when augmented data is incorporated.

5.4.6 CTGAN

In summary, the only imputation method that showed significant benefit from additional training data generated by CTGAN was kNN imputation and all other methods were on the contrary negatively impacted by additional data. As previously suggested, this could be due to the lack of ability for the CTGAN model to capture the true data distribution.

Impact of small data sample The quality of data generated by CTGAN is dependent on the quality that the model was sampling from. For all data sets with 50% miss rates, no complete data rows were remaining and thus no sampling could be done. For data sets with many dimensions, in this study the *News* data set, miss rates on the level of 30% also left no complete data rows. This finding reveals that this approach of using augmented data with CTGAN is limited to data sets with lower miss rates. Even for lower miss rates, the number of complete data rows could remain small and thus affecting the ability for CTGAN to sample data similar to the true distribution due to the limited sample size.

Issues with complex distributions and the lack of hyperparameter tuning Even with a larger data set to sample from, CTGAN has shown difficulties in sampling complex distributions. Especially for mixed data type variables, long-tail distributions and skewed multi-mode continuous variables Zhao *et al.* [78] showed that CTGAN fall short in solving these

challenges and leaving room for improvement for the synthetic data generation. The absence of hyperparameter tuning for the CTGAN model in this study could be another contributing factor to the results obtained. The hyperparameters were adapted as they were proposed in the original paper, without further optimization or fine-tuning. CTGAN aims to fully capture the original data distribution, although it inevitably introduces certain alterations to the distribution. This effect becomes more noticeable when dealing with a limited sample size and complex distributions in the sampled data. The imputation models demonstrate varying levels of sensitivity to these alterations, shown in their response to additional training data.

6 Conclusions and Future Work

6.1 Comparison of GAIN to Standard Imputation Methods

This study aimed to investigate how well a GAN network, namely GAIN, performs in regards to data imputation as compared to other state-of-art data imputation methods. Two versions of GAIN, including the original model and an updated version tailored for categorical variables, were evaluated alongside the other imputation techniques. The updated GAIN version incorporated additional modifications such as an updated activation function, a tuning parameter for categorical loss, and an updated rounding function.

In terms of imputation accuracy, measured by RMSE for numerical variables and PFC for categorical variables, both versions of GAIN were found to be outperformed by MICE, MissForest, and kNN imputation, and in some cases even by the basic method Median/Mode. However, when considering predictive performance measured by AUROC, both GAIN versions showed comparatively stronger performance. When examining the average values, one of the two version was the best performing model out of all imputation methods in this study for five out of 12 data sets. However, the high variability of GAIN makes it impossible to state this result significant on a 95% confidence level.

A comparison between the two GAIN versions showed that the modifications in the updated model did not yield the desired effect for categorical variables. The overall model adjustments led to an improved performance when examining the averages values, particularly in terms of predictive abilities, where the updated version outperformed the original version for 10 out of 15 data sets. Again, when evaluating the result significant on a 95% confidence level, the same conclusion can not be stated because of the variability and thus wide, overlapping confidence intervals.

In conclusion, mainly MissForest and kNN imputation emerged as the top performers in this study. Both MissForest and kNN imputation were considerably better than GAIN in terms of imputation accuracy, yet all three models showed a similar performance with regards to prediction abilities. Considering practicality, kNN imputation proved to be the best model due to both having a low execution time as well as minimal tuning requirements, making it the most suitable choice among the evaluated imputation methods for real-world applications.

Despite the outcomes of this research, there remains potential for the GAIN model to prove its superiority as an imputation method. It is worth noting that the ideal network structure and optimal selection of hyperparameters were likely not identified in this study for the given data sets. Further investigation and refinement are necessary to fully explore the capabilities of GAIN in data imputation.

6.2 Impact of Additional Training Data

Secondly, this study aimed to investigate the impact of adding additional training data to state-of-art data imputation methods generated by a GAN network, namely CTGAN. Likely because of its inability to fully capture the underlying data structure, the additional training data did not

improve the direct imputation nor prediction performance for the majority of experiments. The exceptions for this was MissForest which was limited impacted by the additional data as well as kNN imputation which was improved by the additional data, specially in terms of predictive performance.

Evaluating the overall best performers in terms of direct imputation performance measured by mRMSE in this study, kNN imputation and MissForest, both without additional training data, emerged as the best performers. When examining predictive performance, kNN imputation with or without additional data achieved the best results for the majority of data sets. The additional training data strengthens kNN imputation's position as a optimal choice when selecting imputation method.

To sum up, the additional training data significantly enhanced the predictive performance of kNN imputation, which already without the additional data was one of the two top performing methods. The result of this study accordingly demonstrates that combining kNN imputation with the approach of increasing the training data using CTGAN can be a preferable choice for practical applications where the aim is to maximize the predictive performance of a data set with missing values.

6.3 Future Work

The limitations imposed in this study, as well as the selection of certain factors leaves room for further research. This includes exploring the same research questions yet for MNAR and MAR mechanisms for missing values, higher or lower amounts of missing values, different ratios of test and training data split and higher amounts of additional training data.

In this study, CTGAN was used with the default parameters while the impact of other GAN based synthezisers such as CTAB-GAN+ [78] and TGAN [71] warrants further investigation. The statistical similarity between the generated data and the original data was not analyzed, indicating the need for continued research. There remains a possibility that if the data generated by CTGAN exhibits closer statistical properties and successfully preserves the original data distribution, numerous data imputation models could derive benefits from additional training data. Larger data sets are also considered to influence the CTGAN model positively and demands further study.

Work that has shown to improve the GAIN model, such as SGAIN, WSGAIN-CP, and WSGAIN-GP [47] or GAIN with variable-splitting [7] was not used in this study and invites further analysis.

References

[1] Peshawa Jamal Muhammad Ali et al. "Data normalization and standardization: a technical report". In: *Mach Learn Tech Rep* 1.1 (2014), pp. 1–6.

[2] Martin Arjovsky and Leon Bottou. "Towards Principled Methods for Training Generative Adversarial Networks". In: *International Conference on Learning Representations*. 2017. URL: https://openreview.net/forum?id=Hk4_qw5xe.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. DOI: 10.48550/ARXIV.1701.07875. URL: https://arxiv.org/abs/1701.07875.

[4] Melissa J. Azur et al. "Multiple imputation by chained equations: what is it and how does it work?" In: *International Journal of Methods in Psychiatric Research* 20.1 (2011), pp. 40–49. DOI: https://doi.org/10.1002/mpr.329. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/mpr.329. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/mpr.329.

[5] Felix Biessmann et al. "DataWig: Missing Value Imputation for Tables." In: *J. Mach. Learn. Res.* 20.175 (2019), pp. 1–6.

[6] Ramiro Camino, Christian Hammerschmidt, and Radu State. "Generating Multi-Categorical Samples with Generative Adversarial Networks". In: (2018). DOI: 10.48550/ARXIV.1807.01202. URL: https://arxiv.org/abs/1807.01202.

[7] Ramiro D Camino, Christian A Hammerschmidt, and Radu State. "Improving missing data imputation with deep generative models". In: *arXiv preprint arXiv:1902.10666* (2019).

[8] Villani Cédric. *Optimal Transport. Old and New*. 2009. DOI: https://doi.org/10.1007/978-3-540-71050-9. URL: https://link.springer.com/book/10.1007/978-3-540-71050-9.

[9] Peter Cummings. "Missing data and multiple imputation". In: *JAMA pediatrics* 167.7 (2013), pp. 656–661.

[10] Mwamba Kasongo Dahouda and Inwhee Joe. "A deep-learned embedding technique for categorical features encoding". In: *IEEE Access* 9 (2021), pp. 114381–114391.

[11] Hari Prasanna Das and Costas J Spanos. "Improved dequantization and normalization methods for tabular data pre-processing in smart buildings". In: *Proceedings of the 9th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. 2022, pp. 168–177.

[12] Weinan Dong et al. "Generative adversarial networks for imputing missing data for big data clinical research". In: *BMC medical research methodology* 21 (2021), pp. 1–10.

[13] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[14] Richard O Duda, Peter E Hart, et al. *Pattern classification and scene analysis*. Vol. 3. Wiley New York, 1973.

[15] Carlos A. Escobar et al. *Learning with Missing Data*. 2020. DOI: 10.1109/BigData50022.2020.9377785.

[16] Shahla Faisal and Gerhard Tutz. "Nearest neighbor imputation for categorical data by weighting of attributes". In: *Information Sciences* 592 (2022), pp. 306–319.

[17] Magda Friedjungová et al. "Missing features reconstruction using a wasserstein generative adversarial imputation network". In: *Computational Science–ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part IV*. Springer. 2020, pp. 225–239.

[18] Pedro J Garcıa-Laencina, José-Luis Sancho-Gómez, and Anıbal R Figueiras-Vidal. "Pattern classification with missing data: a review". In: *Neural Computing and Applications* 19 (2010), pp. 263–282.

[19] Matt W Gardner and SR Dorling. "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences". In: *Atmospheric environment* 32.14-15 (1998), pp. 2627–2636.

[20] Lovedeep Gondara and Ke Wang. "Multiple imputation using deep denoising autoencoders". In: *arXiv preprint arXiv:1705.02737* 280 (2017).

[21] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. DOI: 10.48550/ARXIV.1406.2661. URL: https://arxiv.org/abs/1406.2661.

[22] Ye Gu et al. "A layered KNN-SVM approach to predict missing values of functional requirements in product customization". In: *Applied Sciences* 11.5 (2021), p. 2420.

[23] Ishaan Gulrajani et al. *Improved Training of Wasserstein GANs*. 2017. DOI: 10.48550/ARXIV.1704.00028. URL: https://arxiv.org/abs/1704.00028.

[24] Graeme Hawthorne, Graeme Hawthorne, and Peter Elliott. "Imputing cross-sectional missing data: Comparison of common techniques". In: *Australian and New Zealand Journal of Psychiatry* 39.7 (2005), pp. 583–590.

[25] Jianglin Huang et al. "Cross-validation based K nearest neighbor imputation for software quality datasets: an empirical study". In: *Journal of Systems and Software* 132 (2017), pp. 226–252.

[26] The MathWorks Inc. *Statistics and machine learning toolbox*. Natick, Massachusetts, United States, 2022. URL: https://www.mathworks.com/help/stats/index.html.

[27] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. DOI: 10.48550/ARXIV.1502.03167. URL: https://arxiv.org/abs/1502.03167.

[28] Sebastian Jäger, Arndt Allhorn, and Felix Bießmann. "A benchmark for data imputation methods". In: *Frontiers in big Data* 4 (2021), p. 693674.

[29] Eric Jang, Shixiang Gu, and Ben Poole. *Categorical Reparameterization with Gumbel-Softmax*. 2016. DOI: 10.48550/ARXIV.1611.01144. URL: https://arxiv.org/abs/1611.01144.

[30] Hyun Kang. "The prevention and handling of the missing data". In: *Korean journal of anesthesiology* 64.5 (2013), pp. 402–406.

[31] Tero Karras et al. *Training Generative Adversarial Networks with Limited Data*. 2020. arXiv: 2006.06676 [cs.CV].

[32] Wasif Khan et al. "Mixed Data Imputation Using Generative Adversarial Networks". In: *IEEE Access* 10 (2022), pp. 124475–124490.

[33] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[34] Arun Kumar, Matthias Boehm, and Jun Yang. "Data management in machine learning: Challenges, techniques, and systems". In: *Proceedings of the 2017 ACM International Conference on Management of Data*. 2017, pp. 1717–1722.

[35] Sang Kyu Kwak and Jong Hae Kim. "Statistical data preparation: management of missing values and outliers". In: *Korean journal of anesthesiology* 70.4 (2017), pp. 407–411.

[36] Florian Lalande and Kenji Doya. "Numerical data imputation: Choose kNN over deep learning". In: *Similarity Search and Applications: 15th International Conference, SISAP 2022, Bologna, Italy, October 5–7, 2022, Proceedings*. Springer. 2022, pp. 3–10.

[37] Wei-Chao Lin and Chih-Fong Tsai. "Missing value imputation: a review and analysis of the literature (2006–2017)". In: *Artificial Intelligence Review* 53 (2020), pp. 1487–1509.

[38] Zinan Lin et al. "Pacgan: The power of two samples in generative adversarial networks". In: *Advances in neural information processing systems* 31 (2018).

[39] Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*. Vol. 793. John Wiley & Sons, 2019.

[40] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. *The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables*. 2016. DOI: 10.48550/ARXIV.1611.00712. URL: https://arxiv.org/abs/1611.00712.

[41] Markus Maier, Matthias Hein, and Ulrike Von Luxburg. "Optimal construction of k-nearest-neighbor graphs for identifying noisy clusters". In: *Theoretical Computer Science* 410.19 (2009), pp. 1749–1764.

[42] R Malarvizhi and Antony Selvadoss Thanamani. "K-nearest neighbor in missing data imputation". In: *Int. J. Eng. Res. Dev* 5.1 (2012), pp. 5–7.

[43] Phayung Meesad and Kairung Hengpraprohm. "Combination of knn-based feature selection and knnbased missing-value imputation of microarray data". In: *2008 3rd International Conference on Innovative Computing Information and Control*. IEEE. 2008, pp. 341–341.

[44] Celso M de Melo et al. “Next-generation deep learning based on simulators and synthetic data”. In: *Trends in cognitive sciences* (2021).

[45] Maritza Mera-Gaona et al. “Evaluating the impact of multivariate imputation by MICE in feature selection”. In: *Plos one* 16.7 (2021), e0254720.

[46] “Missing value imputation affects the performance of machine learning: A review and analysis of the literature (2010–2021)”. In: *Informatics in Medicine Unlocked* 27 (2021), p. 100799. ISSN: 2352-9148. DOI: https://doi.org/10.1016/j.imu.2021.100799. URL: https://www.sciencedirect.com/science/article/pii/S2352914821002653.

[47] Diogo Telmo Neves, Marcel Ganesh Naik, and Alberto Proença. “SGAIN, WSGAIN-CP and WSGAIN-GP: Novel GAN methods for missing data imputation”. In: *Computational Science–ICCS 2021: 21st International Conference, Krakow, Poland, June 16–18, 2021, Proceedings, Part I*. Springer. 2021, pp. 98–113.

[48] Sergey I. Nikolenko. “The Early Days of Synthetic Data”. In: *Synthetic Data for Deep Learning*. Cham: Springer International Publishing, 2021, pp. 139–159. ISBN: 978-3-030-75178-4. DOI: 10.1007/978-3-030-75178-4_5. URL: https://doi.org/10.1007/978-3-030-75178-4_5.

[49] Fabian Pedregosa et al. *Scikit-learn: Machine Learning in Python*. 2018. arXiv: 1201.0490 [cs.LG].

[50] Adriana Perez et al. “Use of the mean, hot deck and multiple imputation techniques to predict outcome in intensive care unit patients in Colombia”. In: *Statistics in medicine* 21.24 (2002), pp. 3885–3896.

[51] Jason Poulos and Rafael Valle. “Missing data imputation for supervised learning”. In: *Applied Artificial Intelligence* 32.2 (2018), pp. 186–196.

[52] Archana Purwar and Sandeep Kumar Singh. “Hybrid prediction model with missing value imputation for medical data”. In: *Expert Systems with Applications* 42.13 (2015), pp. 5621–5631. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2015.02.050. URL: https://www.sciencedirect.com/science/article/pii/S0957417415001578.

[53] Thomas C Redman. “Improve data quality for competitive advantage”. In: *MIT Sloan Management Review* (1995).

[54] DONALD B. RUBIN. “Inference and missing data”. In: *Biometrika* 63.3 (Dec. 1976), pp. 581–592. ISSN: 0006-3444. DOI: 10.1093/biomet/63.3.581. eprint: https://academic.oup.com/biomet/article-pdf/63/3/581/756166/63-3-581.pdf. URL: https://doi.org/10.1093/biomet/63.3.581.

[55] Joseph L Schafer. “Multiple imputation: a primer”. In: *Statistical methods in medical research* 8.1 (1999), pp. 3–15.

[56] Jadran Sessa and Dabeeruddin Syed. “Techniques to deal with missing data”. In: *2016 5th international conference on electronic devices, systems and applications (ICEDSA)*. IEEE. 2016, pp. 1–4.

[57] SPSS. *Missing data: the hidden problem*. 2004.

[58] Akash Srivastava et al. *VEEGAN: Reducing Mode Collapse in GANs using Implicit Variational Learning*. 2017. DOI: 10.48550/ARXIV.1705.07761. URL: https://arxiv.org/abs/1705.07761.

[59] Daniel J Stekhoven and Peter Bühlmann. "MissForest—non-parametric missing value imputation for mixed-type data". In: *Bioinformatics* 28.1 (2012), pp. 112–118.

[60] Julia Stoyanovich, Bill Howe, and H.V. Jagadish. "Responsible Data Management". In: *Proceedings of the VLDB Endowment* 13.12 (). DOI: 10.14778/3415478.3415570. URL: https://par.nsf.gov/biblio/10184625.

[61] Tressy Thomas and Enayat Rajabi. "A systematic review of machine learning-based missing value imputation techniques". In: *Data Technologies and Applications* 55.4 (2021), pp. 558–585.

[62] Ngoc-Trung Tran et al. *Towards Good Practices for Data Augmentation in GAN Training*. June 2020.

[63] Olga Troyanskaya et al. "Missing value estimation methods for DNA microarrays". In: *Bioinformatics* 17.6 (2001), pp. 520–525.

[64] Gerhard Tutz and Shahla Ramzan. "Improved methods for the imputation of missing data by nearest neighbor methods". In: *Computational Statistics & Data Analysis* 90 (2015), pp. 84–99.

[65] Stef Van Buuren and Karin Groothuis-Oudshoorn. "mice: Multivariate imputation by chained equations in R". In: *Journal of statistical software* 45 (2011), pp. 1–67.

[66] Akbar K Waljee et al. "Comparison of imputation methods for missing laboratory data in medicine". In: *BMJ open* 3.8 (2013), e002847.

[67] Marc K Walton. "Addressing and advancing the problem of missing data". In: *Journal of Biopharmaceutical Statistics* 19.6 (2009), pp. 945–956.

[68] J Sophia Wang and Peter M Aronow. "Listwise Deletion in High Dimensions". In: *Political Analysis* (2023), pp. 1–7.

[69] Yufeng Wang et al. "PC-GAIN: Pseudo-label conditional generative adversarial imputation networks for incomplete data". In: *Neural Networks* 141 (2021), pp. 395–403.

[70] Lilian Weng. "From gan to wgan". In: *arXiv preprint arXiv:1904.08994* (2019).

[71] Lei Xu and Kalyan Veeramachaneni. *Synthesizing Tabular Data using Generative Adversarial Networks*. 2018. DOI: 10.48550/ARXIV.1811.11264. URL: https://arxiv.org/abs/1811.11264.

[72] Lei Xu et al. "Modeling tabular data using conditional GAN". In: *Advances in Neural Information Processing Systems* 32 (2019).

[73] Ke Yang et al. "Fairness-Aware Instrumentation of Preprocessing˜ Pipelines for Machine Learning". In: *Workshop on Human-In-the-Loop Data Analytics (HILDA'20)*. 2020.

[74] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. *GAIN: Missing Data Imputation using Generative Adversarial Nets*. 2018. DOI: 10.48550/ARXIV.1806.02920. URL: https://arxiv.org/abs/1806.02920.

[75] Shichao Zhang et al. "A novel kNN algorithm with data-driven k parameter computation". In: *Pattern Recognition Letters* 109 (2018), pp. 44–54.

[76] Zhongheng Zhang. "Missing data imputation: focusing on single imputation". In: *Annals of translational medicine* 4.1 (2016).

[77] Shengyu Zhao et al. *Differentiable Augmentation for Data-Efficient GAN Training*. 2020. arXiv: 2006.10738 [cs.CV].

[78] Zilong Zhao et al. *CTAB-GAN: Effective Table Data Synthesizing*. 2021. arXiv: 2102.08369 [cs.LG].

[79] Xu Zhou et al. "Federated conditional generative adversarial nets imputation method for air quality missing data". In: *Knowledge-Based Systems* 228 (2021), p. 107261.

Appendix

Direct imputation evaluation

Data set	Miss %	Additional CTGAN data%	Median/Mode					MICE				
			mRMSE	RMSE num	RMSE cat	PFC (%)	Time (s)	mRMSE	RMSE num	RMSE cat	PFC (%)	Time (s)
Mushroom	10	0	0.3955	-	0.3955	15.6410	0.0088	0.2992	-	0.2992	8.9540	16,623.0000
		50	0.4026	-	0.4026	16.2103	0.0129	0.3185	-	0.3185	10.1444	30,812.9900
		100	0.4035	-	0.4035	16.2828	0.0191	0.3354	-	0.3354	11.2520	35,016.3100
	30	0	0.3964	-	0.3964	15.7140	0.0088	0.3071	-	0.3071	9.4312	13,624.0000
		50	0.4083	-	0.4083	16.6742	0.0117	0.2919	-	0.2919	8.5197	21,595.5600
		100	0.4083	-	0.4083	16.6742	0.0255	0.2864	-	0.2864	8.1997	50,137.0800
	50	0	0.3978	-	0.3978	15.8236	0.0098	0.3194	-	0.3194	10.2035	9,907.0000
		50	-	-	-	-	-	-	-	-	-	-
		100	-	-	-	-	-	-	-	-	-	-
Letter	10	0	0.1555	0.1555	-	-	0.0231	0.1522	0.1522	-	-	798.0000
		50	0.1569	0.1569	-	-	0.0245	0.1776	0.1776	-	-	1,247.5640
		100	0.1569	0.1569	-	-	0.0331	0.1831	0.1831	-	-	1,769.7750
	30	0	0.1551	0.1551	-	-	0.0192	0.1614	0.1614	-	-	634.0000
		50	0.1604	0.1604	-	-	0.0276	0.2054	0.2054	-	-	1,304.5640
		100	0.1717	0.1717	-	-	0.0325	0.2179	0.2179	-	-	7,200.4490
	50	0	0.1561	0.1561	-	-	0.0201	0.1736	0.1736	-	-	469.0000
		50	-	-	-	-	-	-	-	-	-	-
		100	-	-	-	-	-	-	-	-	-	-
Bank	10	0	0.4957	0.2661	0.4182	17.4882	0.0441	0.4280	0.1585	0.3976	15.8082	27,945.0000
		50	0.5012	0.2663	0.4245	18.0236	0.0705	0.4583	0.1926	0.4159	17.2990	55,749.5100
		100	0.4998	0.2662	0.4230	17.8898	0.0842	0.4738	0.2041	0.4276	18.2821	70,437.8300
	30	0	0.4942	0.2636	0.4180	17.4764	0.0489	0.4496	0.1823	0.4110	16.8900	19,092.0000
		50	0.4927	0.2556	0.4212	17.7406	0.0650	0.4878	0.2212	0.4348	18.9012	31,385.9900
		100	0.4954	0.2609	0.4212	17.7406	0.0916	0.4982	0.2320	0.4409	19.4418	56,398.3800
	50	0	0.4959	0.2671	0.4179	17.4599	0.0477	0.4752	0.2148	0.4239	17.9660	13,067.0000
		50	-	-	-	-	-	-	-	-	-	-
		100	-	-	-	-	-	-	-	-	-	-
Credit	10	0	0.3412	0.0711	0.3338	11.1390	0.0482	0.2694	0.0667	0.2610	6.8145	53,815.9300
		50	0.3410	0.0728	0.3332	11.1004	0.0717	0.2935	0.0773	0.2831	8.0146	62,607.1400
		100	0.3413	0.0742	0.3332	11.1004	0.0957	0.3017	0.0812	0.2906	8.4432	97,838.2700
	30	0	0.3366	0.0707	0.3291	10.8315	0.0479	0.2852	0.0709	0.2762	7.6288	37,228.0000
		50	0.3366	0.0704	0.3291	10.8315	0.0650	0.3063	0.0700	0.2982	8.8926	73,280.6600
		100	0.3367	0.0711	0.3291	10.8315	0.0874	0.3148	0.0692	0.3071	9.4295	78,805.9900
	50	0	0.3376	0.0711	0.3301	10.8933	0.0447	0.3048	0.0744	0.2956	8.7387	46,321.3000
		50	-	-	-	-	-	-	-	-	-	-
		100	-	-	-	-	-	-	-	-	-	-
News	10	0	0.5046	0.1942	0.4658	21.6931	0.1543	0.3975	0.1412	0.3716	13.8075	121,295.4000
		50	0.5085	0.1953	0.4696	22.0487	0.2228	0.4567	0.1874	0.4165	17.3483	167,948.8000
		100	0.5098	0.1985	0.4696	22.0487	0.3152	0.4741	0.2041	0.4279	18.3069	389,598.1000
	30	0	0.5046	0.1954	0.4652	21.6391	0.1478	0.4273	0.1606	0.3960	15.6822	106,131.0300
		50	-	-	-	-	-	-	-	-	-	-
		100	-	-	-	-	-	-	-	-	-	-
	50	0	0.5041	0.1954	0.4646	21.5900	0.1457	0.4561	0.1840	0.4173	17.4153	95,325.6500
		50	-	-	-	-	-	-	-	-	-	-
		100	-	-	-	-	-	-	-	-	-	-

Table 5: Direct imputation performance for Median/Mode and MICE.

Data set	Miss %	Additional CTGAN data%	kNN imputation					MissForest				
			mRMSE	RMSE num	RMSE cat	PFC (%)	Time (s)	mRMSE	RMSE num	RMSE cat	PFC (%)	Time (s)
Mushroom	10	0	0.2938	-	0.2938	8.6331	13.0536	0.2738	-	0.2738	7.4981	234.6251
		50	0.2973	-	0.2973	8.8401	23.1802	0.2752	-	0.2752	7.5721	190.4786
		100	0.2991	-	0.2991	8.9436	28.3888	0.2755	-	0.2755	7.5881	195.3887
	30	0	0.2857	-	0.2857	8.1649	33.8431	0.2946	-	0.2946	8.6782	226.0323
		50	0.2855	-	0.2855	8.1510	50.7801	0.2907	-	0.2907	8.4540	167.5020
		100	0.2854	-	0.2854	8.1475	86.1779	0.2906	-	0.2906	8.4429	197.2753
	50	0	0.3122	-	0.3122	9.7443	55.9795	0.3084	-	0.3084	9.5111	219.9948
		50	-	-	-	-	-	-	-	-	-	-
		100	-	-	-	-	-	-	-	-	-	-
Letter	10	0	0.0563	0.0563	-	-	20.0696	0.0668	0.0668	-	-	150.4569
		50	0.0564	0.0564	-	-	30.0753	0.0669	0.0669	-	-	170.4551
		100	0.0568	0.0568	-	-	39.2864	0.0670	0.0670	-	-	239.2756
	30	0	0.0949	0.0949	-	-	41.0851	0.0863	0.0863	-	-	181.5598
		50	0.0948	0.0948	-	-	65.8943	0.0874	0.0874	-	-	222.8474
		100	0.0949	0.0949	-	-	97.6715	0.0874	0.0874	-	-	309.1806
	50	0	0.1421	0.1421	-	-	43.3355	0.1117	0.1117	-	-	181.7431
		50	-	-	-	-	-	-	-	-	-	-
		100	-	-	-	-	-	-	-	-	-	-
Bank	10	0	0.3891	0.1201	0.3701	13.6989	242.7307	0.3601	0.0790	0.3514	12.3461	373.9059
		50	0.3900	0.1204	0.3709	13.7589	429.3248	0.3623	0.0787	0.3536	12.5053	667.0185
		100	0.3906	0.1215	0.3712	13.7773	680.5893	0.3627	0.0786	0.3541	12.5362	977.0187
	30	0	0.3905	0.1309	0.3679	13.5349	554.8403	0.3739	0.0915	0.3625	13.1404	367.9164
		50	0.3897	0.1302	0.3673	13.4922	755.7339	0.3714	0.0912	0.3600	12.9625	519.8957
		100	0.3896	0.1303	0.3672	13.4845	1,090.0957	0.3735	0.0909	0.3623	13.1256	666.6593
	50	0	0.4389	0.1914	0.3950	15.5999	752.8814	0.4074	0.1255	0.3876	15.0221	343.9106
		50	-	-	-	-	-	-	-	-	-	-
		100	-	-	-	-	-	-	-	-	-	-
Credit	10	0	0.2488	0.0525	0.2432	5.9145	212.3275	0.2623	0.0481	0.2578	6.6485	876.7438
		50	0.2490	0.0531	0.2433	5.9188	371.5996	0.2599	0.0477	0.2555	6.5265	1,244.0736
		100	0.2488	0.0534	0.2430	5.9059	561.0505	0.2587	0.0477	0.2543	6.4657	3,369.6949
	30	0	0.2631	0.0548	0.2573	6.6212	412.7396	0.2731	0.0517	0.2682	7.1931	726.0621
		50	0.2631	0.0548	0.2574	6.6241	615.7652	0.2827	0.0516	0.2780	7.7285	1,093.8533
		100	0.2633	0.0548	0.2576	6.6342	999.9039	0.2829	0.0517	0.2782	7.7378	1,483.2502
	50	0	0.2805	0.0628	0.2734	7.4761	426.9053	0.2895	0.0567	0.2839	8.0616	751.1649
		50	-	-	-	-	-	-	-	-	-	-
		100	-	-	-	-	-	-	-	-	-	-
News	10	0	0.3824	0.1138	0.3651	13.3282	200.3602	0.3807	0.0667	0.3748	14.0472	4,563.4597
		50	0.3830	0.1139	0.3657	13.3746	510.4984	0.3775	0.0668	0.3716	13.8060	17,908.8383
		100	0.3827	0.1140	0.3653	13.3436	977.3805	0.3778	0.0671	0.3718	13.8253	19,145.3693
	30	0	0.4226	0.1319	0.4015	16.1178	815.9997	0.3978	0.0936	0.3866	14.9487	3,966.0575
		50	-	-	-	-	-	-	-	-	-	-
		100	-	-	-	-	-	-	-	-	-	-
	50	0	0.4671	0.1539	0.4410	19.4521	1,082.1920	0.4082	0.1228	0.3892	15.1518	2,781.4976
		50	-	-	-	-	-	-	-	-	-	-
		100	-	-	-	-	-	-	-	-	-	-

Table 6: Direct imputation performance for kNN imputation and MissForest.

Data set	Miss %	Additional CTGAN data%	GAIN v1					GAIN v2				
			mRMSE	RMSE num	RMSE cat	PFC (%)	Time (s)	mRMSE	RMSE num	RMSE cat	PFC (%)	Time (s)
Mushroom	10	0	0.3012	-	0.3012	9.0751	219.3784	0.3079	-	0.3079	9.4840	86.9120
		50	0.3098	-	0.3098	9.6035	220.0752	0.3532	-	0.3532	12.4776	87.3286
		100	0.3147	-	0.3147	9.9063	220.2326	0.3531	-	0.3531	12.4724	87.7730
		200	0.3169	-	0.3169	10.0461	202.9568	0.3682	-	0.3682	13.5687	94.6598
		500	0.3307	-	0.3307	10.9378	273.9795	0.4168	-	0.4168	17.3811	89.4766
	30	0	0.3106	-	0.3106	9.6935	491.4529	0.2986	-	0.2986	8.9153	133.0322
		50	0.3004	-	0.3004	9.0233	320.3635	0.3001	-	0.3001	9.0064	133.6565
		100	0.3029	-	0.3029	9.1750	313.3883	0.3028	-	0.3028	9.1675	133.7602
		200	0.3074	-	0.3074	9.4500	292.4889	0.3041	-	0.3041	9.2513	133.6157
		500	0.3120	-	0.3120	9.7363	295.5243	0.3116	-	0.3116	9.7095	134.7257
	50	0	0.3023	-	0.3023	9.1395	289.4416	0.3110	-	0.3110	9.6780	133.8178
		50	-	-	-	-	-	-	-	-	-	-
		100	-	-	-	-	-	-	-	-	-	-
		200	-	-	-	-	-	-	-	-	-	-
		500	-	-	-	-	-	-	-	-	-	-
Letter	10	0	0.1307	0.1307	-	-	80.7946	0.1489	0.1489	-	-	32.8274
		50	0.1441	0.1441	-	-	82.5710	0.1552	0.1552	-	-	33.5720
		100	0.1447	0.1447	-	-	83.0033	0.1619	0.1619	-	-	28.5619
		200	0.1509	0.1509	-	-	86.3958	0.1704	0.1704	-	-	33.3429
		500	0.1565	0.1565	-	-	94.0636	0.1926	0.1926	-	-	33.4046
	30	0	0.1410	0.1410	-	-	21.2775	0.1381	0.1381	-	-	7.0703
		50	0.2241	0.2241	-	-	22.6122	0.2266	0.2266	-	-	7.4875
		100	0.2421	0.2421	-	-	23.4082	0.2532	0.2532	-	-	7.7362
		200	0.2681	0.2681	-	-	26.1355	0.2557	0.2557	-	-	8.6405
		500	0.2691	0.2691	-	-	262.0605	0.2622	0.2622	-	-	10.7681
	50	0	0.1535	0.1535	-	-	21.3343	0.1395	0.1395	-	-	6.8924
		50	-	-	-	-	-	-	-	-	-	-
		100	-	-	-	-	-	-	-	-	-	-
		200	-	-	-	-	-	-	-	-	-	-
		500	-	-	-	-	-	-	-	-	-	-
Bank	10	0	0.4571	0.1991	0.4112	16.9196	102.6840	0.4540	0.1679	0.4218	17.7947	42.2187
		50	0.5078	0.2508	0.4411	19.4676	105.4820	0.5029	0.2356	0.4440	19.7166	42.6537
		100	0.5263	0.2757	0.4478	20.0674	107.2780	0.5288	0.2609	0.4598	21.1567	43.0019
		200	0.5255	0.2734	0.4484	20.1154	113.6608	0.5267	0.2548	0.4607	21.2282	43.6108
		500	0.5389	0.2806	0.4599	21.1636	132.0192	0.5529	0.3075	0.4593	21.1036	48.4040
	30	0	0.5023	0.2606	0.4289	18.4549	126.0892	0.4947	0.2301	0.4378	19.1702	54.2925
		50	0.5309	0.2786	0.4515	20.3978	128.1379	0.5348	0.2655	0.4642	21.5671	52.9229
		100	0.5374	0.2845	0.4556	20.7813	130.6764	0.5466	0.2746	0.4726	22.3444	54.3152
		200	0.5296	0.2899	0.4429	19.6227	136.5815	0.5516	0.2769	0.4769	22.7539	55.0712
		500	0.5379	0.2926	0.4511	20.3602	153.5657	0.5647	0.2919	0.4834	23.3711	59.6233
	50	0	0.4960	0.2452	0.4306	18.5818	126.2219	0.5345	0.2847	0.4509	20.3589	52.2066
		50	-	-	-	-	-	-	-	-	-	-
		100	-	-	-	-	-	-	-	-	-	-
		200	-	-	-	-	-	-	-	-	-	-
		500	-	-	-	-	-	-	-	-	-	-
Credit	10	0	0.2865	0.0846	0.2723	7.4319	123.9849	0.2845	0.1039	0.2641	6.9782	47.9572
		50	0.3108	0.1144	0.2889	8.3512	125.6246	0.3051	0.1329	0.2744	7.5363	48.5927
		100	0.3171	0.1239	0.2919	8.5269	128.5328	0.3182	0.1428	0.2840	8.0699	49.4474
		200	0.3303	0.1260	0.3052	9.3200	132.4346	0.3202	0.1422	0.2866	8.2139	50.3687
		500	0.3397	0.1411	0.3089	9.5521	144.5168	0.3660	0.1492	0.3340	11.1587	53.6295
	30	0	0.3028	0.1310	0.2712	7.3627	123.9755	0.3143	0.1298	0.2841	8.0744	47.7763
		50	0.2955	0.0851	0.2829	8.0099	126.5915	0.3301	0.0956	0.3158	9.9887	48.3202
		100	0.2977	0.0892	0.2840	8.0690	128.0890	0.3403	0.1089	0.3223	10.3919	49.1599
		200	0.3100	0.0956	0.2943	8.6716	132.1302	0.3553	0.0819	0.3457	11.9597	50.2976
		500	0.3249	0.0870	0.3130	9.8123	144.0905	0.3767	0.0908	0.3656	13.3710	53.6232
	50	0	0.3658	0.1897	0.3084	9.6314	124.5231	0.3477	0.1960	0.2859	8.1818	49.4874
		50	-	-	-	-	-	-	-	-	-	-
		100	-	-	-	-	-	-	-	-	-	-
		200	-	-	-	-	-	-	-	-	-	-
		500	-	-	-	-	-	-	-	-	-	-
News	10	0	0.5286	0.2028	0.4877	24.0928	156.0280	0.5080	0.2025	0.4659	21.7286	60.4982
		50	0.5528	0.2261	0.5040	25.6637	159.0499	0.5337	0.2478	0.4727	22.3471	58.9748
		100	0.5666	0.2290	0.5172	27.5346	170.0710	0.5427	0.2632	0.4745	22.5234	58.6028
		200	0.5230	0.2287	0.4699	22.3363	188.8049	0.5631	0.2803	0.4883	23.8578	60.7719
		500	0.5654	0.2387	0.5121	26.4237	204.8834	0.5646	0.3019	0.4769	22.7522	65.3983
	30	0	0.5818	0.3011	0.4963	24.9993	127.1362	0.5231	0.2769	0.4426	19.5948	43.1571
		50	-	-	-	-	-	-	-	-	-	-
		100	-	-	-	-	-	-	-	-	-	-
		200	-	-	-	-	-	-	-	-	-	-
		500	-	-	-	-	-	-	-	-	-	-
	50	0	0.5090	0.2006	0.4676	21.9779	167.8975	0.5206	0.2352	0.4639	21.5315	60.5239
		50	-	-	-	-	-	-	-	-	-	-
		100	-	-	-	-	-	-	-	-	-	-
		200	-	-	-	-	-	-	-	-	-	-
		500	-	-	-	-	-	-	-	-	-	-

Table 7: Direct imputation performance for GAIN v1 and GAIN v2.

Prediction evaluation

Data set	Miss %	Additional CTGAN data%	Median/Mode			MICE		
			Accuracy	AUROC	MSE	Accuracy	AUROC	MSE
Mushroom	10	0	0.9938	0.9938	-	0.9969	0.9968	-
		50	0.9938	0.9938	-	0.9938	0.9942	-
		100	0.9938	0.9938	-	0.9938	0.9935	-
	30	0	0.9723	0.9721	-	0.9908	0.9906	-
		50	0.9631	0.9620	-	0.9938	0.9938	-
		100	0.9631	0.9620	-	0.9846	0.9841	-
	50	0	0.9231	0.9211	-	0.9723	0.9717	-
		50	-	-	-	-	-	-
		100	-	-	-	-	-	-
Letter	10	0	0.7438	0.8666	-	0.7613	0.8777	-
		50	0.7413	0.8649	-	0.7025	0.8462	-
		100	0.7413	0.8649	-	0.7175	0.8547	-
	30	0	0.5075	0.7431	-	0.4725	0.7265	-
		50	0.4975	0.7379	-	0.4113	0.6981	-
		100	0.4813	0.7301	-	0.3963	0.6884	-
	50	0	0.3413	0.6585	-	0.2875	0.6327	-
		50	-	-	-	-	-	-
		100	-	-	-	-	-	-
Bank	10	0	0.8841	0.5800	-	0.8835	0.5859	-
		50	0.8841	0.5800	-	0.8841	0.5883	-
		100	0.8865	0.5959	-	0.8865	0.5897	-
	30	0	0.8829	0.5773	-	0.8835	0.5817	-
		50	0.8805	0.5717	-	0.8817	0.5683	-
		100	0.8805	0.5697	-	0.8811	0.5679	-
	50	0	0.8774	0.5555	-	0.8726	0.5258	-
		50	-	-	-	-	-	-
		100	-	-	-	-	-	-
Credit	10	0	0.8133	0.6224	-	0.8142	0.6244	-
		50	0.8108	0.6237	-	0.8050	0.5984	-
		100	0.8108	0.6194	-	0.8083	0.6221	-
	30	0	0.8042	0.5921	-	0.8017	0.5992	-
		50	0.8050	0.5955	-	0.8017	0.5920	-
		100	0.8042	0.5936	-	0.8108	0.6064	-
	50	0	0.7933	0.5824	-	0.7967	0.5600	-
		50	-	-	-	-	-	-
		100	-	-	-	-	-	-
News	10	0	-	-	42,697,557.5181	-	-	42,780,641.5494
		50	-	-	42,658,124.6885	-	-	43,069,866.4412
		100	-	-	42,657,250.5675	-	-	43,905,882.2382
	30	0	-	-	45,732,512.0135	-	-	42,996,370.0172
		50	-	-	-	-	-	-
		100	-	-	-	-	-	-
	50	0	-	-	45,984,065.0999	-	-	43,543,375.0009
		50	-	-	-	-	-	-
		100	-	-	-	-	-	-

Table 8: Prediction performance for Median/Mode and MICE.

Data set	Miss %	Additional CTGAN data%	kNN imputation			MissForest		
			Accuracy	AUROC	MSE	Accuracy	AUROC	MSE
Mushroom	10	0	0.9969	0.9968	-	0.9969	0.9968	-
		50	0.9969	0.9968	-	0.9969	0.9968	-
		100	0.9969	0.9968	-	0.9969	0.9968	-
	30	0	0.9969	0.9968	-	0.9856	0.9850	-
		50	0.9969	0.9968	-	0.9877	0.9871	-
		100	0.9969	0.9968	-	0.9874	0.9869	-
	50	0	0.9631	0.9627	-	0.9579	0.9565	-
		50	-	-	-	-	-	-
		100	-	-	-	-	-	-
Letter	10	0	0.8638	0.9277	-	0.8465	0.9191	-
		50	0.8625	0.9272	-	0.8440	0.9175	-
		100	0.8625	0.9272	-	0.8440	0.9175	-
	30	0	0.6688	0.8264	-	0.6704	0.8288	-
		50	0.6763	0.8307	-	0.6674	0.8281	-
		100	0.6738	0.8289	-	0.6683	0.8281	-
	50	0	0.3888	0.6837	-	0.4764	0.7305	-
		50	-	-	-	-	-	-
		100	-	-	-	-	-	-
Bank	10	0	0.8817	0.5766	-	0.8838	0.5877	-
		50	0.8835	0.5859	-	0.8834	0.5876	-
		100	0.8817	0.5786	-	0.8840	0.5896	-
	30	0	0.8817	0.5724	-	0.8852	0.5853	-
		50	0.8841	0.5800	-	0.8863	0.5877	-
		100	0.8835	0.5817	-	0.8861	0.5880	-
	50	0	0.8768	0.5614	-	0.8759	0.5732	-
		50	-	-	-	-	-	-
		100	-	-	-	-	-	-
Credit	10	0	0.8133	0.6267	-	0.8029	0.5987	-
		50	0.8150	0.6422	-	0.8038	0.5970	-
		100	0.8117	0.6372	-	0.8026	0.5966	-
	30	0	0.8067	0.6052	-	0.7988	0.5903	-
		50	0.8042	0.6123	-	0.7993	0.5802	-
		100	0.8033	0.6060	-	0.7986	0.5808	-
	50	0	0.7992	0.5760	-	0.7947	0.5687	-
		50	-	-	-	-	-	-
		100	-	-	-	-	-	-
News	10	0	-	-	42,740,918.2638	-	-	43,093,960.8535
		50	-	-	42,740,744.7643	-	-	43,126,718.5982
		100	-	-	42,748,974.4959	-	-	43,168,362.2022
	30	0	-	-	45,586,818.5406	-	-	43,191,254.5138
		50	-	-	-	-	-	-
		100	-	-	-	-	-	-
	50	0	-	-	45,950,574.0475	-	-	47,293,929.6606
		50	-	-	-	-	-	-
		100	-	-	-	-	-	-

Table 9: Prediction performance for kNN imputation and MissForest.

Data set	Miss %	Additional CTGAN data%	GAIN v1			GAIN v2		
			Accuracy	AUROC	MSE	Accuracy	AUROC	MSE
Mushroom	10	0	0.9960	0.9958	-	0.9972	0.9971	-
		50	0.9960	0.9959	-	0.9908	0.9906	-
		100	0.9954	0.9953	-	0.9914	0.9913	-
		200	0.9945	0.9943	-	0.9914	0.9914	-
		500	0.9929	0.9929	-	0.9926	0.9924	-
	30	0	0.9905	0.9904	-	0.9920	0.9918	-
		50	0.9926	0.9925	-	0.9905	0.9904	-
		100	0.9880	0.9879	-	0.9877	0.9876	-
		200	0.9902	0.9899	-	0.9886	0.9885	-
		500	0.9880	0.9879	-	0.9862	0.9861	-
	50	0	0.9754	0.9752	-	0.9652	0.9645	-
		50	-	-	-	-	-	-
		100	-	-	-	-	-	-
		200	-	-	-	-	-	-
		500	-	-	-	-	-	-
Letter	10	0	0.7766	0.8832	-	0.7595	0.8749	-
		50	0.7630	0.8767	-	0.7521	0.8713	-
		100	0.7636	0.8766	-	0.7444	0.8676	-
		200	0.7595	0.8738	-	0.7355	0.8627	-
		500	0.7489	0.8683	-	0.7059	0.8474	-
	30	0	0.5366	0.7604	-	0.5433	0.7630	-
		50	0.4329	0.7058	-	0.4196	0.6995	-
		100	0.3994	0.6886	-	0.3778	0.6778	-
		200	0.3963	0.6866	-	0.3885	0.6821	-
		500	0.3815	0.6785	-	0.3780	0.6763	-
	50	0	0.3458	0.6615	-	0.3665	0.6722	-
		50	-	-	-	-	-	-
		100	-	-	-	-	-	-
		200	-	-	-	-	-	-
		500	-	-	-	-	-	-
Bank	10	0	0.8830	0.5819	-	0.8842	0.5887	-
		50	0.8839	0.5855	-	0.8837	0.5905	-
		100	0.8847	0.5872	-	0.8838	0.5858	-
		200	0.8839	0.5843	-	0.8839	0.5830	-
		500	0.8840	0.5864	-	0.8840	0.5810	-
	30	0	0.8829	0.5723	-	0.8824	0.5691	-
		50	0.8837	0.5736	-	0.8826	0.5719	-
		100	0.8828	0.5704	-	0.8837	0.5736	-
		200	0.8834	0.5718	-	0.8841	0.5657	-
		500	0.8833	0.5708	-	0.8834	0.5668	-
	50	0	0.8760	0.5500	-	0.8774	0.5511	-
		50	-	-	-	-	-	-
		100	-	-	-	-	-	-
		200	-	-	-	-	-	-
		500	-	-	-	-	-	-
Credit	10	0	0.8133	0.6204	-	0.8113	0.6198	-
		50	0.8120	0.6213	-	0.8136	0.6231	-
		100	0.8109	0.6176	-	0.8120	0.6220	-
		200	0.8103	0.6183	-	0.8127	0.6257	-
		500	0.8098	0.6186	-	0.8116	0.6203	-
	30	0	0.8009	0.6050	-	0.8047	0.6152	-
		50	0.8034	0.6128	-	0.8047	0.6113	-
		100	0.8020	0.6072	-	0.8043	0.6074	-
		200	0.8064	0.6056	-	0.8056	0.6119	-
		500	0.8051	0.6047	-	0.8018	0.5968	-
	50	0	0.8053	0.5926	-	0.8021	0.5886	-
		50	-	-	-	-	-	-
		100	-	-	-	-	-	-
		200	-	-	-	-	-	-
		500	-	-	-	-	-	-
News	10	0	-	-	43,792,912.8591	-	-	43,284,014.6813
		50	-	-	43,772,561.1972	-	-	43,579,980.1592
		100	-	-	43,775,467.8340	-	-	43,548,399.4015
		200	-	-	43,920,291.5668	-	-	44,443,664.0741
		500	-	-	43,927,704.3533	-	-	43,831,439.1618
	30	0	-	-	45,820,264.5270	-	-	45,370,414.5048
		50	-	-	-	-	-	-
		100	-	-	-	-	-	-
		200	-	-	-	-	-	-
		500	-	-	-	-	-	-
	50	0	-	-	46,159,032.7703	-	-	45,111,158.0332
		50	-	-	-	-	-	-
		100	-	-	-	-	-	-
		200	-	-	-	-	-	-
		500	-	-	-	-	-	-

Table 10: Prediction performance for GAIN v1 and GAIN v2.

Data set	Miss %	List-wise deletion		
		Accuracy	AUROC	MSE
Mushroom	10	0.4848	0.4908	-
	30	-	-	-
	50	-	-	-
Letter	10	0.0414	0.5039	-
	30	0.0000	0.5000	-
	50	-	-	-
Bank	10	0.9192	0.5000	-
	30	0.0000	0.0000	-
	50	-	-	-
Credit	10	0.8559	0.4948	-
	30	-	-	-
	50	-	-	-
News	10	-	-	45,904,746.6624
	30	-	-	-
	50	-	-	-

Table 11: Prediction performance for list-wise deletion.

Confidence Intervals

Data set	Miss%	mRMSE	RMSE num	RMSE cat	PFC (%)	Execution time (s)	Accuracy	AUROC	MSE
Mushroom	10	[0.2729, 0.2747]	-	[0.2729, 0.2747]	[7.45, 7.5462]	[223.7276, 245.5226]	[0.9969, 0.9969]	[0.9968, 0.9968]	-
	30	[0.2937, 0.2955]	-	[0.2937, 0.2955]	[8.6236, 8.7328]	[214.6165, 237.4481]	[0.983, 0.9882]	[0.9824, 0.9876]	-
	50	[0.3079, 0.3089]	-	[0.3079, 0.3089]	[9.4787, 9.5435]	[214.0553, 225.9343]	[0.9529, 0.9629]	[0.9514, 0.9616]	-
Letter	10	[0.0667, 0.0669]	[0.0667, 0.0669]	-	-	[147.3322, 153.5816]	[0.8442, 0.8488]	[0.918, 0.9202]	-
	30	[0.0861, 0.0865]	[0.0861, 0.0865]	-	-	[178.4375, 184.6821]	[0.6633, 0.6775]	[0.8252, 0.8324]	-
	50	[0.1115, 0.1119]	[0.1115, 0.1119]	-	-	[178.036, 185.4502]	[0.469, 0.4838]	[0.7271, 0.7339]	-
Bank	10	[0.3592, 0.361]	[0.0786, 0.0794]	[0.3505, 0.3523]	[12.2789, 12.4133]	[333.6991, 414.1127]	[0.8829, 0.8847]	[0.5845, 0.5909]	-
	30	[0.3727, 0.3751]	[0.0909, 0.0921]	[0.3614, 0.3636]	[13.0578, 13.223]	[320.3713, 367.4499]	[0.884, 0.8864]	[0.5789, 0.5917]	-
	50	[0.4065, 0.4083]	[0.1241, 0.1269]	[0.3867, 0.3885]	[15.0212, 15.023]	[341.0437, 394.7891]	[0.8747, 0.8771]	[0.5654, 0.581]	-
Credit	10	[0.2612, 0.2634]	[0.048, 0.0482]	[0.2567, 0.2589]	[6.593, 6.704]	[819.94, 933.5476]	[0.8015, 0.8043]	[0.5951, 0.6023]	-
	30	[0.2724, 0.2738]	[0.0516, 0.0518]	[0.2675, 0.2689]	[14.9098, 14.9876]	[662.3405, 789.7837]	[0.7957, 0.8019]	[0.5834, 0.5972]	-
	50	[0.2889, 0.2901]	[0.0565, 0.0569]	[0.2833, 0.2845]	[8.03, 8.0932]	[680.6784, 821.6514]	[0.7925, 0.7969]	[0.5651, 0.5723]	-
News	10	[0.379, 0.3824]	[0.0654, 0.068]	[0.3731, 0.3765]	[13.9186, 14.1758]	[4190.3856, 4936.5338]	-	-	[43059959.1082, 43127962.5918]
	30	[0.3969, 0.3987]	[0.0934, 0.0938]	[0.3857, 0.3875]	[14.88, 15.0174]	[3658.9087, 4273.2063]	-	-	[43043533.1283, 43338975.8917]
	50	[0.4069, 0.4095]	[0.1225, 0.1231]	[0.3878, 0.3906]	[15.0467, 15.2569]	[2542.1285, 3020.8667]	-	-	[46501018.1449, 48086841.1751]

Table 12: Confidence intervals for MissForest with 0% additional CTGAN data.

Data set	Miss%	mRMSE	RMSE num	RMSE cat	PFC (%)	Execution time (s)	Accuracy	AUROC	MSE
Mushroom	10	[0.2744, 0.276]	-	[0.2744, 0.276]	[7.5303, 7.6139]	[168.229, 212.7282]	[0.9969, 0.9969]	[0.9968, 0.9968]	-
	30	[0.2892, 0.2922]	-	[0.2892, 0.2922]	[8.3652, 8.5428]	[145.3195, 189.6845]	[0.9854, 0.99]	[0.9847, 0.9895]	-
	50	-	-	-	-	-	-	-	-
Letter	10	[0.0668, 0.067]	[0.0668, 0.067]	-	-	[153.8289, 187.0813]	[0.8425, 0.8455]	[0.9166, 0.9184]	-
	30	[0.0872, 0.0876]	[0.0872, 0.0876]	-	-	[209.5631, 236.1317]	[0.6635, 0.6713]	[0.8259, 0.8303]	-
	50	-	-	-	-	-	-	-	-
Bank	10	[0.3613, 0.3633]	[0.0783, 0.0791]	[0.3525, 0.3547]	[12.4304, 12.5802]	[587.0778, 746.9592]	[0.8825, 0.8843]	[0.5835, 0.5917]	-
	30	[0.3698, 0.373]	[0.0906, 0.0918]	[0.3584, 0.3616]	[12.8431, 13.0819]	[476.4998, 563.2916]	[0.8856, 0.887]	[0.5858, 0.5896]	-
	50	-	-	-	-	-	-	-	-
Credit	10	[0.2589, 0.2609]	[0.0476, 0.0478]	[0.2545, 0.2565]	[6.4744, 6.5786]	[1055.2896, 1432.8576]	[0.8024, 0.8052]	[0.5945, 0.5995]	-
	30	[0.2814, 0.284]	[0.0515, 0.0517]	[0.2767, 0.2793]	[7.6563, 7.8007]	[1078.4313, 1109.2753]	[0.7972, 0.8014]	[0.5753, 0.5851]	-
	50	-	-	-	-	-	-	-	-
News	10	[0.3764, 0.3786]	[0.0663, 0.0673]	[0.3705, 0.3727]	[13.725, 13.887]	[10501.8434, 25315.8332]	-	-	[43100644.0919, 43152793.1045]
	30	-	-	-	-	-	-	-	-
	50	-	-	-	-	-	-	-	-

Table 13: Confidence intervals for MissForest with 50% additional CTGAN data.

Data set	Miss%	mRMSE	RMSE num	RMSE cat	PFC (%)	Execution time (s)	Accuracy	AUROC	MSE
Mushroom	10	[0.2746, 0.2764]	-	[0.2746, 0.2764]	[7.5421, 7.6341]	[171.5507, 219.2267]	[0.9969, 0.9969]	[0.9968, 0.9968]	-
	30	[0.2897, 0.2915]	-	[0.2897, 0.2915]	[8.3935, 8.4923]	[180.1704, 214.3802]	[0.9848, 0.99]	[0.9869, 0.9869]	-
	50	-	-	-	-	-	-	-	-
Letter	10	[0.0669, 0.0671]	[0.0669, 0.0671]	-	-	[205.8422, 272.709]	[0.8421, 0.8459]	[0.8431, 0.8449]	-
	30	[0.0872, 0.0876]	[0.0872, 0.0876]	-	-	[306.2579, 312.1033]	[0.6616, 0.675]	[0.8246, 0.8316]	-
	50	-	-	-	-	-	-	-	-
Bank	10	[0.3616, 0.3638]	[0.0782, 0.079]	[0.353, 0.3552]	[12.4554, 12.617]	[819.2307, 1134.8067]	[0.8834, 0.8846]	[0.587, 0.5922]	-
	30	[0.3715, 0.3755]	[0.0903, 0.0915]	[0.3602, 0.3644]	[12.9757, 13.2755]	[563.2048, 770.1138]	[0.8851, 0.8871]	[0.5827, 0.5933]	-
	50	-	-	-	-	-	-	-	-
Credit	10	[0.2576, 0.2598]	[0.0476, 0.0478]	[0.2532, 0.2554]	[6.4105, 6.5209]	[2846.8877, 3892.5021]	[0.8012, 0.804]	[0.5932, 0.6]	-
	30	[0.2818, 0.284]	[0.0516, 0.0518]	[0.2771, 0.2793]	[7.6746, 7.801]	[1390.0224, 1576.478]	[0.7958, 0.8014]	[0.5745, 0.5871]	-
	50	-	-	-	-	-	-	-	-
News	10	[0.3768, 0.3788]	[0.0662, 0.068]	[0.3708, 0.3728]	[13.7528, 13.8978]	[16989.6809, 21301.0577]	-	-	[43119644.3581, 43217080.0463]
	30	-	-	-	-	-	-	-	-
	50	-	-	-	-	-	-	-	-

Table 14: Confidence intervals for MissForest with 100% additional CTGAN data.

Data set	Miss%	mRMSE	RMSE num	RMSE cat	PFC (%)	Execution time (s)	Accuracy	AUROC	MSE
Mushroom	10	[0.2986, 0.3038]	-	[0.2986, 0.3038]	[8.9202, 9.23]	[214.227, 224.5298]	[0.9946, 0.9974]	[0.9944, 0.9972]	-
	30	[0.2948, 0.3264]	-	[0.2948, 0.3264]	[8.668, 10.719]	[115.6952, 867.2106]	[0.9866, 0.9944]	[0.9865, 0.9943]	-
	50	[0.2991, 0.3055]	-	[0.2991, 0.3055]	[8.9431, 9.3359]	[286.7699, 292.1133]	[0.9713, 0.9795]	[0.9711, 0.9793]	-
Letter	10	[0.1278, 0.1336]	[0.1278, 0.1336]	-	-	[80.2641, 81.3251]	[0.772, 0.7812]	[0.8808, 0.8856]	-
	30	[0.1369, 0.1451]	[0.1369, 0.1451]	-	-	[21.2092, 21.3458]	[0.5231, 0.5501]	[0.7535, 0.7673]	-
	50	[0.1436, 0.1634]	[0.1436, 0.1634]	-	-	[21.2994, 21.3692]	[0.3331, 0.3585]	[0.6551, 0.6679]	-
Bank	10	[0.451, 0.4632]	[0.1889, 0.2093]	[0.4053, 0.4171]	[16.4328, 17.4064]	[101.4431, 103.9249]	[0.8818, 0.8842]	[0.578, 0.5858]	-
	30	[0.4793, 0.5253]	[0.2399, 0.2813]	[0.4116, 0.4462]	[16.9722, 19.9376]	[124.1753, 128.0031]	[0.8819, 0.8839]	[0.5678, 0.5768]	-
	50	[0.4823, 0.5097]	[0.2302, 0.2602]	[0.4162, 0.445]	[17.3132, 19.8504]	[125.0451, 127.3987]	[0.8741, 0.8779]	[0.5432, 0.5568]	-
Credit	10	[0.2791, 0.2939]	[0.0656, 0.1036]	[0.2636, 0.281]	[6.9484, 7.9154]	[123.0236, 124.9462]	[0.8116, 0.815]	[0.617, 0.6238]	-
	30	[0.2922, 0.3134]	[0.1069, 0.1551]	[0.2657, 0.2767]	[7.0632, 7.6622]	[123.0503, 124.9007]	[0.7968, 0.805]	[0.5978, 0.6122]	-
	50	[0.3173, 0.4143]	[0.134, 0.2454]	[0.2836, 0.3332]	[8.0069, 11.2559]	[123.4548, 125.5914]	[0.8017, 0.8089]	[0.584, 0.6012]	-
News	10	[0.4913, 0.5659]	[0.1969, 0.2087]	[0.4481, 0.5273]	[20.1248, 28.0608]	[154.8964, 157.1596]	-	-	[43211882.6572, 44373943.0628]
	30	[0.5467, 0.6169]	[0.2889, 0.3133]	[0.4529, 0.5397]	[20.6258, 29.3728]	[123.1843, 131.0881]	-	-	[45299198.1798, 46341330.8802]
	50	[0.4855, 0.5325]	[0.1913, 0.2099]	[0.4439, 0.4913]	[19.7657, 24.1901]	[163.5234, 172.2716]	-	-	[45809302.3313, 46508763.2087]

Table 15: Confidence intervals for GAIN v1 with 0% additional CTGAN data.

Data set	Miss%	mRMSE	RMSE num	RMSE cat	PFC (%)	Execution time (s)	Accuracy	AUROC	MSE
Mushroom	10	[0.3053, 0.3143]	-	[0.3053, 0.3143]	[9.3262, 9.8808]	[219.2094, 220.941]	[0.9943, 0.9977]	[0.9942, 0.9976]	-
	30	[0.2976, 0.3032]	-	[0.2976, 0.3032]	[8.8549, 9.1917]	[319.1428, 321.5842]	[0.99, 0.9952]	[0.9898, 0.9952]	-
	50	-	-	-	-	-			-
Letter	10	[0.1403, 0.1479]	[0.1403, 0.1479]	-	-	[81.2913, 83.8507]	[0.7578, 0.7682]	[0.8741, 0.8793]	-
	30	[0.2152, 0.233]	[0.2152, 0.233]	-	-	[22.5887, 22.6357]	[0.4165, 0.4493]	[0.6976, 0.714]	-
	50	-	-	-	-	-	-	-	-
Bank	10	[0.4984, 0.5172]	[0.2359, 0.2657]	[0.4327, 0.4495]	[18.7273, 20.2079]	[104.8297, 106.1343]	[0.8828, 0.885]	[0.5809, 0.5901]	-
	30	[0.5215, 0.5403]	[0.2639, 0.2933]	[0.4448, 0.4582]	[19.787, 21.0086]	[126.6351, 129.6407]	[0.8823, 0.8851]	[0.5662, 0.581]	-
	50	-	-	-	-	-	-	-	-
Credit	10	[0.3054, 0.3162]	[0.1096, 0.1192]	[0.283, 0.2948]	[8.0116, 8.6908]	[124.986, 126.2632]	[0.81, 0.814]	[0.6189, 0.6237]	-
	30	[0.2912, 0.2998]	[0.0811, 0.0891]	[0.278, 0.2878]	[7.7314, 8.2884]	[125.2881, 127.8949]	[0.8008, 0.806]	[0.6072, 0.6184]	-
	50	-	-	-	-	-	-	-	-
News	10	[0.5193, 0.5863]	[0.2211, 0.2311]	[0.4677, 0.5403]	[21.8226, 29.5048]	[157.9226, 160.1772]	-	-	[43274779.7653, 44270342.6291]
	30	-	-	-	-	-	-	-	-
	50	-	-	-	-	-	-	-	-

Table 16: Confidence intervals for GAIN v1 with 50% additional CTGAN data.

Data set	Miss%	mRMSE	RMSE num	RMSE cat	PFC (%)	Execution time (s)	Accuracy	AUROC	MSE
Mushroom	10	[0.3094, 0.32]	-	[0.3094, 0.32]	[9.5733, 10.2393]	[219.2325, 221.2327]	[0.9939, 0.9969]	[0.9953, 0.9953]	-
	30	[0.2995, 0.3063]	-	[0.2995, 0.3063]	[8.9726, 9.3774]	[312.4548, 314.3218]	[0.9849, 0.9911]	[0.9879, 0.9879]	-
	50	-	-	-	-	-	-	-	-
Letter	10	[0.141, 0.1484]	[0.141, 0.1484]	-	-	[82.3464, 83.6602]	[0.7566, 0.7706]	[0.8733, 0.8799]	-
	30	[0.2345, 0.2497]	[0.2345, 0.2497]	-	-	[23.3749, 23.4415]	[0.3889, 0.4099]	[0.6833, 0.6939]	-
	50	-	-	-	-	-	-	-	-
Bank	10	[0.5083, 0.5443]	[0.2535, 0.2979]	[0.4383, 0.4573]	[19.2054, 20.9294]	[106.2673, 108.2887]	[0.8838, 0.8856]	[0.5841, 0.5903]	-
	30	[0.5291, 0.5457]	[0.2753, 0.2937]	[0.4442, 0.467]	[19.7482, 21.8144]	[129.4719, 131.8809]	[0.8818, 0.8838]	[0.5655, 0.5753]	-
	50	-	-	-	-	-	-	-	-
Credit	10	[0.3112, 0.323]	[0.1208, 0.127]	[0.2852, 0.2986]	[8.1309, 8.9229]	[127.5519, 129.5137]	[0.81, 0.8118]	[0.6145, 0.6207]	-
	30	[0.2938, 0.3016]	[0.0851, 0.0933]	[0.2808, 0.2872]	[7.8833, 8.2547]	[127.2921, 128.8859]	[0.8, 0.804]	[0.6027, 0.6117]	-
	50	-	-	-	-	-	-	-	-
News	10	[0.5072, 0.626]	[0.2215, 0.2365]	[0.454, 0.5804]	[20.8373, 34.2319]	[168.9716, 171.1704]	-	-	[43417721.4261, 44133214.2419]
	30	-	-	-	-	-	-	-	-
	50	-	-	-	-	-	-	-	-

Table 17: Confidence intervals for GAIN v1 with 100% additional CTGAN data.

Data set	Miss%	mRMSE	RMSE num	RMSE cat	PFC (%)	Execution time (s)	Accuracy	AUROC	MSE
Mushroom	10	[0.314, 0.3198]	-	[0.314, 0.3198]	[9.8626, 10.2296]	[200.9038, 205.0098]	[0.9929, 0.9961]	[0.9943, 0.9943]	-
	30	[0.3052, 0.3096]	-	[0.3052, 0.3096]	[9.3149, 9.5851]	[291.0009, 293.9769]	[0.9883, 0.9921]	[0.9899, 0.9899]	-
	50	-	-	-	-	-	-	-	-
Letter	10	[0.1468, 0.155]	[0.1468, 0.155]	-	-	[85.4667, 87.3249]	[0.7508, 0.7682]	[0.8692, 0.8784]	-
	30	[0.2587, 0.2775]	[0.2587, 0.2775]	-	-	[25.9867, 26.2843]	[0.3878, 0.4048]	[0.682, 0.6912]	-
	50	-	-	-	-	-	-	-	-
Bank	10	[0.5131, 0.5379]	[0.2566, 0.2902]	[0.4418, 0.455]	[19.5307, 20.7001]	[112.6758, 114.6458]	[0.8825, 0.8853]	[0.5789, 0.5897]	-
	30	[0.5189, 0.5403]	[0.2757, 0.3041]	[0.4365, 0.4493]	[19.0597, 20.1857]	[135.7127, 137.4503]	[0.8825, 0.8843]	[0.568, 0.5756]	-
	50	-	-	-	-	-	-	-	-
Credit	10	[0.3266, 0.334]	[0.1229, 0.1291]	[0.3006, 0.3098]	[9.0348, 9.6052]	[131.4675, 133.4017]	[0.8081, 0.8125]	[0.6145, 0.6221]	-
	30	[0.3041, 0.3159]	[0.0834, 0.1078]	[0.2881, 0.3005]	[8.3088, 9.0344]	[131.3688, 132.8916]	[0.8044, 0.8084]	[0.6, 0.6112]	-
	50	-	-	-	-	-	-	-	-
News	10	[0.4895, 0.5565]	[0.2243, 0.2331]	[0.4335, 0.5063]	[18.6812, 25.9914]	[186.7986, 190.8112]	-	-	[43572832.4217, 44267750.7119]
	30	-	-	-	-	-	-	-	-
	50	-	-	-	-	-	-	-	-

Table 18: Confidence intervals for GAIN v1 with 200% additional CTGAN data.

Data set	Miss%	mRMSE	RMSE num	RMSE cat	PFC (%)	Execution time (s)	Accuracy	AUROC	MSE
Mushroom	10	[0.3261, 0.3353]	-	[0.3261, 0.3353]	[10.6351, 11.2405]	[272.4958, 275.4632]	[0.9909, 0.9949]	[0.9929, 0.9929]	-
	30	[0.3096, 0.3144]	-	[0.3096, 0.3144]	[9.5894, 9.8832]	[294.1031, 296.9455]	[0.9851, 0.9909]	[0.9879, 0.9879]	-
	50	-	-	-	-	-	-	-	-
Letter	10	[0.1539, 0.1591]	[0.1539, 0.1591]	-	-	[93.31, 94.8172]	[0.7417, 0.7561]	[0.8644, 0.8722]	-
	30	[0.2623, 0.2759]	[0.2623, 0.2759]	-	-	[262.0255, 262.0955]	[0.371, 0.392]	[0.6732, 0.6838]	-
	50	-	-	-	-	-	-	-	-
Bank	10	[0.5312, 0.5466]	[0.2714, 0.2898]	[0.4508, 0.469]	[20.309, 22.0182]	[131.4944, 132.544]	[0.883, 0.885]	[0.5817, 0.5911]	-
	30	[0.531, 0.5448]	[0.2827, 0.3025]	[0.4439, 0.4583]	[19.7049, 21.0155]	[152.367, 154.7644]	[0.8817, 0.8849]	[0.5665, 0.5751]	-
	50	-	-	-	-	-	-	-	-
Credit	10	[0.3334, 0.346]	[0.1355, 0.1467]	[0.3023, 0.3155]	[9.1432, 9.961]	[143.4248, 145.6088]	[0.8074, 0.8122]	[0.6149, 0.6223]	-
	30	[0.3162, 0.3336]	[0.0856, 0.0884]	[0.3041, 0.3219]	[9.2629, 10.3617]	[142.7196, 145.4614]	[0.8025, 0.8077]	[0.599, 0.6104]	-
	50	-	-	-	-	-	-	-	-
News	10	[0.5366, 0.5942]	[0.231, 0.2464]	[0.4798, 0.5444]	[23.1887, 29.6587]	[202.3186, 207.4482]	-	-	[43509216.1974, 44346192.5092]
	30	-	-	-	-	-	-	-	-
	50	-	-	-	-	-	-	-	-

Table 19: Confidence intervals for GAIN v1 with 500% additional CTGAN data.

Data set	Miss%	mRMSE	RMSE num	RMSE cat	PFC (%)	Execution time (s)	Accuracy	AUROC	MSE
Mushroom	10	[0.3034, 0.3124]	-	[0.3034, 0.3124]	[9.2052, 9.7628]	[86.2209, 87.6031]	[0.9956, 0.9988]	[0.9955, 0.9987]	-
	30	[0.296, 0.3012]	-	[0.296, 0.3012]	[8.7575, 9.0731]	[132.1886, 133.8758]	[0.9876, 0.9964]	[0.9872, 0.9964]	-
	50	[0.3068, 0.3152]	-	[0.3068, 0.3152]	[9.4154, 9.9406]	[132.7046, 134.931]	[0.96, 0.9704]	[0.9592, 0.9698]	-
Letter	10	[0.1427, 0.1551]	[0.1427, 0.1551]	-	-	[28.0606, 37.5942]	[0.7501, 0.7689]	[0.8701, 0.8797]	-
	30	[0.134, 0.1422]	[0.134, 0.1422]	-	-	[7.04, 7.1006]	[0.5289, 0.5577]	[0.7552, 0.7708]	-
	50	[0.1379, 0.1411]	[0.1379, 0.1411]	-	-	[6.8757, 6.9091]	[0.3574, 0.3756]	[0.6675, 0.6769]	-
Bank	10	[0.4484, 0.4596]	[0.1627, 0.1731]	[0.4172, 0.4264]	[17.4039, 18.1855]	[40.8879, 43.5495]	[0.8833, 0.8851]	[0.5859, 0.5915]	-
	30	[0.4905, 0.4989]	[0.2236, 0.2366]	[0.4335, 0.4421]	[18.7965, 19.5439]	[53.8793, 54.7057]	[0.8806, 0.8842]	[0.5635, 0.5747]	-
	50	[0.5218, 0.5472]	[0.2586, 0.3108]	[0.4386, 0.4632]	[19.2379, 21.4799]	[51.6986, 52.7146]	[0.8763, 0.8785]	[0.5467, 0.5555]	-
Credit	10	[0.2773, 0.2917]	[0.0884, 0.1194]	[0.2612, 0.267]	[6.8298, 7.1266]	[47.2618, 48.6526]	[0.8092, 0.8134]	[0.6161, 0.6235]	-
	30	[0.3003, 0.3283]	[0.1015, 0.1581]	[0.2806, 0.2876]	[7.8758, 8.273]	[47.6234, 47.9292]	[0.8017, 0.8077]	[0.607, 0.6234]	-
	50	[0.3339, 0.3615]	[0.1727, 0.2193]	[0.2802, 0.2916]	[7.8589, 8.5047]	[45.2955, 53.6793]	[0.7981, 0.8061]	[0.5779, 0.5993]	-
News	10	[0.4978, 0.5182]	[0.1986, 0.2064]	[0.4548, 0.477]	[20.6952, 22.762]	[59.3956, 61.6008]	-	-	[43094852.6763, 43473176.6837]
	30	[0.5079, 0.5383]	[0.2499, 0.3039]	[0.4372, 0.448]	[19.1193, 20.0703]	[42.8507, 43.4635]	-	-	[44777825.2403, 45963003.7597]
	50	[0.5119, 0.5293]	[0.2185, 0.2519]	[0.456, 0.4718]	[20.7948, 22.2682]	[54.5477, 66.5001]	-	-	[44061464.7899, 46160851.2701]

Table 20: Confidence intervals for GAIN v2 with 0% additional CTGAN data.

Data set	Miss%	mRMSE	RMSE num	RMSE cat	PFC (%)	Execution time (s)	Accuracy	AUROC	MSE
Mushroom	10	[0.3483, 0.3581]	-	[0.3483, 0.3581]	[12.1292, 12.826]	[86.6777, 87.9795]	[0.988, 0.9936]	[0.9878, 0.9934]	-
	30	[0.2985, 0.3017]	-	[0.2985, 0.3017]	[8.9086, 9.1042]	[132.611, 134.702]	[0.987, 0.994]	[0.9869, 0.9939]	-
	50	-	-	-	-	-	-	-	-
Letter	10	[0.1485, 0.1619]	[0.1485, 0.1619]	-	-	[31.2427, 35.9013]	[0.745, 0.7592]	[0.8676, 0.875]	-
	30	[0.2204, 0.2328]	[0.2204, 0.2328]	-	-	[7.4555, 7.5195]	[0.4051, 0.4341]	[0.6918, 0.7072]	-
	50	-	-	-	-	-	-	-	-
Bank	10	[0.4948, 0.511]	[0.2218, 0.2494]	[0.4395, 0.4485]	[19.3125, 20.1207]	[41.7206, 43.5868]	[0.8826, 0.8848]	[0.587, 0.594]	-
	30	[0.5269, 0.5427]	[0.2633, 0.2677]	[0.4546, 0.4738]	[20.6768, 22.4574]	[52.1487, 53.6971]	[0.881, 0.8842]	[0.5619, 0.5819]	-
	50	-	-	-	-	-	-	-	-
Credit	10	[0.2982, 0.312]	[0.1249, 0.1409]	[0.2694, 0.2794]	[7.2584, 7.8142]	[48.0022, 49.1832]	[0.812, 0.8152]	[0.6187, 0.6275]	-
	30	[0.3215, 0.3387]	[0.0893, 0.1019]	[0.3066, 0.325]	[9.4078, 10.5696]	[47.9587, 48.6817]	[0.8011, 0.8083]	[0.6051, 0.6175]	-
	50	-	-	-	-	-	-	-	-
News	10	[0.528, 0.5394]	[0.2439, 0.2517]	[0.4667, 0.4787]	[21.7793, 22.9149]	[57.5541, 60.3955]	-	-	[43348039.2994, 43811921.019]
	30	-	-	-	-	-	-	-	-
	50	-	-	-	-	-	-	-	-

Table 21: Confidence intervals for GAIN v2 with 50% additional CTGAN data.

Data set	Miss%	mRMSE	RMSE num	RMSE cat	PFC (%)	Execution time (s)	Accuracy	AUROC	MSE
Mushroom	10	[0.3499, 0.3563]	-	[0.3499, 0.3563]	[12.2453, 12.6995]	[86.6742, 88.8718]	[0.9885, 0.9943]	[0.9913, 0.9913]	-
	30	[0.3002, 0.3054]	-	[0.3002, 0.3054]	[9.0082, 9.3268]	[132.902, 134.6184]	[0.9857, 0.9897]	[0.9876, 0.9876]	-
	50	-	-	-	-	-	-	-	-
Letter	10	[0.1551, 0.1687]	[0.1551, 0.1687]	-	-	[26.7616, 30.3622]	[0.7327, 0.7561]	[0.8616, 0.8736]	-
	30	[0.249, 0.2574]	[0.249, 0.2574]	-	-	[7.7111, 7.7613]	[0.3671, 0.3885]	[0.6725, 0.6831]	-
	50	-	-	-	-	-			-
Bank	10	[0.5204, 0.5372]	[0.2527, 0.2691]	[0.4518, 0.4678]	[20.4224, 21.891]	[42.0437, 43.9601]	[0.8815, 0.8861]	[0.5798, 0.5918]	-
	30	[0.5408, 0.5524]	[0.2707, 0.2785]	[0.467, 0.4782]	[21.8194, 22.8694]	[53.539, 55.0914]	[0.8823, 0.8851]	[0.5674, 0.5798]	-
	50	-	-	-	-	-	-	-	-
Credit	10	[0.3123, 0.3241]	[0.1324, 0.1532]	[0.2793, 0.2887]	[7.8003, 8.3395]	[48.7641, 50.1307]	[0.8099, 0.8141]	[0.6183, 0.6257]	-
	30	[0.334, 0.3466]	[0.101, 0.1168]	[0.3165, 0.3281]	[10.0177, 10.7661]	[48.5467, 49.7731]	[0.7994, 0.8092]	[0.5972, 0.6176]	-
	50	-	-	-	-	-	-	-	-
News	10	[0.5377, 0.5477]	[0.2564, 0.27]	[0.4695, 0.4795]	[22.0492, 22.9976]	[58.0198, 59.1858]	-	-	[43315326.3176, 43781472.4854]
	30	-	-	-	-	-	-	-	-
	50	-	-	-	-	-	-	-	-

Table 22: Confidence intervals for GAIN v2 with 100% additional CTGAN data.

Data set	Miss%	mRMSE	RMSE num	RMSE cat	PFC (%)	Execution time (s)	Accuracy	AUROC	MSE
Mushroom	10	[0.3611, 0.3753]	-	[0.3611, 0.3753]	[13.0471, 14.0903]	[86.1335, 103.1861]	[0.9888, 0.994]	[0.9914, 0.9914]	-
	30	[0.302, 0.3062]	-	[0.302, 0.3062]	[9.1186, 9.384]	[132.5648, 134.6666]	[0.9847, 0.9925]	[0.9813, 0.9957]	-
	50	-	-	-	-	-	-	-	-
Letter	10	[0.1668, 0.174]	[0.1668, 0.174]	-	-	[31.6325, 35.0533]	[0.7285, 0.7425]	[0.8592, 0.8662]	-
	30	[0.2492, 0.2622]	[0.2492, 0.2622]	-	-	[8.6048, 8.6762]	[0.3777, 0.3993]	[0.6763, 0.6879]	-
	50	-	-	-	-	-			-
Bank	10	[0.5208, 0.5326]	[0.2436, 0.266]	[0.4556, 0.4658]	[20.761, 21.6954]	[43.4649, 43.7567]	[0.8827, 0.8851]	[0.5784, 0.5876]	-
	30	[0.5444, 0.5588]	[0.2669, 0.2869]	[0.4708, 0.483]	[22.1787, 23.3291]	[54.4483, 55.6941]	[0.883, 0.8852]	[0.5603, 0.5711]	-
	50	-	-	-	-	-	-	-	-
Credit	10	[0.3156, 0.3248]	[0.1313, 0.1531]	[0.2845, 0.2887]	[8.0889, 8.3389]	[49.993, 50.7444]	[0.8104, 0.815]	[0.6222, 0.6292]	-
	30	[0.3483, 0.3623]	[0.0805, 0.0833]	[0.3385, 0.3529]	[11.4721, 12.4473]	[49.743, 50.8522]	[0.8019, 0.8093]	[0.6055, 0.6183]	-
	50	-	-	-	-	-	-	-	-
News	10	[0.5552, 0.571]	[0.2755, 0.2851]	[0.4791, 0.4975]	[22.9612, 24.7544]	[59.9456, 61.5982]	-	-	[42619301.2938, 46268026.8544]
	30	-	-	-	-	-	-	-	-
	50	-	-	-	-	-	-	-	-

Table 23: Confidence intervals for GAIN v2 with 200% additional CTGAN data.

Data set	Miss%	mRMSE	RMSE num	RMSE cat	PFC (%)	Execution time (s)	Accuracy	AUROC	MSE
Mushroom	10	[0.4088, 0.4248]	-	[0.4088, 0.4248]	[16.7056, 18.0566]	[88.3342, 90.619]	[0.9893, 0.9959]	[0.9924, 0.9924]	-
	30	[0.3085, 0.3147]	-	[0.3085, 0.3147]	[9.5154, 9.9036]	[133.9008, 135.5506]	[0.9832, 0.9892]	[0.9861, 0.9861]	-
	50	-	-	-	-	-	-	-	-
Letter	10	[0.1851, 0.2001]	[0.1851, 0.2001]	-	-	[32.673, 34.1362]	[0.6976, 0.7142]	[0.8431, 0.8517]	-
	30	[0.2571, 0.2673]	[0.2571, 0.2673]	-	-	[10.7458, 10.7904]	[0.3673, 0.3887]	[0.6707, 0.6819]	-
	50	-	-	-	-	-			-
Bank	10	[0.5477, 0.5581]	[0.2988, 0.3162]	[0.4549, 0.4637]	[20.6953, 21.5119]	[48.0821, 48.7259]	[0.8833, 0.8847]	[0.5756, 0.5864]	-
	30	[0.5576, 0.5718]	[0.285, 0.2988]	[0.477, 0.4898]	[22.7527, 23.9895]	[59.1162, 60.1304]	[0.8823, 0.8845]	[0.5615, 0.5721]	-
	50	-	-	-	-	-	-	-	-
Credit	10	[0.3596, 0.3724]	[0.1398, 0.1586]	[0.3296, 0.3384]	[10.86, 11.4574]	[53.0683, 54.1907]	[0.8102, 0.813]	[0.6164, 0.6242]	-
	30	[0.3704, 0.383]	[0.0895, 0.0921]	[0.3591, 0.3721]	[12.8964, 13.8456]	[52.9473, 54.2991]	[0.7982, 0.8054]	[0.587, 0.6066]	-
	50	-	-	-	-	-	-	-	-
News	10	[0.5597, 0.5695]	[0.2942, 0.3096]	[0.4715, 0.4823]	[22.2362, 23.2682]	[64.7452, 66.0514]	-	-	[43570695.6523, 44092182.6713]
	30	-	-	-	-	-	-	-	-
	50	-	-	-	-	-	-	-	-

Table 24: Confidence intervals for GAIN v2 with 500% additional CTGAN data.

Hyperparameter selection

Data set	Miss rate %	**kNN**	**GAIN v1 & v2**			**GAIN v2**	
		k	batch-size	hint-rate	alpha	beta	tau
Mushroom	10	41	128	0.1	0.5	0.5	0.5
	30	41	256	0.1	2	50	1
	50	41	256	0.1	10	1	5
Letter	10	11	256	0.9	2	0.1	10
	30	11	64	0.1	2	10	1
	50	11	64	0.1	10	10	10
Bank	10	13	64	0.1	1	0.5	1
	30	13	128	0.1	10	0.1	0.5
	50	13	128	0.1	2	0.1	5
Credit	10	53	64	0.1	2	0.5	10
	30	53	64	0.1	10	1	0.5
	50	53	64	0.1	2	10	5
News	10	19	256	0.9	1	0.5	0.5
	30	19	128	0.9	1	50	0.1
	50	19	256	0.5	1	10	1

Table 25: Summary of the optimal hyperparameters for the methods kNN, GAIN v1 and GAIN v2 for each data set.

GAIN Train and Test Loss

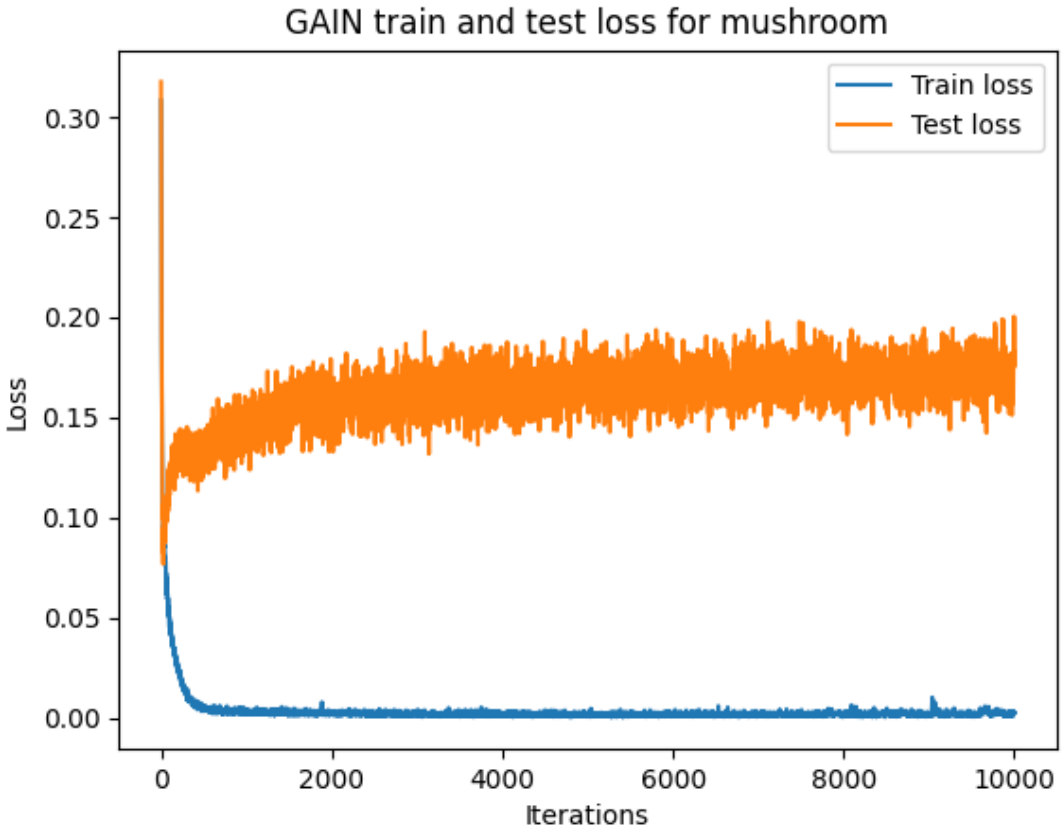

Figure 40: GAIN v1 train and test loss for one run for the mushroom data set with 10% missing values.

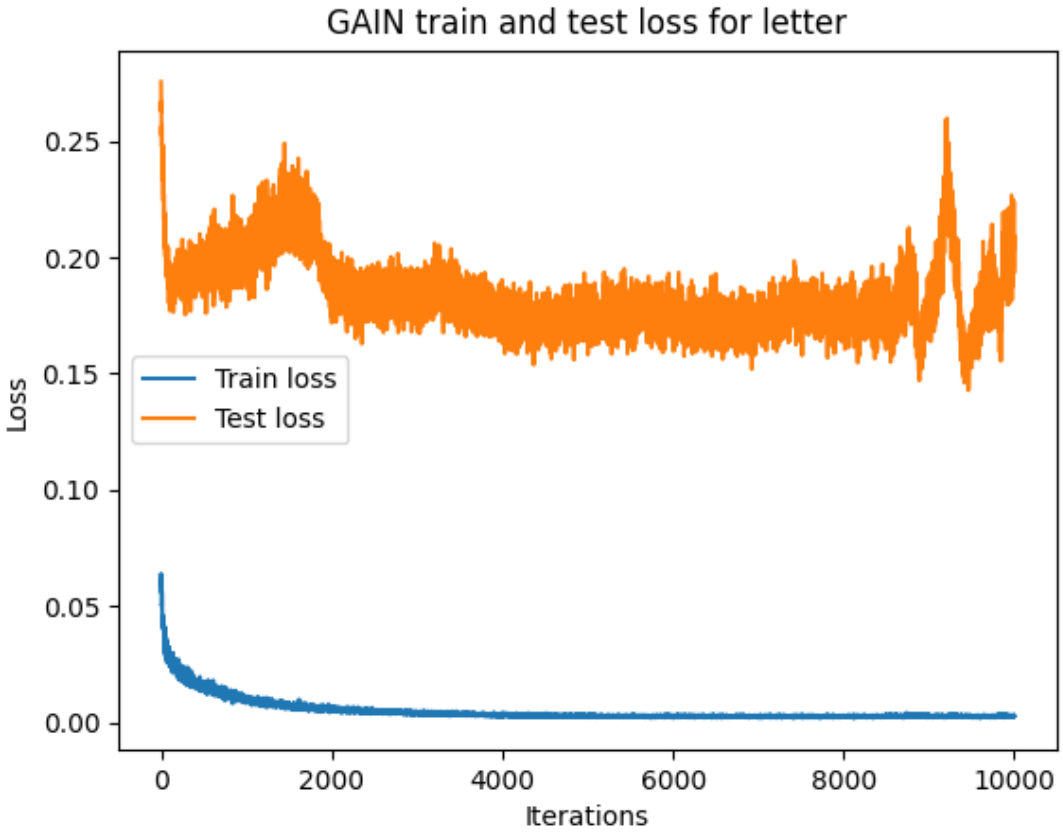

Figure 41: GAIN v1 train and test loss for one run for the letter data set with 30% missing values.

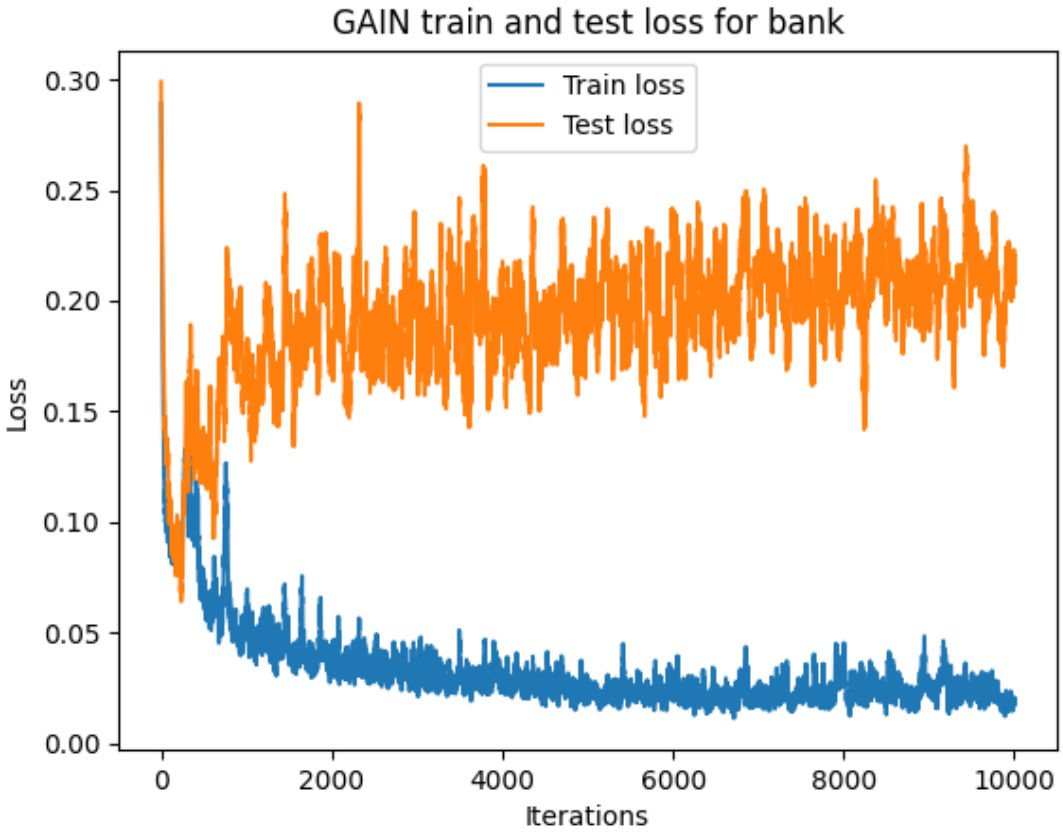

Figure 42: GAIN v1 train and test loss for one run for the bank data set with 50% missing values.

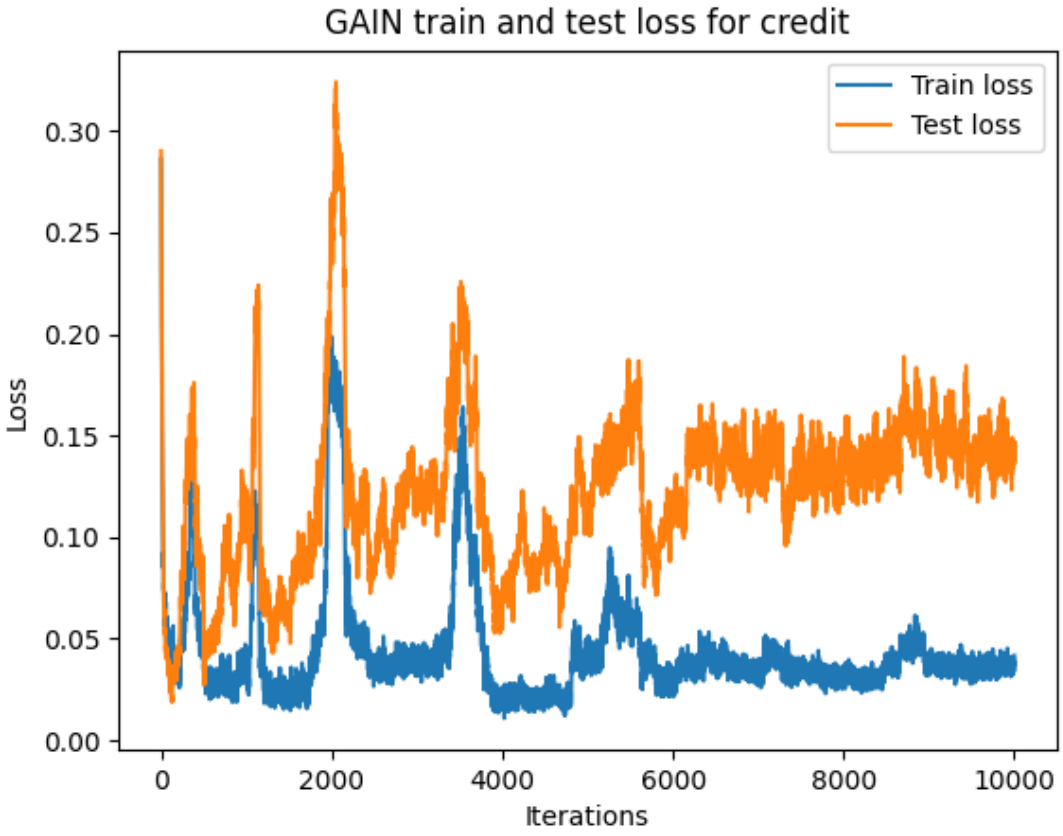

Figure 43: GAIN v1 train and test loss for one run for the credit data set with 50% missing values.

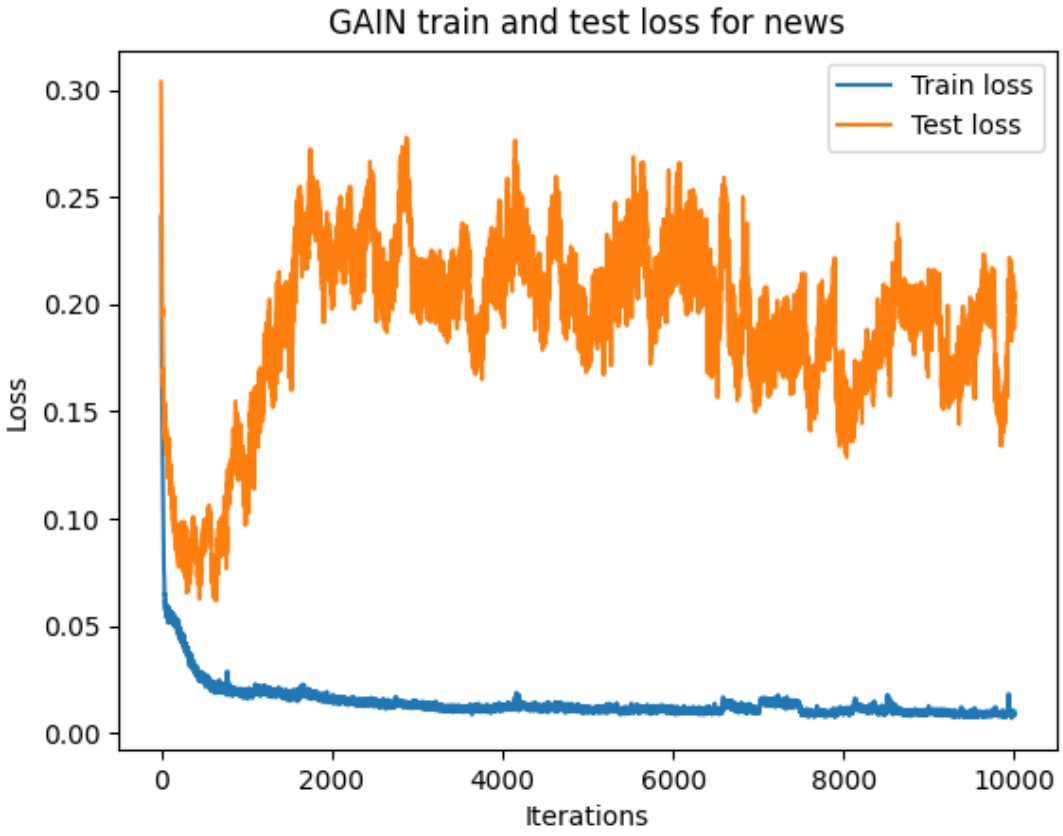

Figure 44: GAIN v1 train and test loss for one run for the news data set with 30% missing values.