Missing data in classification problems

Eva Berezovska

A thesis submitted in partial fulfilment of the requirements for the degree of
Master of Science in
Data - Intensive Analysis

University of
St Andrews

School of Mathematics and Statistics
University of St Andrews

Supervisors:
Dr Nicolò Margaritella
Dr Hannah Worthington

August 2025

Declaration

I certify that this project report has been written by me, is a record of work carried out by me, and is essentially different from work undertaken for any other purpose or assessment.

Signature: [illegible]

Date:
August 12, 2025

Acknowledgements

I would like to express my gratitude to my supervisor, Dr Nicolò Margaritella, for his exceptional guidance, support, and encouragement throughout this project. His readiness to always find time for discussions and to provide prompt and thoughtful feedback have been invaluable. I am especially grateful for such great learning opportunity to work on this topic. My sincere thanks also go to my co-supervisor Dr Hannah Worthington for her valuable support and insightful suggestions. Both of you created a very positive and collaborative environment, working toward a shared goal. Finally, I would like to acknowledge my group mates Lars and Pierre for their peer support and for helping me view the topic from multiple perspective, making the learning process richer and insightful.

Abstract

This work investigates the impact of imputation methods on missing data in classification problems under multiple scenarios. The aim was to identify the best imputation-classification method combinations for varying missing rates, mechanisms, and data types. A synthetic dataset was generated with mixed data types and three levels of predictive influence on a Bernoulli target variable. Missingness was then induced using a balanced mix of MAR and MCAR mechanisms at 10% and 25% rates, applied separately to the training and test partitions of the complete dataset. Three imputation methods (MICE, KNN, Simple Imputer) were applied and evaluated using RMSE and MAE, then imputed data was used to train three classifiers (SVM, Random Forest, Neural Network) and assessed by Accuracy and F1 score. All scenarios were repeated on 100 simulations to ensure consistency and reliability of results. Results indicated a strong influence of missing rate on both classification accuracy and stability. However, imputer-classifier compatibility was also crucial, with some combinations proving more resilient that others. Overall, imputation quality should not be assessed solely on error metrics since top performers by RMSE/MAE did not necessary achieve strong classification performance or preserved interpretability of results. The choice of the best imputer-classifier combination, therefore, requires considering factors beyond error estimates - particularly predictive accuracy, robustness to changes in missingness and ability to preserve true feature importance - in alignment with the goals of the research task.

Table of Contents

1 Introduction

“Obviously the best way to treat missing data is not to have them.”
— *Orchard and Woodbury (1972, p. 697)*

This remark reflects a long-standing discomfort with missing data in scientific research. Traditionally, missingness has been treated as a flaw - something to minimize, ignore, or quietly delete (van Buuren, 2012). However, in many real-world settings, especially in social sciences, healthcare, and survey-based research, incomplete data is nearly unavoidable (Enders, 2010). For classification tasks, this challenge is particularly acute, as missing data can introduce bias, waste valuable information and distort feature relationships that define class boundaries, leading to incorrect decisions in applications like medical diagnosis or risk assessment.

The standard approach in classification problems follows a two-stage process: imputation of missing data followed by classifier training on the imputed dataset. As Shadbahr et al. (2023) points out, while this approach is necessary since most classification algorithms cannot process incomplete data directly, poor imputation can cause algorithms to incorrectly prioritize certain variables, misrepresenting the true factors determining classification outcomes.

The choice of effective imputation methods requires understanding the sources of missingness. Rubin (1976) classified these into three mechanisms based on how the probability of a missing value relates to the data: Missing Completely at Random (MCAR), Missing at Random (MAR) and Missing Not at Random (MNAR). MCAR occurs when missingness probability is uniform across observations and unrelated to any data values - as example when measurement equipment randomly malfunctions (van Buuren, 2012). MAR represents a less restrictive condition where missingness depends on observed data but remains independent of the missing values themselves. For example, when income data is more frequently missing among younger respondents, yet age is fully observed. MNAR occurs when missingness depends on the unobserved values themselves, for example if number of patients in the cancer trial become so ill that they can no longer participate in the study (Enders, 2010). While appropriate imputation techniques exist for MCAR and MAR, MNAR represents a special case. This requires explicit modeling of the missingness process and standard imputation methods generally fail. In practice, missing data often occurs through mixed mechanisms, combining MCAR and MAR, and/or MNAR patterns.

Numerous imputation methods for handling missing data are commonly classified into – single and multiple imputations. Single imputation replaces missing values with one estimate such as the mean, median, or mode. While occasionally sufficient under MCAR and computationally efficient, often distort variable relationships (Enders, 2010). Multiple imputations introduced by Little & Rubin (1987), address these limitations by generating several plausible values for each missing item, creating multiple complete datasets, analysing each separately, and pooling results to capture uncertainty. Popular implementations include Joint Modelling (JM) and Fully Conditional Specification (FCS) also known as Multivariate imputation by Chained Equations (MICE), applicable preferably under MAR (Van Buuren, 2012). Additionally, novel machine learning algorithms such as K-Nearest Neighbors, Support Vector Machines, and Generative Adversarial Imputation Networks are gaining popularity as imputation methods.

Applications of popular imputation methods across various scenarios in classification tasks have been thoroughly examined in several recent esteemed studies that inspired our own work. Shadbahr et al. (2023) evaluated five popular imputation methods Mean/mode, KNN, MICE, MissForest, and GAIN under MCAR scenarios across real-world and simulated datasets with mixed data types and feature importances. They induced 25-50% missingness in both training and test sets, assessing downstream performance with classifiers. Their key findings revealed no single imputation method consistently outperformed others across all datasets, and poor imputation could significantly compromise model interpretability. Counterintuitively, lower-quality imputations were reported for occasionally improved classification performance, possibly acting as regularization. They also identified unresolved technical issues with categorical variable imputation, as most methods produce continuous values rather than required binomial values.

Unlike Shadbahr et al. exploring only MCAR scenarios, Sasu et al. (2025) investigated all three missingness mechanisms (MCAR, MAR, MNAR) with a focus on machine-learning imputation methods KNN, SVM, GAINs, VAE, GRU-D on continuous missing data. Their findings revealed that KNN and SVM consistently outperformed other techniques across different missingness mechanisms. However, they noted limitations of these methods in MNAR scenarios. Their work emphasized that selecting imputation methods based on the specific missing data mechanism is crucial to maintain data integrity. Valid solution to challenge related the categorical features imputation identified by Shadbahr et al. was suggested by El Badisy et al. (2024) – the use of Gower distances. They compared machine learning single imputation methods (KNN, missMDA, CART, missForest, missRanger, missCforest) against multiple imputation methods (miceCART and miceRF) with 30% rate under MAR mechanism. Their key finding revealed a critical trade-off between imputation approaches: single imputation methods delivered superior predictive accuracy while multiple imputation methods produced less biased statistical estimates. Key takeaway was that method selection should align with the primary research goal of either prediction or inference. However, findings were limited to solely MAR scenarios.

The aim of this dissertation was to investigate the impact of missing data imputation methods in classification problems. The objectives were to compare the performance of the imputation techniques on downstream classifier performance, assess the preservation of feature importance relationships following imputation, and identify best imputation-classifier combinations. This was accomplished through a comprehensive set of scenarios that examined multiple missingness mechanisms, mixed data types, and features with different importance weights. The dissertation is organized as follows: Chapter 2 presents the simulation scenarios methodology, covering synthetic dataset generation, missing data induction, three imputation and three classification methods and feature importance methods. Chapter 3 reports the explored scenarios results, including the imputation quality and classification performance evaluation, as well as the feature importance preservation. The dissertation concludes with discussion of key findings and their implications on missing data in classification tasks in Chapter 4. Appendix outlines details on some of the methodologies related to data generation, imputation and classification methods assessment. For clarity in this dissertation, we recognized that terms “variable” and “feature” can be used interchangeably, however the first is more commonly used within statistical domain, while the latter is prevalent in machine learning contexts. Since our study is on intersection of both domains, “Variable” is used when discussing statistical aspects – data generation, missingness mechanisms and statistical imputation methods; “Feature” is used when addressing machine learning components – classification methods and machine-learning based imputation methods.

2 Methods

This chapter outlines the methodology for our simulation study examining missing data scenarios in classification problems. The sections follow the sequential stages of our framework - from data generation through imputation to classification – with each providing the theoretical foundations and implementations details for that phase.

Section 2.1 provides an overview of the simulation design, outlining the framework of missing data scenarios and evaluation metrics. Section 2.2 details the synthetic data generation process. Section 2.3 describes the induced missingness under different mechanisms and rates. Section 2.4 presents the three imputation methods: MICE (2.4.1), KNN (2.4.2), and Simple Imputer (2.4.3). Section 2.5 details the classification methods: Support Vector Machines (2.5.1), Random Forest (2.5.2), and Neural Networks (2.5.3). Finally, Section 2.6 explains the assessment of feature importance preservation across all scenarios.

2.1 Overview of simulation design

Missing data scenarios framework of the simulation study is schematically illustrated in Figure 2.1 and described below:

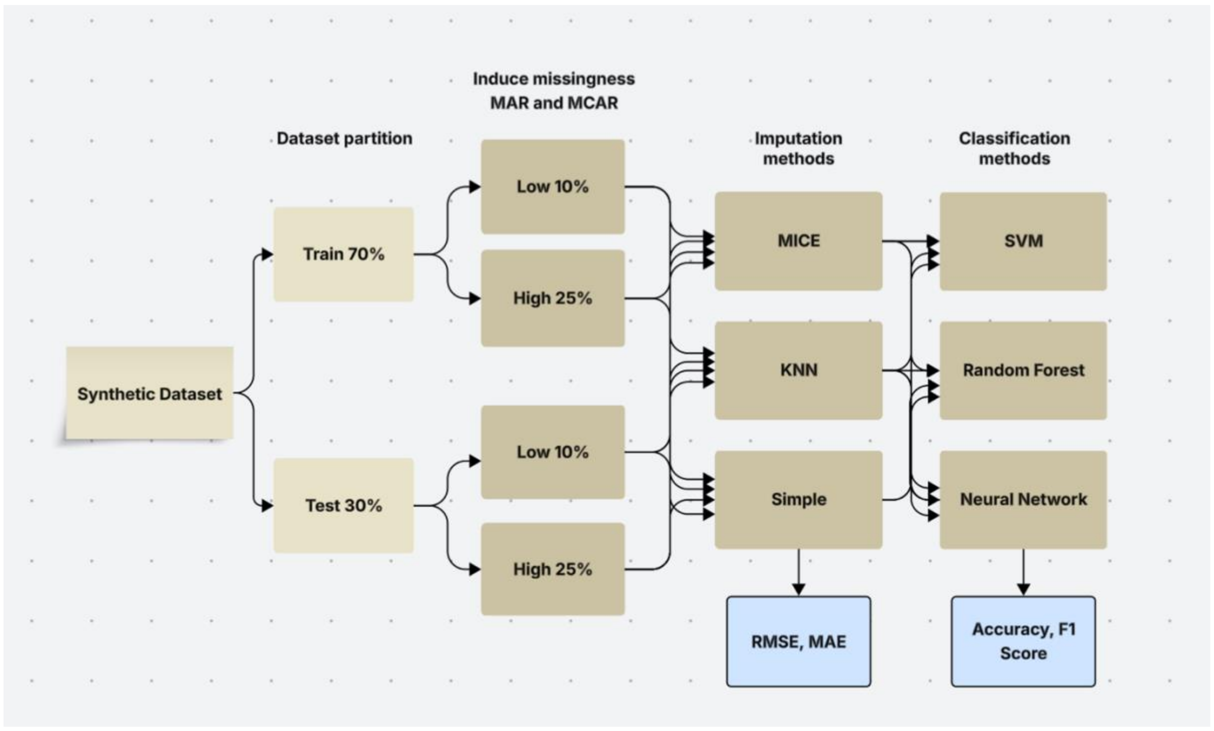


Figure 2.1 Overview of the simulation framework for evaluating imputation and classification performance under missing data scenarios.

Each simulation starts with generating a synthetic complete dataset with mixed data types – continuous, discrete and categorical. The complete dataset is partitioned into train set containing 70% of observations and a test set containing 30%. Missingness is induced into train and test sets separately under MCAR and

MAR mechanisms with a balanced proportion 50/50. One train and test pair is induced at 10% and the other pair at 25% missing data rate. Both train and test sets under both missing rate scenarios were imputed by three different imputation methods – MICE (Multiple Imputation by Chained Equations), KNN (K-Nearest Neighbours) and Simple Imputer. Imputation quality was evaluated on train and test sets using Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) metrics. Finally, each pair of imputed by different imputer datasets was used to train three classification methods – SVM (Support Vector Machine), Random Forest and Neural Network. The classification results were evaluated using Accuracy and F1 score. Each classification algorithm calculated feature importance.

We repeated the entire simulation process 100 times to ensure statistical stability in both imputation quality and classification performance metrics.

2.2 Synthetic dataset generation

The properties of generated synthetic dataset are presented in Table 2.1:

Dataset property	**Values**
Sample size	1000
Number of predictors	25
Data types	15 continuous
	5 discrete
	5 categorical (4 binary, 1 multinomial nominal)
Response variable	1 binary (Bernoulli)
Predictive Variables	7 high important
	9 medium important
	9 low important
Missing mechanisms	MCAR and MAR (50/50 proportion)
Missing rates	10% and 25%

Table 2.1 Properties of the generated synthetic dataset per each iteration of simulation

Variables are generated independently using Python `numpy.random` module: 15 continuous variables generated following a standard normal distribution using `np.random.normal()`; 5 discrete variables generated following Poisson distribution with different λ (lambda) values, ranging from 2 to 6, using `np.random.poisson();` 5 categorical variables were created using `np.random.choice(),` including 4 binomial variables with two levels "Yes" and "No" and 1 multinomial nominal variable with 3 levels ("Level_A", "Level_B", "Level_C"). All numerical variables both continuous and discrete were standardized with `StandardScaler()` from `scikit-learn`. This ensured that all numeric values had mean 0 and standard deviation 1. Categorical variables were one-hot encoded using `OneHotEncoder()` from `scikit-learn`, with the first category dropped to avoid multicollinearity.

We verified that distributions appear as intended for their data type by plotting histograms. Continuous variables overall are normally distributed with no heavy skewness or heavy tails. Discrete and categorical variables showed reasonable frequency patterns across levels. Response variable was confirmed to have balanced classes. Variable distributions for each representative data type and target shown in Figure 2.2, other plots of remaining variables follow a similar pattern and can be found in Appendix A. We verified that there is no multicollinearity among predictors as it could lead to distortion of their individual weight (influence) on the response variable. To assess multicollinearity, we computed Variance Inflation Factor (VIF) for all numerical features using the variance_inflation_factor() function from Python's statsmodels library. All VIF values fell between 1 and 2, suggesting – no severe collinearity between the variables.

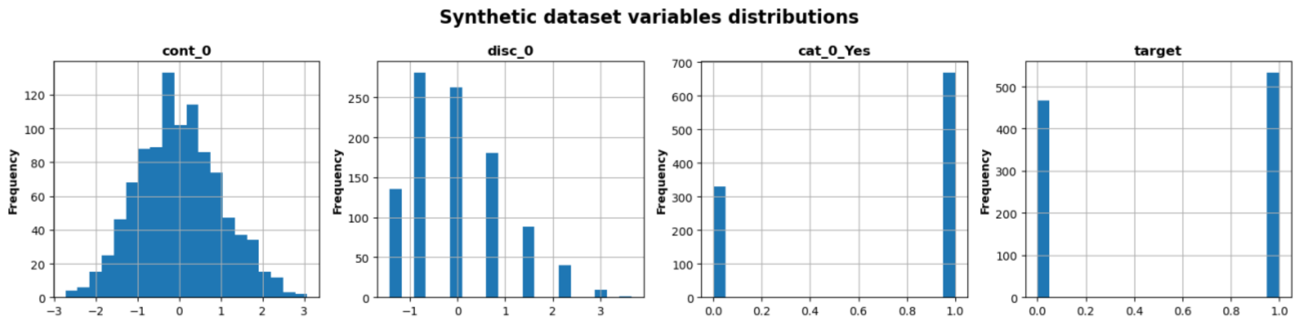


Figure 2.2 Distribution of representative generated standardized variables. Histograms show one variable of each data type: continuous (cont_o), discrete (disc_0), categorical (cat_0_Yes) and binary target variable

In real datasets, predictors typically do not have equal contribution to the target variable, thus we manually assigned weights to each variable to mimic this feature. Weights were selected to ensure the linear predictor XtB does not exceeds 5 in logistic regression model as inverse logit values above 5 produce probabilities > 0.99. This would create an overly deterministic rather probabilistic relationship between predictors and outcome. Thus, all 25 variables were grouped into three levels of predictive importance - high, medium and low and assigned weights within the range (3 and -3) in absolute value. Each group was assigned a different range of coefficient values, with high-importance variables receiving larger weights (range 2.0 – 3.0 in absolute value), medium-importance variables receiving moderate weights (range 1.0 – 2.0 in absolute value), and low-importance variables, receiving near-zero weights (0.0 – 1.0 in absolute value). Other configurations of weight ranges have been tested from higher (between 12.0 and 0.1) to lower (between 0.7 and 0.05) influencing the classification results. This is outlined in detail in Appendix A. Each level is representative with all three data types numerical and categorical. A complete list of variables and their assigned weights is presented in Table 2.2.

High importance	**Weight high**	**Medium importance**	**Weight medium**	**Low Importance**	**Weight low**
Cont_0	2.09	Cont_1	-1.64	Cont_2	-0.4
Cont_3	-2.66	Cont_5	1.42	Cont_4	-0.12
Cont_7	-2.31	Cont_9	-1.25	Cont_6	0.72
Disc_1	2.81	Cont_11	-1.45	Cont_8	0.21
Disc_3	2.81	Cont_13	-1.84	Cont_10	-0.08
Cat_0_Yes	-2.45	Disc_0	-1.93	Cont_12	-0.0
Cat_2_Yes	-2.42	Disc_4	1.49	Cont_14	0.02
		Cat_1_Yes	1.05	Disc_2	-0.2
		Cat_4_Level_C	-1.43	Cat_3_Yes	-0.67
				Cat_4_Level_B	0.91

Table 2.2 List of high, medium and low importance variables and their weights. Presents weights distribution within the assigned ranges on 1 simulation of generated dataset, variables generated in other simulations preserve their group of importance within assigned weight range, but specific weight value is different.

These weights determine each variable's relationship with the response variable. To generate the classification target variable, we applied a logistic model where each feature was multiplied by its assigned weight based on importance. The sum of these products was passed through the logistic function ensuring that more important variables have a stronger influence on the response probability. The response variable is yi ~ Bernoulli (pi), where logit(pi) = XiTBeta, yi – response variable (e.g. 0 or 1), pi – probability of success for the i-th observation, Xi – vector of predictor variables for the i-th observation, Beta – vector of coefficients (weights), XiTBeta – product between variables and weights (i.e., linear predictor).

Initially, this setup produced an imbalanced class distribution with a strong bias toward the positive class. To achieve a more balanced class distribution, we searched for an intercept value that would shift the average predicted probability closer to 0.5. As a result, an intercept of 4 was the closest for making the balanced class leading to a less biased classification models results.

2.3 Missing Data Simulation

This section details the rationale for the chosen missing data rates, selection of MCAR and MAR variables for data amputation, including MAR predictors, and the technical implementation of inducing missingness using the chosen library.

As a preprocessing step before inducing missingness, we partitioned the complete synthetic dataset into training (70%) and test (30%) sets using `train_test_split()` from `scikit-learn`. This produced 700 samples in the training set and 300 samples in the test set. The implemented stratified split based on the target variable to maintain balanced class distribution in both subsets. The early dataset partitioning is a common practice with imputation methods aimed to preserve integrity of results for the following reasons:

First, early splitting prevents data leakage. When imputation methods learn patterns from observed data to fill missing values, applying these methods to a complete dataset before splitting would allow information from test observations to influence training imputations - creating artificially optimistic performance estimates during classification tasks. Second, for MAR mechanisms, missingness depends on relationships between observed variables. Introducing missingness before splitting would disrupt these dependency patterns when partitioning, potentially creating inconsistent missingness mechanisms between subsets. By splitting first, we ensure MAR mechanism maintain the statistical properties independently within each subset. This approach creates a proper evaluation framework where test data remains truly unseen during both imputation and subsequent classification steps, leading to realistic performance evaluation.

To induce missingness, we used the `MultivariateAmputation` class from Python `pyampute` library. This library implements the multivariate amputation framework originally developed by

(Schouten et al., 2018) under the supervision of Prof. Stef van Buuren. The package offers high control over missingness patterns, mechanisms (MCAR, MAR, MNAR), frequency, and targeted variables. The method was first introduced by Schouten as R function ampute in the famous multiple imputation package `mice`, and later translated into Python as `pyampute`. Professor van Buuren is widely recognized as the creator of Multiple Imputation by Chained Equations (MICE) - the imputation method we explore later in this study.

Another important decision concerns the choice of missing data rate. There is no common definition of what rate of missing data is considered as "low" or high". However, we consulted established guidelines from literature on when missing data rate can become problematic to inform our decision on missing data rates. Schafer (1999) suggests that 5% missingness represents a minimal threshold, while others indicate bias may increase notably beyond 10% (Bennett, 2001) or 20% (Peng et al., 2006). However, these thresholds are best interpreted in context of missing mechanism present and generally are considered as part of discussion about what amount of missing data can be neglected. Therefore, we considered missing rate of 10% as a threshold when missingness has limited impact on model bias, labelling it as "low" missing data rate. While we have also chosen 25 % rate as a more severe, yet manageable impact on bias, labelling it as "high" missing data rate.

The missing rate within `pyampute` is specified using the `prop` argument. However, this means only the fraction of data to amputate within the selected features for amputation and does not translate to overall dataset missingness. We achieved 10% and 25% overall missingness by empirically tuning the prop value (excluding the response variable). The actual missing rate was obtained using the formula:

$$\text{Missing Rate (\%)} = \left(\frac{\text{Total Number of Missing Values}}{\text{Total Number of Predictor Cells}}\right) \times 100$$

The key `pyampute` parameter is `pattern`, which specifies how missingness is introduced in the dataset - by the mechanism (MCAR, MAR, MNAR or combined), by the list of incomplete variables to be affected, by the frequency (relative probability of applying each pattern) and optionally, predictors and weights used to predict MAR-affected variables.

We implemented balanced amputation mechanisms with 50% of MCAR and 50% of MAR, applied separately to the training and test datasets. In practice, missing data often contains a mix of MCAR and MAR mechanisms and identifying the true cause of missingness is not a straightforward task. Some values may have been lost randomly and others due to observed variables, thus mixing the mechanisms in our study makes it more realistic.

The features selected for each mechanism were chosen to ensure representation across data types (continuous, discrete, categorical) and predictive importance levels (high, medium, low). The list of MCAR and MAR imputed features in the Table 2.3 below. MAR predictors assigned are features that are nor MCAR or MAR imputed – Cont_7, Cont_9, Cont_11 and Disc_3 with associated weights {1.5, 1.3, 1.1, 1.0}. These weights determine the likelihood of a value being missing as a function of those predictors, in line with the MAR definition. The assignment of weights in the MAR mechanism follows a logistic function internally in pyampute. That is, for a variable XjXj governed by MAR, the probability that XjXj is missing is modeled using a logistic transformation of a weighted linear combination of the specified predictors. Notably, pyampute package does not prefer categorical features as MAR predictors so we did not use them.

MCAR features	Importance level	MAR features	Importance level
Cont_0	High	Cont_3	High
Cont_1	Medium	Cont_4	Low
Cont_2	Low	Cont_5	Medium
Disc_2	Low	Disc_0	Medium
Disc_4	Medium	Disc_1	High
Cat_0_Yes	High	Cat_4_Level_B	Low
Cat_3_Yes	Low	Cat_4_Level_C	Medium

Table 2.3 Table MCAR and MAR variables. Presents the list of variables that we used to induce MCAR and MAR missing mechanisms, representative across data types and levels of predictive variables.

As a result, we created four datasets with induced missingness: 1.Train and test with 10% missingness (low) and 2.Train and test with 25% missingness (high).

2.4 Imputation Methods

At this point, we generated complete synthetic dataset and induced the missingness at 10% (low) and 25% (high) rate with mixed MCAR and MAR mechanisms. This section outlines the implementation of three imputation methods MICE and KNN as more advanced, and Simple Imputer as a baseline. All three methods assessed with RMSE and MAE metrics. While MICE and KNN were implemented in R since it supports a better environment for our tasks, Simple Imputer was implemented in Python.

2.4.1 Multivariate Imputation by Chained Equations (MICE)

Theoretic background and methodology

MICE or Multiple Imputation by Chained Equations is popular flexible statistical method for dealing with missing data, particularly useful with mixed data types. MICE implement Rubin's multiple imputation framework through an iterative approach. The fundamental concept of MICE derives from recognizing that missing data creates uncertainty that must be properly quantified in subsequent analyses (van Buuren and Groothuis-Oudshoorn, 2011).

The statistical foundation of MICE is based on Rubin's (1987) three-step process. First, multiple completed datasets are created where missing values are replaced with plausible values drawn from appropriate posterior predictive distributions. Second, standard complete-data statistical methods are applied to each imputed dataset independently. Finally, results are pooled using Rubin's rules that properly account for both within-imputation variance (statistical uncertainty if we knew true imputation values) and between-imputation variance (uncertainty due to missing data).

While Rubin originally proposed joint modelling strategies (e.g., assuming a multivariate normal distribution), MICE adopt an alternative approach known as Fully Conditional Specification (FCS), or chained equations (van Buuren, 2012), which fits within Rubin's general multiple imputation method but avoids the need to specify a full joint distribution.

In FCS, each variable with missing data is imputed using its own univariate conditional model, specified appropriate to variable type (e.g., Bayesian linear regression for continuous variables, logistic regression for binary variables etc). Each variable's imputation is conditioned on all other variables, including those with previously imputed values. This cycle repeats for a specified number of iterations (`maxit`), creating a Markov chain that converges to a stationary distribution representing the joint uncertainty about missing values (van Buuren et all., 2011). The entire process repeats (`m`) times independently to generate multiple complete datasets.

Key MICE parameters are `m` and `maxit`. Parameter `m` (number of imputations) determines how many complete datasets MICE produces. Originally, Rubin (1987) suggested that 3 to 5 imputations are sufficient for valid inference in many cases. However, later research showed that higher values of `m` (20 or more) can reduce Monte Carlo error and improve the reproducibility of estimates, especially when the fraction of missing information is high (van Buuren et all., 2011). Parameter `maxit` (number of iterations) controls how many iteration cycles are performed within each imputation. Higher `maxit` leads to reaching convergence, where imputation stabilize. Usually, values are in the range from 5 to 10, but the decision should be based on convergence diagnostics, such as trace plots.

Implementation and parameters optimization

We started our MICE implementation by transforming previously one-hot encoded categorical variables to factors, as otherwise MICE would have incorrectly treated them as numerical values. Importantly, we discovered that even though discrete variables were scaled and visually resembled continuous distributions, their underlying structure was not truly Gaussian as expected by MICE's "`norm`" method, which we set initially. Our visual diagnostics through strip plots in Figure 2.3 revealed this mismatch clearly. For instance, when applying the "`norm`" method to discrete variables, MICE imputed values following a Gaussian distribution while the true values exhibited a distinctly different pattern. In contrast, when using the "`pmm`" method for discrete variables, the imputed values closely matched the actual distribution of observed data.

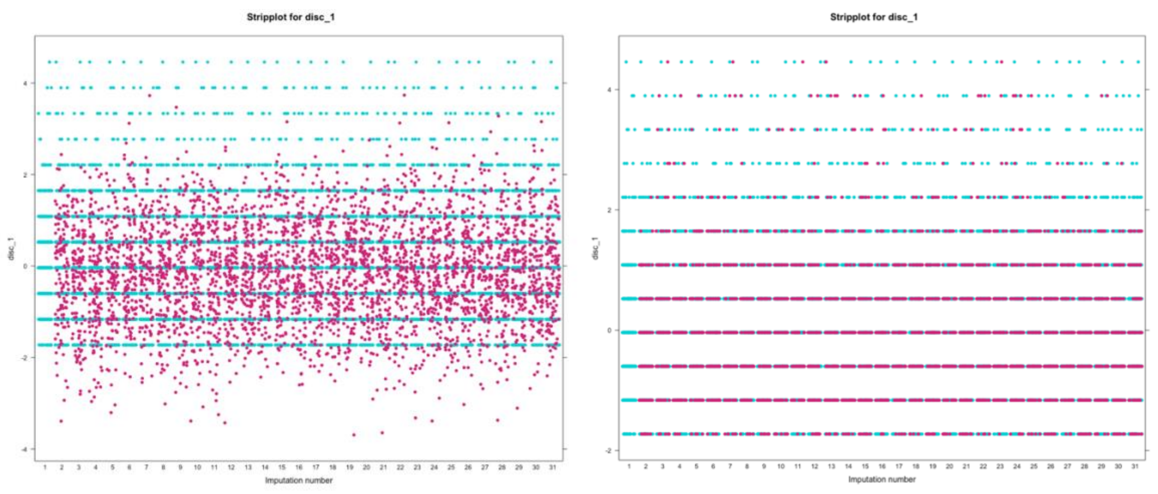


Figure 2.3 Figure Stripplots for discrete variables. Left: Distribution mismatch using the "norm" method, where imputed values follow Gaussian distributiob unlike the original discrete data. Right: Correct distribution preservation using the "pmm" method, which matches the discrete structure of the original values.

Therefore, we assigned the MICE methods for each data type in line with the statistical properties of each variable: `"norm"` (Bayesian linear regression) - for continuous variables, `"pmm"` (predictive mean matching) - for discrete numeric variables, and `"logreg"` (logistic regression) for binary categorical variables.

To initialize MICE, we followed the recommended approach from the official documentation (van Buuren et all., 2011) by running mice first with maxit = 0 so that we do the setup first and customize the methods and tuning afterwards. Another important property is prediction matrix, allowing to specify predictors for the variables with missing values if those are known, for instance in case with MAR. But since the design of our scenarios was aimed for comparing the performance of multiple imputation methods, we did not want to give mice an unfair advantage. Other imputers we explored do not have that special property. So, prediction matrix was constructed to allow all variables to be used as predictors for imputing others except for each variable was not used to predict itself (diagonal set to 0), and the outcome variable.

For tuning we optimized the two key parameters: "`maxit`" and "`m`", separately for the datasets with low and high levels of missingness. To reduce computational cost, this parameter optimization was conducted using a single simulation run. Tuning and search of best parameters was executed solely on training sets and only then applied on test. The selected best parameters values were then applied on 100 simulations repetitions.

To determine the best `"maxit"` value, we used trace plots - a standard diagnostic tool for assessing convergence in multiple imputation. These plots display how mean imputed values change across iterations for each imputation chain. According to van Buuren (2011), convergence is indicated when streams from different imputations appear well-mixed and fluctuate randomly around a stable level, without visible trends or separation between chains. We inspected these plots for each variable with missing values, increasing "`maxit″` until the chains stabilized across all imputations with all streams overlapped freely and showed no sign of systematic divergence. For both low and high missingness datasets, trace plots demonstrated stable trajectories by iteration 20 as shown in Figure 2.4 with no persistent trends or oscillations beyond this point. Consequently, we selected maxit = 20 as our best value. The trace plots for other variable types can be found in Appendix B.

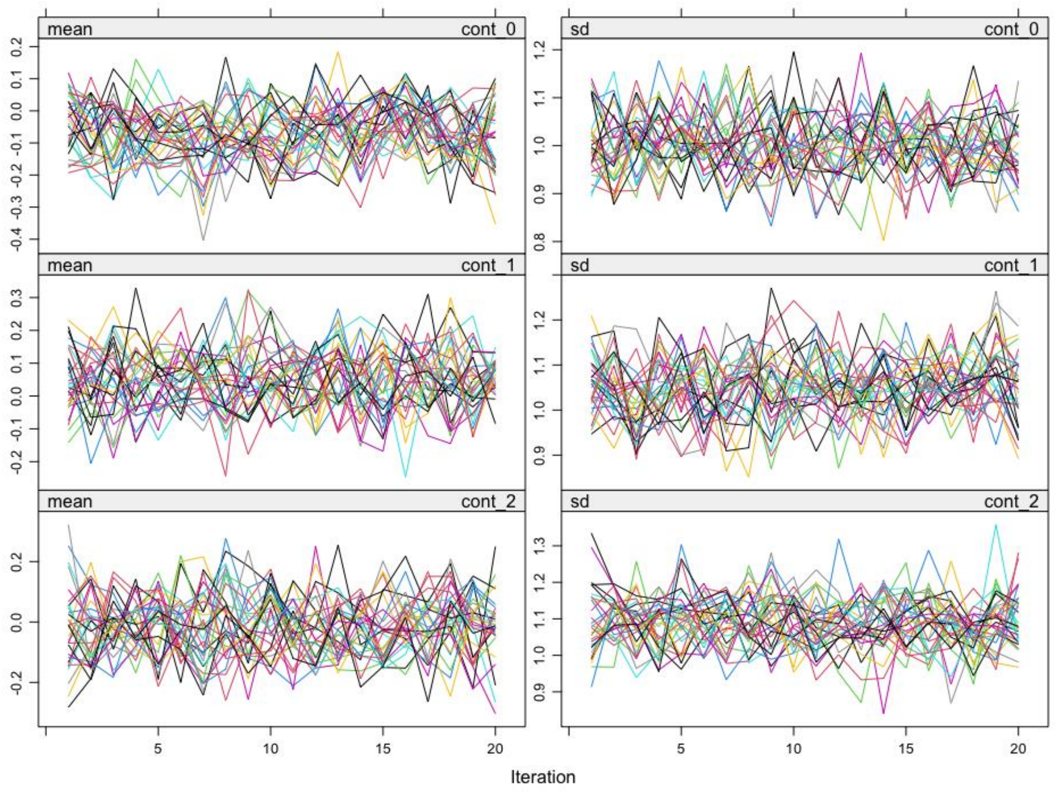


Figure 2.4 Trace plots for convergence check in MICE shows mean imputed values across 20 iterations for three continuous variables with reached convergence.

To select the number of imputations “m” we used RMSE density plots as an empirical diagnostic tool. The plot visualizes the distribution of root mean squared errors (RMSE) for a given variable with missing data (cont_3 in our case) across multiple imputations and repeated simulations. For each candidate value of “m” (20, 30, and 40), we ran 20 independent simulations, recording the RMSE for each imputed dataset.

By visualizing the RMSE distributions, we evaluated both central tendency (typical imputation accuracy) and spread (variability across imputations and runs). As shown in Figure 2.5 based on high missing rate dataset, as “m” increased, the RMSE distributions became narrower and more concentrated, indicating improved stability and reduced simulation variability. At “m” = 30, the distribution was relatively unimodal and tight, with negligible additional benefit from raising to 40. We therefore selected *m* = 30 as a balanced choice - sufficiently high to ensure reliable imputations without unnecessary computational cost. Similar results were obtained for low missing rate and can be seen in Appendix B.

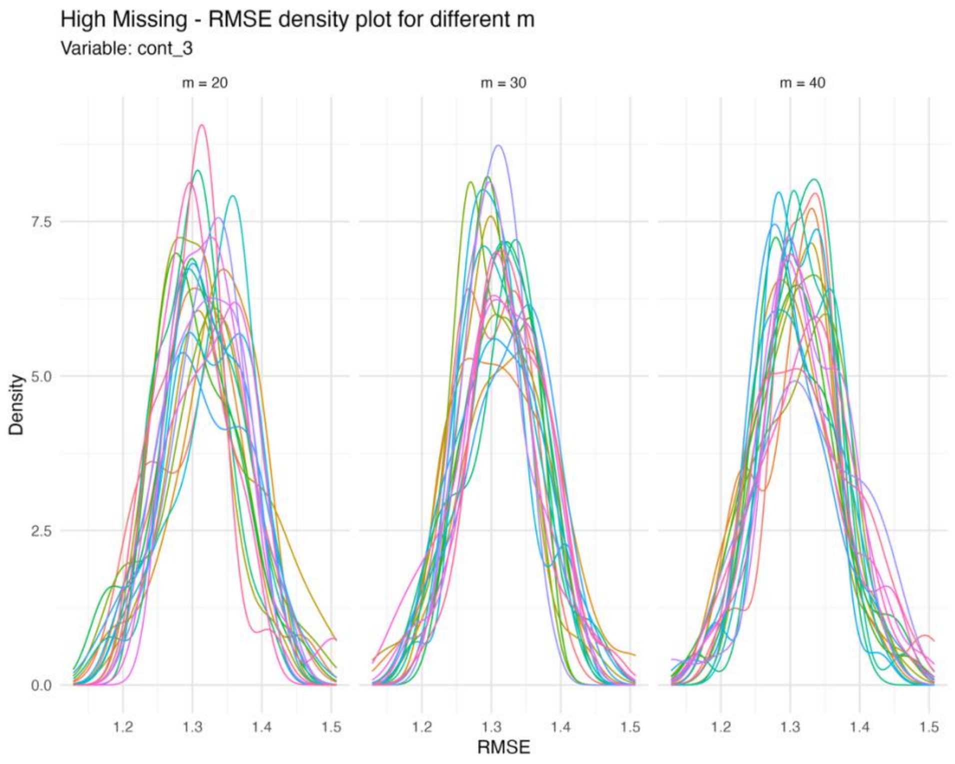


Figure 2.5 Figure RMSE Density plot for different "m" shows the distributions of RMSE on 20 iterations with three values of m=20, 30 and 40 for variable cont_3 in high missing rate dataset

Finally, we applied selected best m and maxit parameters on train and test dataset of both low and high missing data pairs and calculated pooled RMSE and MAE metrics in line with Rubin's rule. Results were presented and discussed in the following chapters in details. However, appropriate implementation of imputed by MICE datasets would require application of each to train each classifier. Given the selected parameters, each simulation would produce 120 imputed datasets (30 imputations multiplied by 4 high and low train and test sets) per one simulation. On all 100 simulations that would require training three classification models on 1200 MICE imputed datasets resulting in 36000 datasets. That was not feasible computationally in our circumstances, therefore we compromised by producing pooled estimates based on 30 imputations with 20 iterations per each MICE simulation and run on 100 simulations in total. Visual MICE diagnostics was also obtained using standard MICE documentation tools and can be found in Appendix B.

2.4.2 K-Nearest Neighbors (KNN) Imputation

Theoretic background and methodology

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm commonly used for classification tasks. KNN is considered a "lazy learner" since it doesn't explicitly train a model to make predictions; instead, it classifies new observations based on the majority class among their *k* closest neighbors, determined using a distance metric such as Euclidean or Manhattan distance. The parameter *k* represents the number of neighbors considered, directly influencing the model's bias-variance trade-off (Albon, 2018). As Alborn explained: "If k = n where n is the number of observations, then we have high bias but low variance. If k = 1, we will have low bias but high variance. The best model will come from finding the value of k that balances this bias-variance trade-off."

This core idea of neighborhood-based reasoning naturally extends to missing data imputation. In KNN imputation, the goal is to replace each missing value by estimating it from the corresponding values in its *k* nearest complete observations. For a given incomplete record, the algorithm identifies its *k* nearest neighbors using a distance metric computed over the available (non-missing) attributes. Then, for each missing variable, imputed value is derived by aggregating the observed values from these neighbors—using either a statistical function (e.g., mean or median for continuous variables) or a voting rule (e.g., majority category for categorical variables).

The effectiveness of KNN imputation critically depends on the choice of distance metric. Standard distance measures such as Euclidean or Manhattan distance assume all variables are continuous and scaled. These assumptions do not hold for categorical variables. To address this, we used Gower distances (Gower, 1971), a general-purpose similarity measures appropriate for mixed-type data. Gower distance defines the pairwise dissimilarity between observations by computing attribute-wise distances, appropriately scaled and type-specific, and then aggregate them. For continuous variables, it calculates the absolute difference normalized by the range of the variable. For categorical variables, it assigns a score of 0 if two values match and 1 otherwise.

Implementation and parameters optimization

KNN imputation was implemented in R using the `kNN()` function from the `VIM` package (Kowarik & Templ, 2016), using the parameter "metric" = "gower" to ensure we treat mixed data appropriately. Selected parameters and their values outlined in the Table 2.4:

Parameter	**Selected value**	**Definition**
Distance "metric"	gower	Applied Gower distances for variables
Number of neighbors "k"	25	Number of closest neighbors
Numeric imputation function "numFun"	mean	Averages the values from the selected neighbors
Categorical imputation function "catFun"	maxCat	Chooses the most frequent category from neighbors
Standardization "method"	range	Ensures that all features contribute equally to the distance calculation by scaling them between 0 and 1
Missing values indicator "impNA"	TRUE	Indicates that missing values should be replaced
Imputed variable indicator "imp_var"	FALSE	Disables creation of additional binary indicators for imputed variables to maintain consistency across all methods.

Table 2.4 Selected parameters and definitions for KNN imputation.

For KNN Imputer the most important parameter is k, representing the number of neighbors used in imputation. It seems easy to select the highest possible value for k, but we need to be mindful about the bias-variance trade-off mentioned before. To determine a suitable k, we implemented a grid search for each candidate k combined with a hold-out validation from the training data. Since the true values were known from the complete dataset before amputation, we computed the RMSE and MAE between the imputed and true values. We repeated this procedure for all values of k and aggregated the performance metrics. The results visualized with elbow plots displaying RMSE at candidate values of k. This visualisation diagnostics helped to identify at what level of k we have the lowest error (RMSE) and from which point increasing k does not lead to significant gain. As seen in Figure 2.6, for the low-missingness dataset, the validation RMSE initially decreased with increasing k, until k = 25, beyond that point the gain was not reasonable. Similar results were obtained on high missing dataset, that can be found in Appendix B.

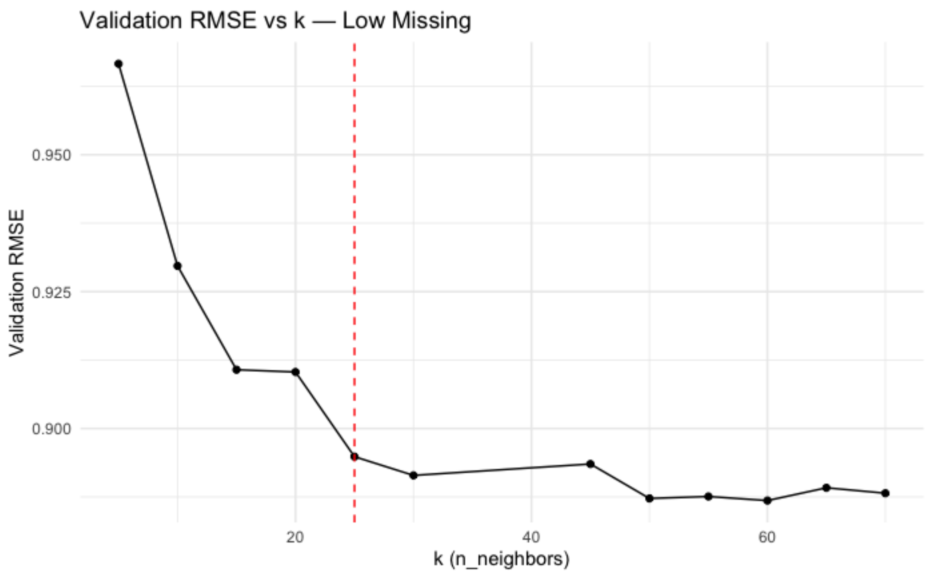


Figure 2.6 Figure Elbow plot RMSE vs k values shows the best value of k = 25 parameter for KNN Imputer on dataset with low missing data rate with respect to bias-variance trade-off.

After we selected the best k for both missing rate scenarios, we applied KNN Imputer on train sets and then on test sets, obtained evaluations presented and discussed in detail in the following chapters. Imputed by KNN datasets obtained after running on all 100 simulations were then trained on classification methods.

2.4.3 Simple Imputation

Simple Imputation is the basic statistical method used for handling missing data. It replaces each missing value with a simple statistic such as mean, median or mode of the observed data. Unlike multiple imputation methods, Simple Imputation is a single imputation technique, meaning each missing entry is replaced with a single estimated value. Although it is straightforward and computationally efficient, simple imputation may underestimate variance and inflate statistical certainty since it ignores the inherent uncertainty of missingness (van Buuren, 2012).

In our analysis, we used the `SimpleImputer` class from the `scikit-learn` library, separately handling numerical and categorical features. For numerical features, missing values were imputed with the mean of observed data, whereas categorical features were imputed using the most frequent (mode) category:

First, the imputers were fitted only on the training set using Python `fit_transform` method and then applied to the test set with `transform`. Same as with other imputation methods we evaluated the metrics separately for missing rate scenarios.

2.5 Classification methods

Previously we generated synthetic datasets, introduced missingness under MCAR and MAR mechanisms as low (10%) and high (25%) missing rates and applied three imputation methods – MICE, KNN and Simple Imputer to impute the missing values. This section outlines the application of classification methods (classifiers) on imputed by each imputation method dataset: Support Vector Machines (SVM), Random Forest (RF), and Neural Network (NN). Classification performance was evaluated based on Accuracy and F1 Score. Additionally, we calculated the feature importance of each classification method - to compare if the results match the initial setting when we assigned weights for features at the generation stage. For each classifier we provided brief background on the algorithm, justification of our choice of hyperparameters and implementation details.

For classifiers hyperparameter optimization, we first performed an extensive grid search on a single iteration. Our goal was to identify top performing hyperparameters across different imputation methods and missingness levels. Based on the results, we then narrowed the grid search for the main simulation phase with multiple iterations. A more focused tuning grid reduced the computational cost while we targeted most promising values for hyperparameters. This strategy was applied for each classification method in the following section.

2.5.1 Support Vector Machines (SVM)

Support Vector Machines (SVMs) are supervised learning algorithm commonly used for classification, both linear and nonlinear, as well as for regression (Geron, 2022). SVM goal is to find a decision boundary, referred to as a hyperplane, that best separates data points from different classes. The boundary needs to be the one that maximizes the margin between the closest points of opposing classes, referred as support vectors (Géron, 2022; Albon, 2018). As Geron noted: “You can think of an SVM classifier as fitting the widest possible street between the classes”. SVMs “shine” on smaller or medium-sized datasets, which matches the property of our synthetic datasets.

We used `SVC` from `sklearn.svm` to implement SVM. The classifier has several hyperparameters that influence its ability to find the optimal decision boundary. The list of key SVM’s hyperparameters and their role are described in the Table 2.5:

Hyperparameter	**Role**
“C”	Regularization parameter that controls the trade-off between maximizing the margin and minimizing classification error. A low C allows for a wider margin but tolerates more misclassifications, while a high C aims to classify all training examples correctly by fitting a narrower margin.

“kernel”	Specifies the kernel function to transform the data. Can be 'linear' and 'rbf' (Radial Basis Function). The 'linear' kernel aims to find a linear separating hyperplane, while 'rbf' maps data into a higher-dimensional space to capture non-linear relationships.
“gamma”	Controls how much weight the model gives to the training points nearby. A low gamma leads to smoother, more generalized decision boundaries; a high gamma means the model pays more attention to each point individually, which can lead to overfitting.

Table 2.5 Key SVM hyperparameters list

Full list of candidate values used for an extensive grid search on single iteration and the narrowed grid search applied for simulation stage presented in Table 2.6 below. This resulted in a total of up to 120 combinations tested per dataset. Each combination was evaluated using the 5-fold cross validation on training set:

Hyperparameters	**Extensive search candidates (one iteration)**	**Focused search candidates (100 iterations)**
“C”	1, 2, 3, 4, 5, 10, 15, 20, 25, 30, 40, 50	1,2,3,10,15,20
“kernel”	'linear', 'rbf'	'linear'
“gamma”	'scale', 'auto'	'scale'

Table 2.6 SVM hyperparameters candidates list

Lower values of “C” (e.g. 1,2,3) were generally more effective on high-missing datasets, likely due to increased noise and uncertainty. Larger values of “C” (e.g. 10, 15, 20) performed better on low missingness or complete datasets, preferring more strict decision boundaries where data was more reliable.

2.5.2 Random Forest

Random Forest is an ensemble machine learning method that combines the predictions of multiple decision trees to improve accuracy and robustness. As Géron (2022) describes, this approach relies on the “wisdom of the crowd,” where aggregating diverse models outperforms any individual one. RFs use bagging (bootstrap aggregating): each tree is trained on a random subset of data and considers a random subset of features at each split. This increases model diversity and reduces overfitting by lowering variance at the cost of slightly higher bias. Predictions are made via majority voting. Random Forests perform well on complex, noisy datasets and scale effectively. Géron emphasizes their power and simplicity, calling them “one of the most powerful machine learning algorithms available today.”

We used `RandomForestClassifier` from `sklearn.ensemble` with the list of key hyperparameters and their role described in Table 2.7:

Hyperparameter	**Role**
“n_estimators”	Number of trees in the forest. A higher number can improve stability but increases computation time.
“max_depth”	Maximum depth of each tree. Shallower trees reduce overfitting and training time.

“min_samples_split”	Minimum number of samples required to split a node. Higher values constrain tree growth and improve generalization.
“min_samples_leaf”	Minimum number of samples required at a leaf node. This helps prevent overly specific splits.
“max_features”	Number of features to consider when looking for the best split.
“bootstrap”	Whether bootstrap samples are used when building trees.
“oob_score”	Enables out-of-bag estimation for model validation.
“class weight”	Setting to 'balanced' compensates for any class imbalance caused by missingness.

Table 2.7 Key Randon Forest hyperparameters list

List of candidate values used for the initial extensive and the narrowed grid search applied for simulation stage presented in Table 2.8.This resulted in a total of up to 1440 combinations tested per dataset. Each combination was evaluated using the 5-fold cross validation on training set:

Hyperparameters	**Extensive search candidates (one iteration)**	**Focused search candidates (100 iterations)**
“n_estimators”	50, 100, 150, 200	50, 100, 150 (high only)
“max_depth”	1, 2, 3, 4, 5, 10	2–5 (complete/low), 1–2 or 5 (high)
“min_samples_split”	50, 100, 150, 200, 250	50, 100 (complete/low), 100, 250 (high)
“min_samples_leaf”	20, 30, 40, 50, 60, 70, 80	20, 30 (complete/low), 40, 60 (high)
“max_features”	'sqrt', 'log2'	'sqrt' (complete/low), 'log2' (high)
“bootstrap”	True	True
“oob_score”	True	True
“class_weight”	'balanced' (high only)	'balanced' (high only)

Table 2.8 Random Forest hyperparameters candidates list

For high-missing datasets, shallower trees (max_depth = 1 or 2), larger leaf sizes (min_samples_leaf = 60+), and fewer considered features (max_features='log2') were more effective because these settings reduced overfitting caused by noise (both due to imputation and generated features with low importance). For complete and low-missing datasets, deeper trees (up to 5), smaller leaves (20–30), and broader feature consideration ('sqrt') improved model expressiveness and accuracy.

2.5.3 Neural Networks (NN)

Neural Networks (NNs) are powerful machine learning models inspired by the structure of the human brain. Each unit (or neuron) receives inputs, applies weights and a bias, and passes the result through a non-linear activation function (Albon, 2018). The most common architecture is the feedforward neural network, or Multilayer Perceptron (MLP), which consists of an input layer, one or more hidden layers, and an output layer (Albon, 2018).

Neural networks learn by a process known as backpropagation: the model computes predictions, compares them to true labels using a loss function, and updates weights to minimize error. This cycle repeats over multiple epochs until convergence. According to Géron (2022), neural networks are universal function approximators and excel at modeling complex, non-linear patterns - particularly when using deep architectures. However, their performance depends heavily on careful preprocessing, initialization, and sufficient data.

We used `MLPClassifier` from `scikit-learn` and with the list of key hyperparameters and their role described in the Table 2.9:

Hyperparameters	**Role**
“hidden_layer_sizes”	Defines the architecture of the network.
“activation”	Non-linear transformation applied at each neuron.
“alpha”	L2 regularization term to prevent overfitting by penalizing large weights.
“learning_rate_init”	Initial step size for the optimization algorithm. Balances speed and stability of learning.
“max_iter”	Number of epochs (passes over the training set). Set high to ensure convergence during grid search.

Table 2.9 Key Neural Network hyperparameters list

Candidate values are presented in the Table 2.10 below. This resulted in a total of up to 216 combinations tested per dataset. Each combination was evaluated using the 5-fold cross validation on training set:

Hyperparameters	**Extensive search candidates (one iteration)**	**Focused search candidates (100 iterations)**
“hidden_layer_sizes”	(50,), (100,), (50, 50), (100, 50)	(50, 50), (100, 50) *(complete, low)* (10,), (50,), (100,) *(high)*
“activation”	'relu', 'tanh', 'logistic'	'tanh', 'logistic' *(complete, low)* 'logistic' *(high)*
“alpha”	0.0001, 0.001, 0.01, 2.0	0.0001, 0.001, 0.01 *(complete, low)* 0.0001, 0.01, 2.0 *(high)*
“learning_rate_init”	0.001, 0.01, 0.1	0.001, 0.01
“max_iter”	2000, 3000	2000, 3000

Table 2.10 Neural Network hyperparameters candidates list

In general, we found that simpler architectures (e.g., one hidden layer with fewer units) and stronger regularization performed better on datasets with high missingness, helping reduce overfitting caused by the noise. Conversely, slightly deeper networks and more flexible activation functions (e.g., 'tanh') performed well on more complete datasets, likely benefiting from their capacity to learn richer representations where data quality allowed.

2.6 Feature Importance

To access how accurately imputation and classification methods preserved the feature importance relationships a classifier-specific methods were used for high-missing and low-missing scenarios imputed by different imputers. Additionally, feature importance was calculated for complete dataset for comparison.

For SVM, we extracted feature importance from the model coefficients, using the absolute values of `model_coef_[0]`. In linear SVMs, which is our case, these coefficients directly represent each feature's influence on decision boundary. Larger absolute values indicate stronger impact on classification. Random Forest classifiers have built-in feature importance via the `model_feature_importances` attribute, which quantifies importance based on the reduction in impurity (Gini) when splitting on a particular feature across all trees in the forest. Neural Networks required a different approach since they don't have such direct measure. We used `permutation importance` from `scikit-learn`, which works by randomly shuffling each feature's values and measuring the resulting drop in model performance. We ran this with 5 repetitions to get stable results.

3 Results

3.1 Imputation results

The aim of this section is to evaluate the performance of three imputation methods – MICE, KNN and Simple Imputer based on different missing scenarios. As mentioned in Section 2.4 imputation methods were evaluated on train and test sets using RMSE and MAE metrics. This section outlines comparison of imputation methods performance by missing rates, both distributions based on 100 iterations and aggregated average results. Further detailing into performance results stratified by imputed data types with various missing mechanisms and missing rates. Distributions of 100 iterations are presented in figure plots, aggregated average results based on 100 iterations are demonstrated in tables.

3.1.1 Sample-wise performance

Overall, imputation performance stratified by method and missing rate low (10%) and high (25%) is presented in Figure 3.1. Both metrics results show that Simple Imputer provided relatively better imputation quality with the lowest error rates in both missing rate scenarios. KNN followed closely while MICE showed the highest error rates in both scenarios. Notably, both Simple and KNN achieved similar performance with both missing rates scenarios, and MICE had the largest discrepancy. The boxplot width indicates the stability of each method's performance based on 100 simulations. All methods had more narrow distributions, hence more stable results in high missing scenario and higher variability in low missing scenario. Simple Imputer demonstrated the widest spread of variability, particularly in low missing conditions indicating less stable results.

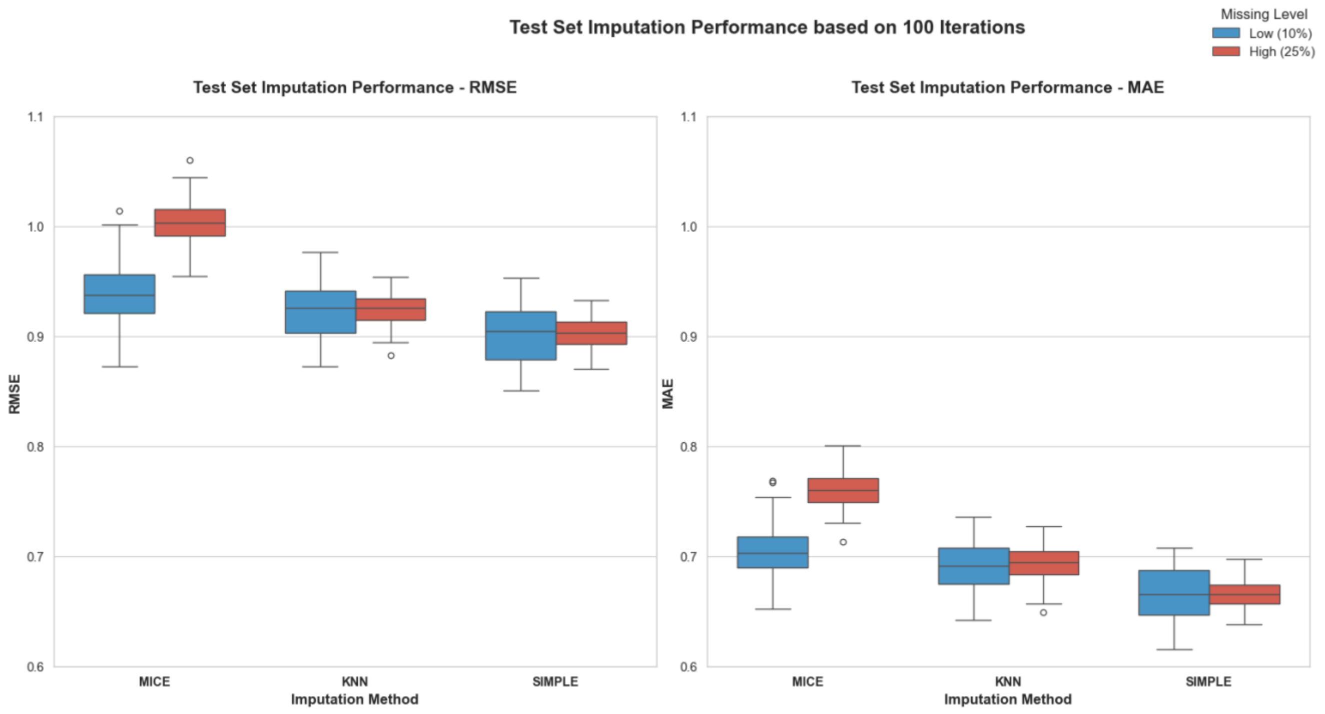


Figure 3.1 Test set imputation performance based on 100 iterations. Boxplots present the distribution of RMSE (left) and MAE (right) per each imputation method applied on low (10%) and high (25%) missing rates.

Aggregated by average imputation performance metrics across all methods based on 100 simulations shown in Table 3.1 confirms the patterns observed in the boxplots. Based on test set results, Simple Imputer achieved the lowest errors with RMSE values of 0.9025 and 0.9018 for low and high missing rates respectively. KNN followed closely with values of 0.9227 and 0.9241, while MICE showed higher errors at 0.9399 and 1.0047. RMSE test performance change from low to high missingness for KNN was only 0.15% worse (i.e. RMSE on high missing rate was 0.15% higher than on low missing). Simple Imputer demonstrated modest change too, while MICE had a substantial 6.89% RMSE and 7.83% MAE increase. This pattern is visible in the Figure 3.1 as well. The performance difference between train and test sets were minimal (less than approximately 1.01%) for KNN and Simple Imputer. While for MICE the difference was more substantial, particularly at high missing rates where test RMSE and MAE were higher that respective train sets by 5.15% and 6.02% respectively.

Dataset	Method	RMSE (Low 10%)	RMSE (High 25%)	RMSE % Change	MAE (Low 10%)	MAE (High 25%)	MAE % Change
Train	KNN	0.9135	0.9186	0.56	0.6840	0.6876	0.54
	MICE	0.9187	0.9555	4.01	0.6867	0.7163	4.31
	SIMPLE	0.9046	0.9046	0.01	0.6685	0.6685	-0.01
Test	KNN	0.9227	0.9241	0.15	0.6904	0.6938	0.50
	MICE	0.9399	1.0047	6.89	0.7042	0.7594	7.83
	SIMPLE	0.9025	0.9018	-0.08	0.6657	0.6662	0.08

Table 3.1 Average imputation performance metrics based on 100 iterations. Lower values indicate better imputation quality for both RMSE and MAE. The % change for both metrics indicate performance difference from low to high missing rate scenario: positive values suggest the error increase, negative values – decrease.

3.1.2 Data type performance

As mentioned in Section 2.3 missingness was induced representatively across all data types – continuous, discrete, categorical. Also, both MAR and MCAR mechanisms were applied on data types proportionally: 3 continuous, 3 discrete and 2 categorical variables per each mechanism. Thus, imputation methods performance by data type, missing mechanism and rate could been fairly evaluated and compared.

Imputation performance stratified by data type and missing mechanism (MAR vs MCAR) is summarized in Table 3.2 as average RMSE results based on 100 iterations. All methods achieved similar performance levels for continuous variables on both missing mechanisms with RMSE in range 0.99 – 1.08. Simple Imputer demonstrated relatively better results based on RMSE 1.00 for MCAR, 0.99 for MAR, KNN closely followed with RMSE 1.02 for MCAR, 1.01 for MAR. Missing mechanism had minimal impact of continuous variables imputation, RMSE difference between MAR and MCAR is appx. 0.01. Discrete variables had similar results with the only notable difference for MICE showing modestly better results under MAR than MCAR (RMSE 0.03 difference).

RMSE performance for categorical variables ought to be interpreted with caution since they have constrained ranges (binary values 0 and 1) leading to producing lower error ranges than for numerical variables. Thus, although it may visually appear that imputation performance was better for categorical variables, the comparison of categorical with numerical variables it is misleading based on RMSE and MAE metrics. However, across the methods Simple Imputer again performed relatively better with the lowest error rates RMSE 0.54 under MAR, 0.64 under MCAR, followed by KNN with RMSE 0.60 under MAR, 0.66 under MCAR and MICE with RMSE 0.61 under MAR, 0.68 under MCAR. In this case, all methods showed better results under MAR compared to MCAR: Simple Imputer – at 0.10, KNN – at 0.06 and MICE at 0.07 RMSE reduction.

The distribution of RMSE results based on all iterations revealed that MICE showed the widest spread in performance for numerical variable types with both missing mechanisms, except for categorical variables under MCAR. For all other instances, under MAR mechanism all methods showed somewhat more narrow distributions rather than under MCAR, particularly for categorical variables. Full details provided in Appendix C.

Data type	MICE (MCAR)	MICE (MAR)	KNN (MCAR)	KNN (MAR)	SIMPLE (MCAR)	SIMPLE (MAR)
continuous	1.08	1.07	1.02	1.01	1.00	0.99
discrete	1.08	1.05	1.02	1.01	1.00	0.99
categorical	0.68	0.61	0.66	0.60	0.64	0.54

Table 3.2 Average imputation performance based on RMSE by method, variable type and missing mechanism based on 100 iterations

Imputation performance stratified by data type and missing rates (low 10% and high 25%) is summarized in Table 3.3 as average RMSE results based on 100 iterations. Simple Imputer and KNN demonstrated best and stable performance across both low and high rates and all data types. Except for results for categorical variables where Simple Imputer performed better: RMSE 0.59 for low and high, while KNN's 0.62 for low and 0.63 for high. In contrast, MICE showed a notable performance reduction from RMSE 1.04 on low missing rate to 1.11 on high rate. As well as somewhat lower performance on all data types and missing levels overall.

The distribution of RMSE values across all iterations for this varying missing rates again revealed that MICE showed the widest performance variability for numerical variable types, while Simple Imputer demonstrated stability across missing rates. Full details provided in Appendix C.

Data type	MICE (Low)	MICE (High)	KNN (Low)	KNN (High)	SIMPLE (Low)	SIMPLE (High)
continuous	1.04	1.11	1.01	1.01	1.00	1.00
discrete	1.03	1.11	1.01	1.02	1.00	1.00
categorical	0.62	0.66	0.62	0.63	0.59	0.59

Table 3.3 Average imputation performance based on RMSE by method, variable type and missing rate based on 100 iterations

Overall, Simple Imputer consistently outperformed other methods in all scenarios with lowest test RMSE 0.9 and most stable performance, followed closely by KNN with RMSE 0.92 and MICE with RMSE 0.94. MICE had the greatest sensitivity to change from low to high missing rates with + 6.89% RMSE increase. Data type evaluation demonstrated that categorical variables had sufficiently better performance under MAR mechanism compared to MCAR for all methods. While numerical variables were minimally impacted by the missing mechanism.

3.2 Classification results

The aim of this section to evaluate the impact of imputation methods on downstream classification performance across missing rate scenarios. As detailed in Section 2.5, classification methods SVM, Random Forest and Neural Network were trained on imputed datasets and evaluated using Accuracy and F1 Score metrics. This section provides classification performance results applied on imputation methods visualized by distribution of metrics obtained on 100 iterations and summarized by average performance.

Overall classification accuracy performance on test sets for each classifier stratified by imputation methods and missing rate based on 100 iterations is presented in Figure 3.2. Three panels represent classifiers (SVM, Random Forest, Neural Network) with colour-encoded imputation methods (orange for MICE, purple for KNN, green for Simple Imputer, and dashed red for compete dataset). Solid lines represent low missing rates (10%), dotted lines – high missing rates (25%).

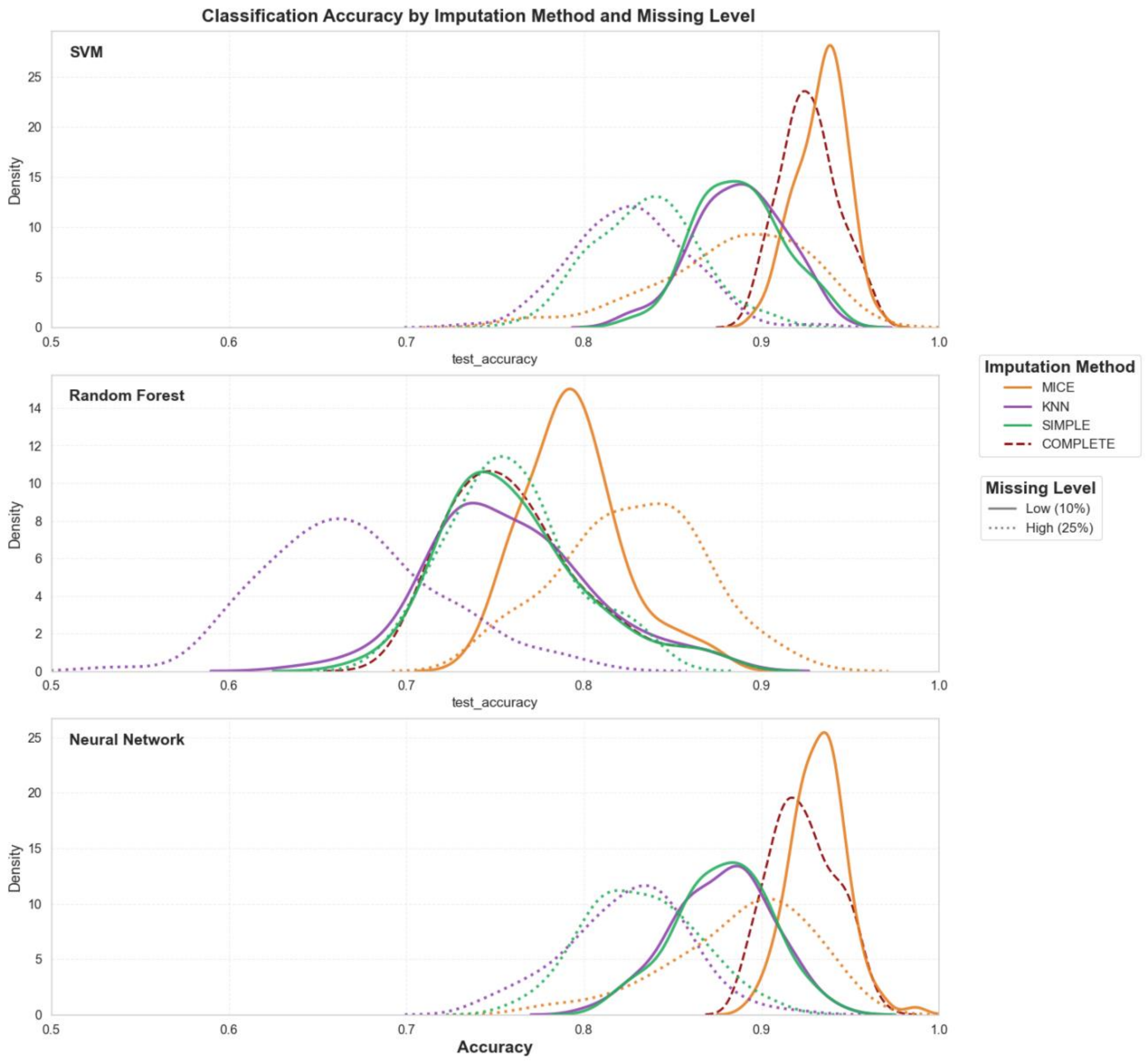


Figure 3.2 Classification performance based on 100 iterations. Density plots show distribution of accuracy scores per classifier (SVM, Random Forest, Neural Network) for each imputation method applied on low (10%) and high (25%) missing rates.

MICE impact on classification performance

Unlike the imputation quality results, MICE-imputed datasets demonstrated impressive performance results on classification. Trained on SVM, MICE performed superior compared with other methods, particularly at low missing rates. Average test accuracy score achieved 0.934 by MICE slightly outperformed Complete dataset performance (0.927). At low missing rate, Accuracy distribution achieved over 100 iterations is within approximately [0.88 and 0.97] range, showing a tall, concentrated peak, which indicates consistently high performance. However, at high missing rates performance declined to mean accuracy of 0.883. Distribution shifted to the broader range within [0.75 and 0.95] considerably wider, which indicates increased variability of performance in high-rate scenario.

Random Forest classifier performance was lower for all imputation methods compared with other classifiers. MICE, however, achieved a surprising result in this combination – mean accuracy of 0.795

which exceeded the Complete dataset (0.762) at low missing rate. Distribution on 100 iterations was within approximately [0.74 and 0.9] range showing a prominent peak around [0.78 - 0.81]. Even more notably, at high missing rates MICE was the only case when mean Accuracy surpassed the low missing results with mean Accuracy 0.827. Distribution shifted to more broad range within [0.75 and 0.95].

Combination with Neural Network was almost identical at low missing rates as with SVM – achieved mean accuracy 0.932, slightly better than Complete set results (0.924). At low missing rate, Accuracy distribution is within approximately [0.90 and 0.97] range with a small tail towards 1.00. Similar to SVM results, at high missing rates performance decreased to accuracy of 0.888 and distribution shifted to the broader range within [0.75 and 0.95].

KNN impact on classification performance

Overall, KNN-imputed datasets demonstrated moderate performance results on classification methods, considerably lower than MICE yet comparable or slightly lower than Simple Imputer in most scenarios. Trained on SVM, KNN achieved mean test Accuracy at 0.888, underperforming both MICE (0.934) and Complete dataset performance (0.927). At low missing rate, Accuracy achieved based on 100 iterations was within approximately [0.8 and 0.95] range, resulting in a moderately wide distribution with lower peak height. This was an indication that KNN performed less consistently than MICE. At high missing rates, performance significantly declined to mean accuracy of 0.826. Distribution shifted to the broader range within [0.70 and 0.90] considerably wider, which indicates increased variability of performance in high-rate scenario.

Combination of KNN with Random Forest classifier resulted in the lowest performance for all imputation methods and other classifiers. KNN achieved mean accuracy of 0.759 at low missing rate, slightly below both Simple (0.761) and the Complete dataset (0.762). Distribution on 100 iterations was within approximately [0.6 and 0.9] range showing a moderate peak around [0.72 - 0.76]. At high missing rates, KNN demonstrated the most significant performance decline - mean Accuracy dropped to 0.670. Distribution shifted dramatically leftward to a broad range within [0.55 and 0.80] and a tail within [0.5 and 0.55].

Combination with Neural Network resulted in similar to SVM performance - mean accuracy 0.878, underperforming both MICE (0.932) and Complete set results (0.924). At low missing rate, Accuracy distribution is within approximately [0.83 and 0.95] range. Similar to SVM results, at high missing rates performance decreased to mean accuracy of 0.824 and distribution shifted to the broader range within [0.7 and 0.9].

Simple Imputer impact on classification performance

In both missing rate scenarios Simpe Imputer was equal or superior to KNN on nearly all classifiers.
At low missing rates, Simple Imputer matched KNN with SVM (both 0.888). On Random Forest (0.761 vs 0.759) and Neural Network (0.8783 vs 0.8775) Simple Imputer slightly outperformed KNN. At high missing rates, it performed better than KNN on all classifiers. Interestingly, Simple Imputer demonstrated

the most stable performance in both missing rate scenarios in combination with Random Forest - low and high missing rates (0.761 to 0.759) respectively.

Summarized classification performance results are shown in Table 3.4, which presents test accuracy and F1 scores for all classifier-imputation method combinations across both low (10%) and high (25%) missing rates.

Classifier	**Imputation**	**Accuracy (Low 10%)**	**Accuracy (High 25%)**	**F1 Score (Low 10%)**	**F1 Score (High 25%)**
SVM	Complete Data	0.9269	0.9269	0.9469	0.9469
	MICE	0.9337	0.8833	0.9519	0.9139
	KNN	0.8875	0.8262	0.9184	0.8744
	Simple	0.8875	0.8351	0.9182	0.8815
Random Forest	Complete Data	0.7623	0.7623	0.8507	0.8507
	MICE	0.7945	0.8265	0.8682	0.8705
	KNN	0.7590	0.6697	0.8445	0.7432
	Simple	0.7606	0.7586	0.8485	0.8180
Neural Network	Complete Data	0.9236	0.9236	0.9443	0.9443
	MICE	0.9323	0.8883	0.9508	0.9174
	KNN	0.8775	0.8242	0.9109	0.8731
	Simple	0.8783	0.8322	0.9118	0.8805

Table 3.4 Test accuracy and F1 scores by classifier, imputation method, and missing rate

Overall, MICE-imputed datasets classification results outperformed KNN and Simple Imputer, in both low and high missing rate scenarios. Particularly, the strongest classification performance achieved was in combination with SVM and Neural Network classifiers at low missing scenario, resulting in test accuracy scores of 0.934 and 0.932 respectively and outperforming even Complete dataset. Simple Imputer consistently outperformed KNN in both scenarios, particularly at higher missing rates. However, unlike all other imputation methods, it showed stability with Random Forest classification with approximately 0.2% accuracy decline between missing levels. KNN imputation proved most sensitive to increased missingness, with performance dropping with Random Forest combination (-11.7%). All methods achieved higher accuracy with SVM and Neural Network compared to Random Forest, except for MICE: at high missing rates, MICE surprisingly improved Random Forest performance (0.827 vs complete data's 0.762). Similar results were achieved based on F1 Score, which can be seen in Appendix C.

A comparison of train and test performance metrics is provided in Appendix C. Results highlight that, even though train performance patterns were consistent with test, Random Forest demonstrated sufficient gap between train and test sets suggesting overfitting even on baseline complete dataset.

3.3 Feature Importance

The aim of this section is to assess the imputation and classification methods impact on feature importance preservation. As described in Section 2.2, we generated 25 variables grouped into three levels of predictive importance by weights assigned – 7 high important, 9 medium important and 9 low important variables. Though the specific weights varied with each of 100 iterations of simulation, the specific variables group remained constant. To assess how well the imputation-classification combination was able to preserve the true predictability level of variables we calculated the feature importances for each classification iteration as described in Section 2.6. In this section we evaluated and compared how accurately each imputation-classification combination was able to identify the relative importance of features in both missing rate scenarios. For assessment, we extracted the feature importance values produced by each classifier and sorted them in descending order. We then categorized them into top 7 features as highly important, the next 9 as medium and remaining 9 as low. Accuracy of the match with true importance groups was calculated as a percentage of features assigned correctly.

The matching accuracy percentages averaged based on all 100 iterations presented for low missing rate in Table 3.5 and for high missing rate scenario in Table 3.6. Each table outlines imputation-classification performance with percentages for each feature importance group along with the average results and ranking for convenience.

Classifier	Imputation	High Importance (%)	Medium Importance (%)	Low Importance (%)	Average (%)	Rank
SVM	MICE	85.29	86.7	74.00	81.92	1
	KNN	83.29	87.2	73.22	81.31	2
	Simple	82.57	86.5	71.67	80.31	3
Random Forest	MICE	63.57	65.3	46.78	58.42	7
	KNN	63.14	65.3	46.89	58.35	8
	Simple	63.57	58.8	42.78	54.54	9
Neural Network	MICE	70.14	77.6	56.44	68.27	4
	KNN	68.86	75.8	54.11	66.42	5
	Simple	68.57	74.3	53.11	65.42	6

Table 3.5 Average feature importance group matching accuracy (%) based on 100 iterations for the low missing rate scenario (10%). Results are presented by imputer–classifier combination, with separate percentages for high, medium, and low importance groups, the overall average accuracy, and the rank based on average performance.

Classifier-wise at low missing rate, SVM was able to preserve the most accurate feature importance for all imputation methods at between 80.31% and 81.92% on average, as seen in Table 3.5. Particularly, the highest accuracy was achieved for high and medium importance features at between 82.57% and 87.2%. Neural Network preserved the importances less accurately ranging from 65.42% to 68.27% on average, particularly better with matching true medium importance features at 74.3 - 77.6 % accuracy but struggled to correctly identify low importance features. Random Forest was the weakest, ranging only from 54.54% to 58.42%, so about half of the features were not accurately identified.

Imputation-wise, MICE-imputed datasets demonstrated on average better ability to preserve accurate feature importance comparing with other methods across all classifiers. Best results achieved in combination with SVM classifier. KNN on average slightly outperformed the Simple Imputer in pair with SVM and Neural Network by within 1% accuracy improvement, and more sufficiently with Random Forest by around 4% accuracy improvement.

Classifier	**Imputation**	**High Importance (%)**	**Medium Importance (%)**	**Low Importance (%)**	**Average (%)**	**Rank**
SVM	MICE	63.57	70.6	49.67	61.46	3
	KNN	72.43	80.1	62.11	71.81	2
	Simple	77.14	81.3	64.67	74.42	1
Random Forest	MICE	61.00	65.8	45.67	57.54	6
	KNN	54.43	64.2	43.67	54.46	8
	Simple	56.71	55.9	36.00	49.23	9
Neural Network	MICE	66.00	65.6	46.56	59.12	5
	KNN	64.57	65.9	47.56	59.19	4
	Simple	62.57	64.9	45.11	57.42	7

Table 3.6 Average feature importance group matching accuracy (%) based on 100 iterations for the high missing rate scenario (25%). Results are presented by imputer–classifier combination, with separate percentages for high, medium, and low importance groups, the overall average accuracy, and the rank based on average performance.

Classifier-wise at high missing rate, all imputer-classifier combinations demonstrated significant drop in performance. SVM still preserved the most accurate feature importance, though with reduced performance ranging from 61.46% to 74.42% as seen in Table 3.6. Medium features maintained the highest preservation accuracy at between 70.6% and 81.3%. Neural Network performance dropped to 57.42% to 59.19% on average, however showed consistency on all imputation methods. Random forest was least able to accurately retail feature importance within the range of 49.23% to 57.54% on average. Particularly weak performance was seen on low importance features in combination with Simple Imputer, where Random Forest correctly identified only 36%.

Imputation-wise, Simple Imputer unexpectedly performed best with SVM (74.42%), but worst with Random Forest (49.23%). KNN performed relatively well with SVM (71.81%) and weaker with Neural Network (59.19%). Notably, MICE ability to preserve feature importance dropped significantly with SVM to 61.46% from its top ranking at low missing rates. On Random Forest, the shift of missing rate almost did not affect MICE performance with only the difference in around 1% of accuracy on average. With Neural Network, MICE demonstrated also significant drop in performance from 68.27% on low to 59.12% on high missingness.

To summarize, at low missing rate, SVM was able to preserve feature importance relatively the best comparing to other classifiers with average accuracy 80-82%. Random Forest struggled significantly (54-58%). MICE-imputed datasets were able to retain feature importance best comparing with other imputation methods, particularly well combining with SVM (81.92%) and Neural Network (68.27%) classifiers. However, when shifting to higher missing rates MICE with SVM combination dropped by 20.46% in accuracy. Overall, SVM retained top performance with higher missing rates, even though imputation level performance dropped. Combination of Random Forest - Simple Imputer proved to be the weakest, dropped to the lowest overall accuracy (49.23%), with particularly poor ability to retain low feature importances (36%).

4 Discussion

The impact of missing data imputation methods in classification problems was investigated across multiple scenarios through a simulation study. Synthetic datasets were generated with mixed data types, three predefined levels of predictive importance, and a Bernoulli target variable. Missingness was induced using a balanced combination of MCAR and MAR mechanisms at low (10%) and high (10%) rates, applied separately to training and test partitioned subsets (70/30 split). Three imputation methods (MICE, KNN, Simple Imputer) were applied to each dataset and evaluated using RMSE and MAE. The imputed datasets were then trained on three classification methods (SVM, Random Forest, Neural Network) and assessed by Accuracy and F1 Score. Each scenario was repeated over 100 iterations to obtain the distributions and stability of results. In this chapter we discuss these findings in the context of existing literature, focusing on imputation quality, imputer-classifier interactions, the impact of missing rates, missing mechanisms and feature importance preservation.

Imputation evaluation resulted in Simple Imputer outperforming KNN and MICE by RMSE and MAE, with stable results in both missing rate scenarios. KNN followed closely, and MICE had the highest errors and showed the greatest sensitivity to increased missingness. At first glance, these results differ from Sasu et al. (2025), who found that machine learning methods, particularly KNN consistently outperformed other methods on all missingness mechanisms. However, Shadbahr et al. (2023) note that no single method performs best across all contexts, which aligns with our own results. More importantly, they caution that RMSE and MAE focus on per-value accuracy and can underestimate a method's ability to restore the multivariate structure of the data. This was evident in our work as well - despite ranking worst on RMSE and MAE, MICE demonstrated outstanding performance on downstream classification performance. The observed error distributions over 100 iterations also provided insight into methods stability of performance. Simple Imputer and KNN both showed narrow spreads at each missingness level, suggesting consistent performance, while MICE displayed greater variability. This variability, however, does not necessarily mean weakness and classification results highlighted that. These findings echo El Badisy et al.'s (2024) observation that methods optimised for predictive accuracy are not always those that minimise statistical bias, and that different evaluation criteria can lead to different "best" methods. This highlights that sample-wise error metrics and downstream predictive performance do not necessarily align.

When comparing the effect of missingness mechanism by variable types, the effect on continuous and discrete variables was minimal, with RMSE differences of about 0.01 for all imputers. Categorical variables consistently performed better under MAR, most notably for Simple Imputer (0.54 vs. 0.64). Though RMSE calculations on binary categorical values are unlikely informative. Nonetheless, under MAR, categorical variables had narrower distributions for all methods, indicating greater consistency when missingness followed a systematic pattern.

The combination of imputation method and classifier produced several interesting and sometimes surprising findings, many of which are aligned with the recent literature. Our results generally confirm Shadbahr et al. (2023) claim that missingness rate has stronger influence on classification performance than imputation or classification method. But some combinations appeared quite resilient to missing rate change. For instance, MICE with SVM and Neural Network achieved the highest accuracies at low missingness - exceeding even the complete dataset benchmark. Yet performance significantly dropped at 25% missingness with accuracy distributions widening over 100 iterations. By contrast, Simple Imputer with Random Forest retained almost identical performance at both missing rates (0.761 vs. 0.759).

Some results reflected the “counterintuitive relationship” described by Shadbahr et al. (2023), where lower-quality imputations by RMSE/MAE nonetheless supported competitive or even superior classification performance. The clearest example was MICE with Random Forest, where the high-missingness mean accuracy (0.827) not only surpassed the low-missingness result (0.795) but also outperformed the complete dataset benchmark (0.762). Shadbahr et al. mentioned that imputed values may occasionally serve as a regularization for machine learning algorithms helping overcome overfitting and improving predictability. Possibly MICE higher variability in errors could serve as a regularization for fairly overfitting in most scenarios Random Forest, which improved its performance.

Sasu et al. (2025) claimed that decision tree models improved substantially when paired with KNN imputations, yet in our work KNN–Random Forest (Random Forest is an ensemble of many decision trees) was the weakest combination. When missing rate shifted to high, accuracy dropped from 0.759 at low missingness to 0.670. The error distributions widened sufficiently and shifted left (to around 0.55), suggesting a substantial loss in stability. Possible explanation of this result is that even with Gower distances, KNN’s neighbour-averaging tendency could smooth class boundaries and reduce variability so much so that disrupted optimal tree splits. The Simple Imputer-Random Forest combination, in contrast, remained stable across missingness scenarios, even though its imputation strategy is directly averaging. Possible interpretation could be in a rather uniform nature of Simple Imputer’s “averaging” properties that introduced more inform shifts in data that disrupted tree splits to a les extent than KNN did.

Beyond classification accuracy and stability, an equally important question is how different imputation–classifier combinations affect the interpretability of the resulting models. Although previous studies (Shadbahr et al., 2023; Sasu et al., 2025) highlighted that imputation quality can influence model interpretability and warned that poorly imputed datasets may assign misleading feature importances, none have directly measured this effect. This study addressed this gap by evaluating how accurately each imputer–classifier combination preserved the true ranking of variable importance. The results showed notable differences from other performance metrics: Simple Imputer, which achieved the best RMSE, demonstrated a strong bias in feature importance preservation, particularly with Random Forest, where low-importance features were frequently overestimated. In contrast, MICE, which had the highest RMSE, retained the most accurate importance structure in several scenarios, especially when combined with SVM and Neural Network. These results confirm that imputers that had better error estimates performance are not necessarily best for interpretability.

Additionally, we explored how varying weights assigned to predictors influenced the outcomes of both imputation and classification quality. The results in Appendix A revealed a clear and intuitive, yet important pattern: when predictive signal weakened by reducing the weight ranges for all predictor groups by their importance – classification performance has sufficiently declined for all scenarios and combinations. Conversely, when the predictive signal was increased – classification performance substantially improved consistently with all scenarios. Most importantly, weights adjustments had minimal influence on RMSE and MAE and the ranking of imputers remained unchanged. This highlights a key limitation of evaluating the imputation quality by per-value error metrics – they measure how close imputed values are to true ones but not whether they support the model's ability to make reliable predictions. In other words, when predictive signal in data changes, two datasets can have similar RMSE for the same imputer but produce very different classification results. This means that evaluating imputed data quality solely based on errors can be misleading, it should be assessed in the context of how well imputed data supports the specific predictive task.

Limitations and future work

First, even though synthetic dataset allowed for controlled simulation study, its advantages could also serve as constraints on generalisability. Real-world datasets often have more complex non-linear relationships and thus imputation and classification methods could demonstrate even more complex and nuanced outcomes. Assumption is that Simple Imputer would not perform that well on RMSE with non-linear relationships. Secondly, in our study a combination of MCAR and MAR was explored with fixed and relatively not extreme rate of missingness. A more challenging MNAR scenario with wider range of missing rates could be studied to check missing data in classification problems from another perspective. Thirdly, we aimed to explore the representative selection of popular imputation methods including single, (Simple Imputer), multiple imputations (MICE) and machine learning method (KNN). However, the literature mentions many more advanced machine learning methods, such as GAIN and VAE and "empowered" mice methods such as miceforest that would extent multiple imputation along with ML methods experiments.

Another important limitation was computational constraints related to proper execution of mice-classification estimation. Although best-performing m and maxit parameters were selected and applied to train - test pairs at both missingness levels, a fully correct application would require training each classifier on every imputed dataset separately. With the chosen parameters, this would mean generating 120 imputed datasets (30 imputations × 4 train/test sets) per simulation, leading to 36,000 classifier trainings across 100 simulations - computationally infeasible in this study. Therefore, we compromised by producing pooled estimates from 30 imputations with 20 iterations per MICE run, aggregated over 100 simulations. Even with pooled mice estimates the run time of 100 simulations took around 14 hours with automated tuning of each classification methods taking large proportion of time. However, the computational cost was achieved with not automating the imputation methods tuning. We tuned imputers using just one generated dataset and selected best parameters based on that. While each iteration generated new datasets, even with same controlled parameters the values were different, and imputation parameters could also be more case specific. With less computational constraints, a more appropriate imputation tuning could be achieved. Another thing to consider is the choice of metrics for imputation quality evaluation. While RMSE and MAE are valid and popular methods, they have limitations on assessing important imputation aspects such as ability to preserve distributional similarity, feature importance and relationships, also categorical accuracy. Suggested alternatives include Wasserstein and Kullback–Leible distances.

Conclusions and Practical recommendations

This study demonstrates that classification performance trained on imputed datasets is influenced by multiple factors beyond the choice of the best imputer or classifier alone. Over 100 simulations results, the missing rate was identified as indeed influential factor: accuracy and stability of results almost always declined as missing rate increased from 10% to 25% even for imputer-classification that performed exceptionally well when missing rate was low. However, some combinations demonstrated resilience, for instance Simple Imputer with Random Forest. This leads to a suggestion that the compatibility between imputers and classifiers is more important rather than selecting the most advanced method in isolation. MICE combined with SVM or Neural Network delivered the highest accuracies when missing rate was low but dropped the performance when missing rate increased. While MICE with Random Forest when missing rate increased the performance even improved comparing to low-rate results. By contrast, with KNN Random Forest proved very unstable with missingness increased.

Results also revealed that imputation accuracy as measured by RMSE and MAE seem not a reliable predictor of downstream classification performance. The rankings of MICE and Simple Imputer completely reversed with the top performer in one case becoming outsider in the other when comparing their performance on imputation quality based on RMSE and classification accuracy.

Interpretability of results was another important factor in classification tasks. Ability to better preserve the true feature importance leading to a more accurate interpretation of classification predictions had little to do with top estimated imputation performance. This was proved again by Simple Imputer and MICE performance comparison when MICE together with SVM was able to better retain the true relationships of predictors to the target. Moreover, the strength of the predictive signal in data appeared to be an influential factor on classification performance and less affecting the imputation quality. Therefore, this reinforces the notion that imputation evaluation should not be limited to solely error metrics since similar RMSE/MAE results may lead to dramatically different classification performance.

Above all, when selecting the best imputation-classifier methods combination, researchers should account for specific priorities, balancing predictive accuracy, stability and interpretability of results, while not relying solely on error metrics for quality evaluation. There is likely no single universal "one best for all" imputer-classifier combination, but given specific research goals and circumstances appropriate combinations can achieve desired performance and occasionally even exceed the expected results.

References

Code submitted with ReadMe file to SharePoint folder **-** Final Code submission.

Albon, C. (2018). *Machine Learning with Python Cookbook*. ‘O’Reilly Media, Inc.’, pp.251-336.

Aurélien Géron (2022). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. ‘O’Reilly Media, Inc.’, pp.164-320.

Bennett, D.A. (2001). How can I deal with missing data in my study? *Australian and New Zealand Journal of Public Health*, 25(5), pp.464–469. doi:https://doi.org/10.1111/j.1467-842x.2001.tb00294.x.

Enders, C.K. (2010). *Applied Missing Data Analysis*. Guilford Press.

Gabriel-Vasilică Sasu, Bogdan-Iulian Ciubotaru, Goga, N. and Andrei Vasilățeanu (2025). Addressing Missing Data Challenges in Geriatric Health Monitoring: A Study of Statistical and Machine Learning Imputation Methods. *Sensors*, 25(3), doi:https://doi.org/10.3390/s25030614.

Gower, J.C. (1971). A General Coefficient of Similarity and Some of Its Properties. *Biometrics*, 27(4), p.857. doi:https://doi.org/10.2307/2528823.

Imad El Badisy, Graffeo, N., Khalis, M. and Giorgi, R. (2024). Multi-metric comparison of machine learning imputation methods with application to breast cancer survival. *BMC Medical Research Methodology*, 24(1). doi:https://doi.org/10.1186/s12874-024-02305-3.

Kowarik, A. and Templ, M. (2016). Imputation with the R Package VIM. *Journal of Statistical Software*, 74(7). doi:https://doi.org/10.18637/jss.v074.i07.

Little, R. J. A. and Rubin, D. B. (1987). *Statistical Analysis with Missing Data.* John Wiley & Sons, New York.

Numpy.org. (2024). *Random sampling (numpy.random) — NumPy v2.0 Manual*. [online] Available at: https://numpy.org/doc/2.0/reference/random/index.html

Peng, C.-Y. J., Harwell, M., Liou, S.-M., & Ehman, L. H. (2006). *Advances in missing data methods and implications for educational research*. In S. Sawilowsky (Ed.), *Real data analysis* (pp. 31-78). Greenwich, CT: Information Age Publishing, Inc.

Rubin, D.B. (1976). Inference and missing data. *Biometrika*, 63(3), pp.581–592. doi:https://doi.org/10.1093/biomet/63.3.581.

Schafer, J.L. (1999). Multiple imputation: a primer. *Statistical Methods in Medical Research*, 8(1), pp.3–15. doi:https://doi.org/10.1191/096228099671525676.

Schouten, R.M. (2021). *Welcome to pyampute’s documentation! — pyampute 0.0.1 documentation*. [online] Github.io. Available at: https://rianneschouten.github.io/pyampute/build/html/index.html

Schouten, R.M., Lugtig, P. and Vink, G. (2018). Generating missing values for simulation purposes: a multivariate amputation procedure. *Journal of Statistical Computation and Simulation*, 88(15), pp.2909–2930. doi:https://doi.org/10.1080/00949655.2018.1491577.

Scikit-Learn (2025). *sklearn.ensemble.RandomForestClassifier — scikit-learn 0.20.3 Documentation*. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

Scikit-learn (2019). *sklearn.preprocessing.OneHotEncoder — scikit-learn 0.22 documentation*. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html.

Scikit-learn (2010). *sklearn.neural_network.MLPClassifier — scikit-learn 0.20.3 Documentation*. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html.

Scikit-learn (2018). *sklearn.model_selection.train_test_split — scikit-learn 0.20.3 documentation*. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html.

Scikit-learn (2019). *sklearn.svm.SVC — scikit-learn 0.22 Documentation*. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

Scikit-learn (2019). *StandardScaler*. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

Stef Van Buuren and Groothuis-Oudshoorn, K. (2011). mice: Multivariate Imputation by Chained Equations inR. *Journal of Statistical Software*, [online] 45(3). doi:https://doi.org/10.18637/jss.v045.i03.

Stef Van Buuren (2012). *Flexible imputation of missing data*. Boca Raton, Fl: Crc Press.

Tolou Shadbahr, Roberts, M., Stanczuk, J., Gilbey, J., Teare, P., Dittmer, S., Thorpe, M., Ramon Viñas Torné, Sala, E., Pietro Lió, Patel, M., Preller, J., Selby, I., Breger, A., Weir-McCall, J.R., Effrossyni Gkrania-Klotsas, Korhonen, A., Jefferson, E., Langs, G. and Yang, G. (2023). The impact of imputation quality on machine learning classifiers for datasets with missing values. *Communications Medicine*, 3(1). doi:https://doi.org/10.1038/s43856-023-00356-z.

Variance Inflation Factor . [online] Available at: https://www.statsmodels.org/dev/generated/statsmodels.stats.outliers_influence.variance_inflation_factor.html.

Appendix A

Synthetic Dataset generation

A.1 Data distribution of generated dataset for all variables

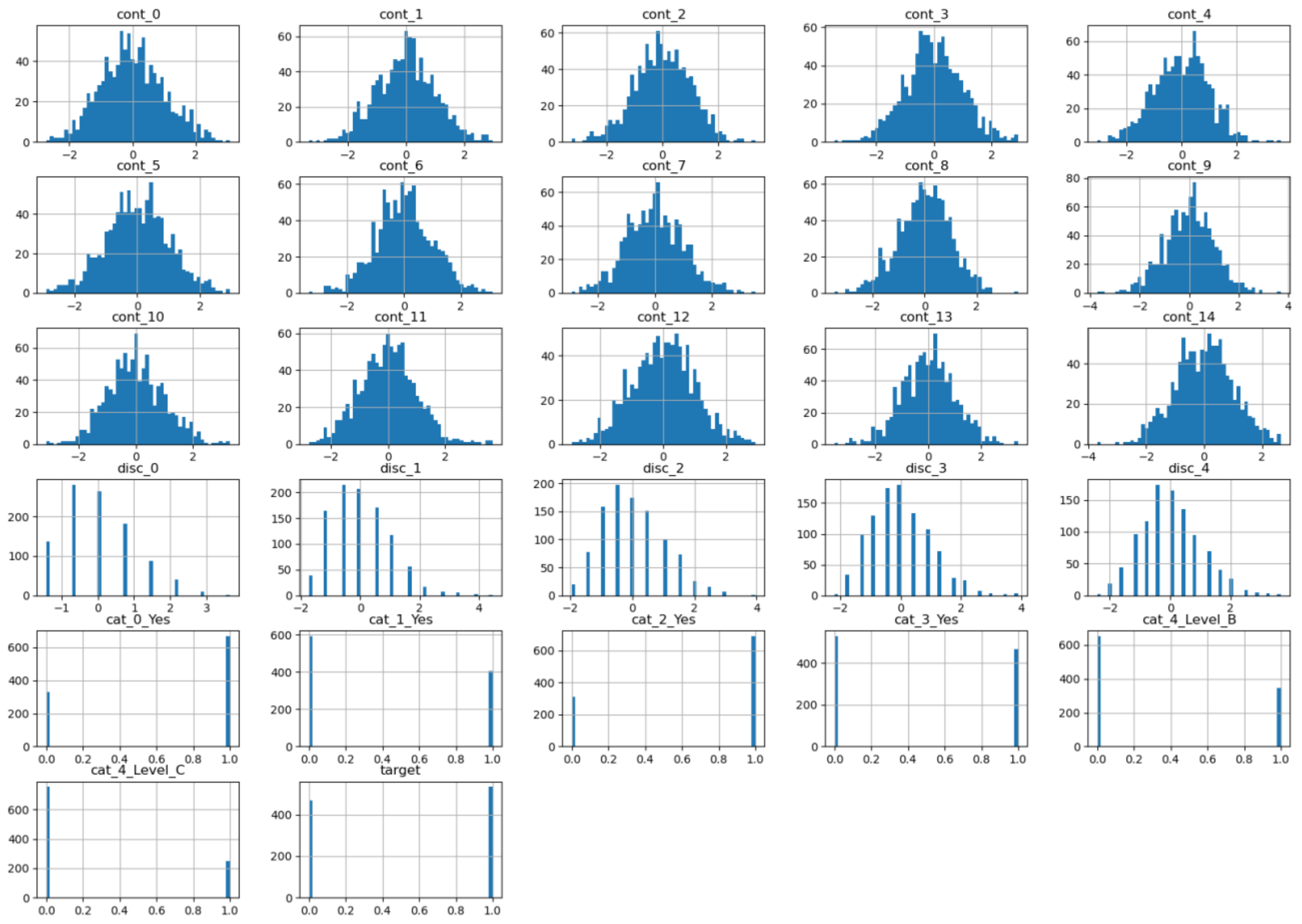


Figure A.1. Distribution plots for all variables in the synthetic dataset. Continuous variables display normal distributions with no heavy skewness or outliers. Discrete and categorical variables show balanced or reasonable frequency distributions across levels. The target variable has balanced classes.

A.2 Impact of Different Weight Configurations on Classification Performance

In this configuration, lower weights were assigned to all predictor groups: high-importance variables - range (0.50 - 0.70 , medium-importance variables - (0.20 - 0.40), low-importance variables - (0.05 - 0.15). The model intercept was adjusted to -1 to maintain balanced class proportions. Reduced weights substantially weakened the signal in the data, leading to sufficient decline in both classification accuracy and F1 score. Results for low (10%) and high (25%) missingness scenarios are shown in Table A.2.a. This illustrates the strong dependency of imputation–classification outcomes on the underlying signal strength of predictors.

Classifier	Imputation	Accuracy (Low 10%)	Accuracy (High 25%)	F1 Score (Low 10%)	F1 Score (High 25%)
SVM	Complete Data	0.7211	0.7211	0.7035	0.7035
	MICE	0.7389	0.7356	0.7223	0.7298
	KNN	0.7167	0.6689	0.6937	0.6339
	Simple	0.7211	0.6922	0.6996	0.6636
Random Forest	Complete Data	0.6789	0.6789	0.6217	0.6217
	MICE	0.6911	0.7078	0.6498	0.7060
	KNN	0.6278	0.6133	0.5789	0.5827
	Simple	0.6611	0.6378	0.6105	0.6170
Neural Network	Complete Data	0.7233	0.7233	0.7085	0.7085
	MICE	0.7211	0.7011	0.7025	0.6971
	KNN	0.7089	0.6600	0.6922	0.6310
	Simple	0.6989	0.6800	0.6868	0.6392

Table A.2.a Classification accuracy and F1 scores resulted with generated datasets with reduced predictor weights.

In another configuration, higher weights were assigned: high-importance variables - range (8.0 - 12.0), medium-importance variables - (2.0 - 4.0), low-importance variables - (0.10 - 0.50). The model intercept was adjusted to -15. In this case, higher weights substantially increased the signal in the data, resulting in sufficient improvement in both classification accuracy and F1 score. Results for low (10%) and high (25%) missingness scenarios with increased weights are shown in Table A.2.b.

Classifier	Imputation	Accuracy (Low 10%)	Accuracy (High 25%)	F1 Score (Low 10%)	F1 Score (High 25%)
SVM	Complete Data	0.9281	0.9281	0.9473	0.9473
	MICE	0.9344	0.8849	0.9520	0.9142
	KNN	0.8885	0.8262	0.9179	0.8717
	Simple	0.8886	0.8356	0.9180	0.8798
Random Forest	Complete Data	0.7650	0.7650	0.8488	0.8488
	MICE	0.7967	0.8295	0.8673	0.8716

Classifier	**Imputation**	**Accuracy (Low 10%)**	**Accuracy (High 25%)**	**F1 Score (Low 10%)**	**F1 Score (High 25%)**
	KNN	0.7595	0.6713	0.8400	0.7412
	Simple	0.7625	0.7595	0.8456	0.8165
Neural Network	Complete Data	0.9249	0.9249	0.9448	0.9448
	MICE	0.9329	0.8891	0.9508	0.9172
	KNN	0.8783	0.8244	0.9104	0.8707
	Simple	0.8794	0.8330	0.9115	0.8789

Table A.2.b Classification accuracy and F1 scores resulted with generated datasets with increased predictor weights.

However, the change of weights either decreasing or increasing apparently had limited impact on the imputation quality measured by RMSE and MAE. As seen in Table Table A.2.c with imputation results with lower weights and Table A.2.d with imputation results with higher weights, in both instances, the ranking of imputers performance remained as we observed before.

Dataset	**Method**	**RMSE (Low 10%)**	**RMSE (High 25%)**	**RMSE % Change**	**MAE (Low 10%)**	**MAE (High 25%)**	**MAE % Change**
Train	KNN	0.9130	0.9212	0.90	0.6858	0.6936	1.14
	MICE	0.9198	0.9640	4.81	0.6923	0.7264	4.93
	SIMPLE	0.8995	0.9031	0.40	0.6665	0.6702	0.56
Test	KNN	0.9181	0.9271	0.98	0.6973	0.6964	-0.13
	MICE	0.9487	1.0028	5.70	0.7115	0.7596	6.76
	SIMPLE	0.8973	0.9044	0.79	0.6718	0.6748	0.45

Table A.2.c Imputation evaluation by RMSE and MAE of datasets generated with lower weights run on 3 iterations.

Dataset	**Method**	**RMSE (Low 10%)**	**RMSE (High 25%)**	**RMSE % Change**	**MAE (Low 10%)**	**MAE (High 25%)**	**MAE % Change**
Train	KNN	0.9137	0.9184	0.52	0.6842	0.6877	0.51
	MICE	0.9189	0.9552	3.95	0.6869	0.7161	4.24
	SIMPLE	0.9048	0.9045	-0.03	0.6686	0.6685	-0.02

Dataset	Method	RMSE (Low 10%)	RMSE (High 25%)	RMSE % Change	MAE (Low 10%)	MAE (High 25%)	MAE % Change
Test	KNN	0.9229	0.9241	0.13	0.6906	0.6942	0.51
	MICE	0.9403	1.0050	6.88	0.7045	0.7594	7.80
	SIMPLE	0.9026	0.9017	-0.09	0.6660	0.6663	0.04

Table A.2.d Imputation evaluation by RMSE and MAE of datasets generated with higher weights run on 3 iterations.

Appendix B

Imputation methods

B.1 MICE diagnostics plots

B.1.a Trace plots for discrete and categorical data types

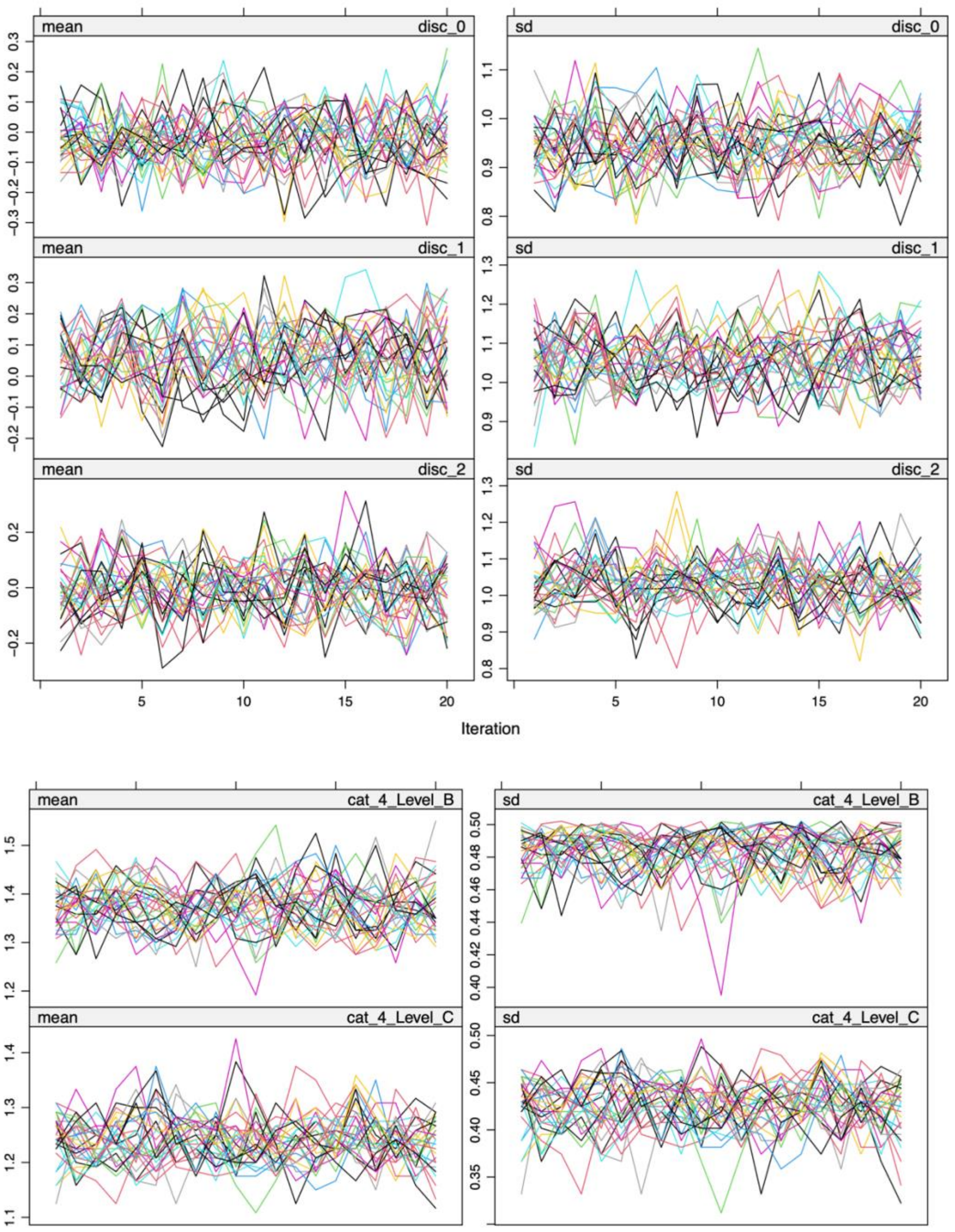


Figure B.1.a. Trace plots for representative discrete and categorical variables across 20 iterations of MICE imputation, showing mean (left panels) and standard deviation (right panels) trajectories for each imputation chain. Chains are well-mixed and fluctuate randomly around stable values with no visible trends, confirming convergence by iteration 20.

B.1.b Density m Low

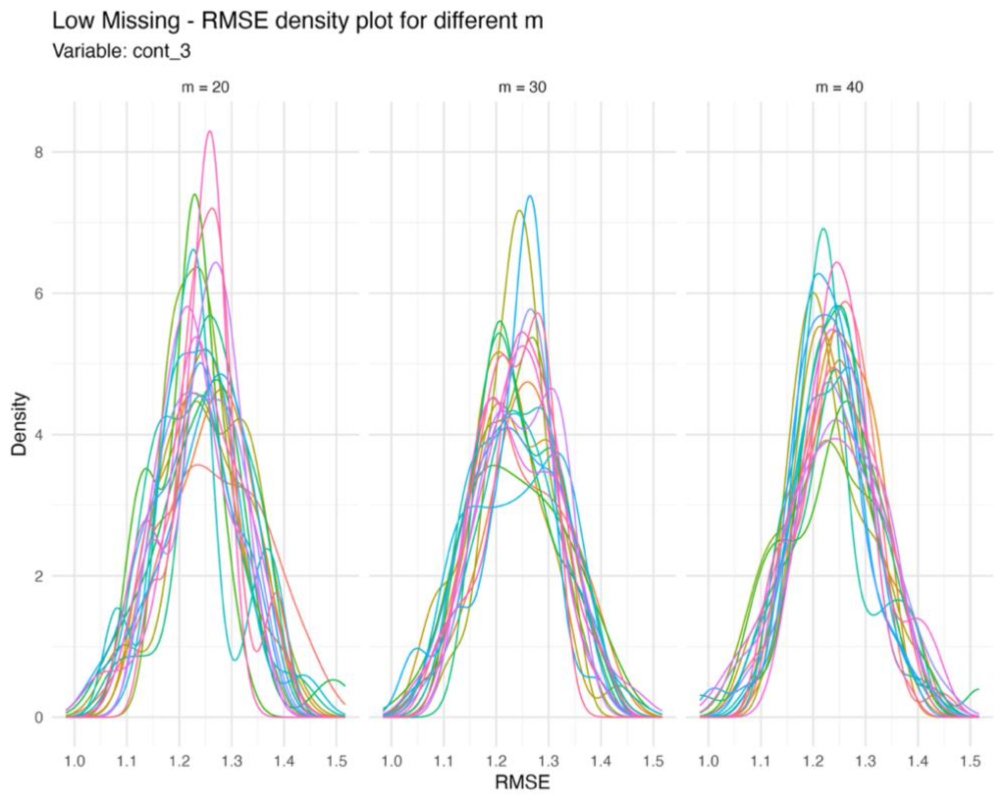


Figure B.1.b. RMSE density plots for variable cont_3 across 20 simulations under low missingness (10%), for candidate numbers of imputations m = 20, 30, and 40. Each coloured line represents an independent simulation. As with the high-missingness case, RMSE variability decreased with increasing m with m = 30. Though with m = 40 there seemed to be better stability, we had to take into account computational cost and therefore our choice was m=30 for low missing scenario as well.

B.1.c Strip plots for MICE post-imputation diagnostics

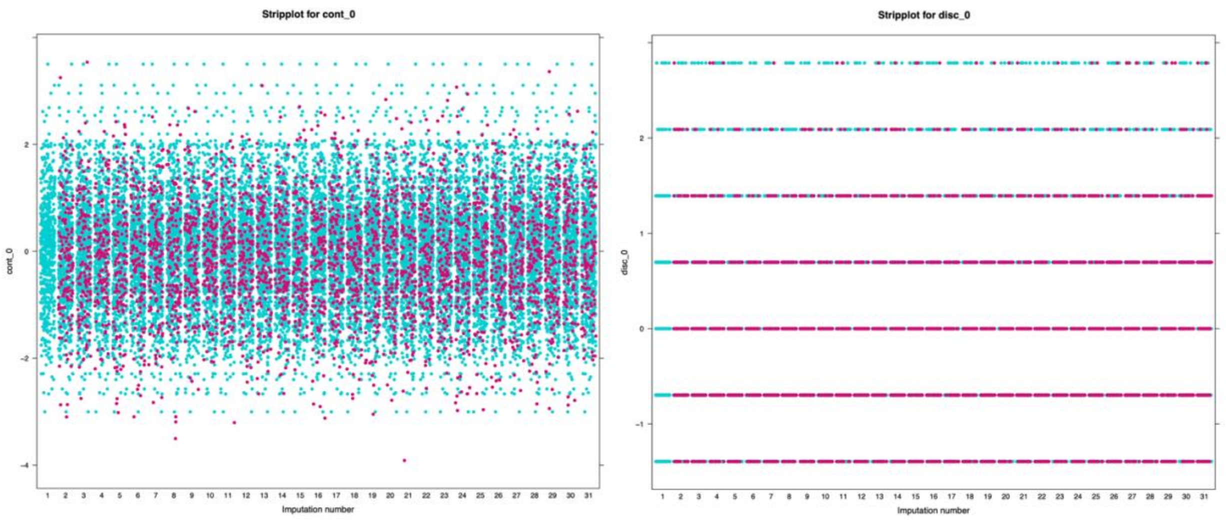
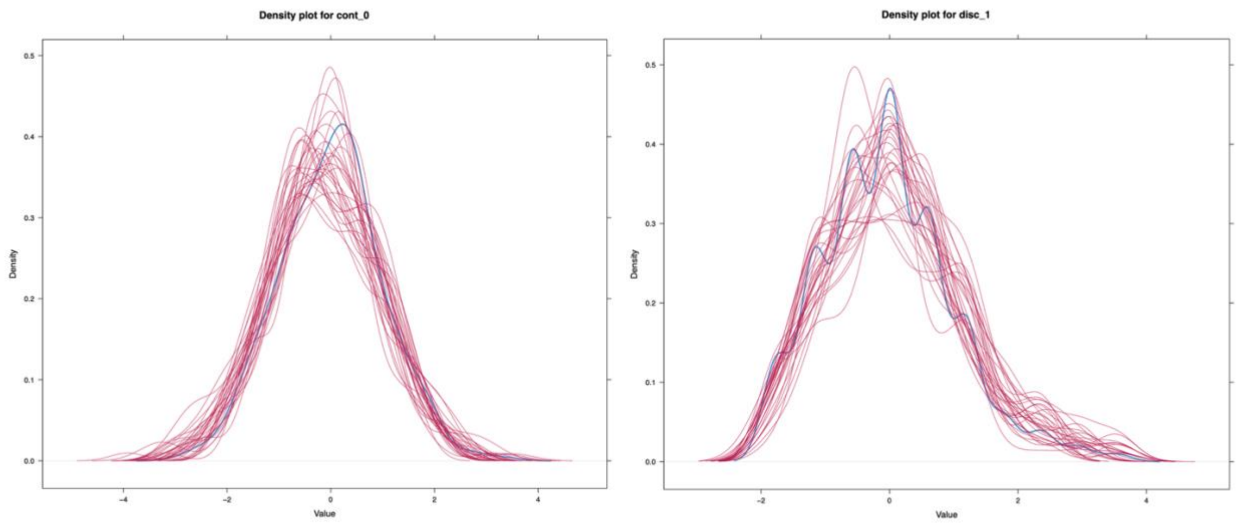


Figure B.1.c – Strip plots for continuous (left) and discrete (right) variables.Observed values (turquoise) and imputed values (pink) are shown across 30 imputations. For the continuous variable (cont_0), imputed values align closely with the observed distribution, with no evidence of systematic bias or

extreme-value clustering. For the discrete variable (disc_0), imputed values overlap observed categories proportionally, preserving the original frequency structure.

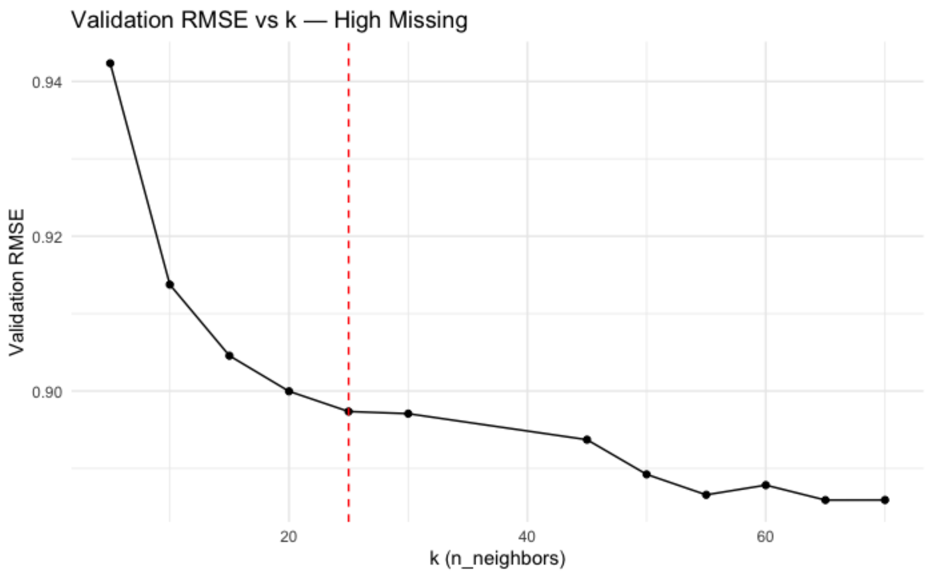
B.1.d Density plots for imputed vs. observed values



B.1.d Density plots for imputed vs. observed values (continuous and discrete). Density plots illustrate the distribution of imputed values (red lines) over observed values (blue line) for a representative continuous variable (cont_0, left) and discrete variable (disc_1, right). The close alignment between imputed and observed distributions indicates that MICE preserved the original data structure without introducing unrealistic variability.

B2 KNN diagnostics

B.2.a KNN high dataset k



B.2.a Elbow plot of RMSE vs. k values for KNN imputer (high-missingness dataset). The plot shows validation RMSE for candidate values of k (number of neighbors) in the KNN imputer, based on the high-missingness dataset. RMSE decreases sharply as k increases from small values, with no substantial gain beyond k =25 (red dashed line). This point represents a balanced choice with respect to the bias–variance trade-off, as larger k values had minimal further improvement but increasing smoothing of variation.

Appendix C

Results

C.1 Imputation:

C.1.a Distribution of Imputation performance based on RMSE stratified by methods, variable type and missing mechanism MAR and MCAR

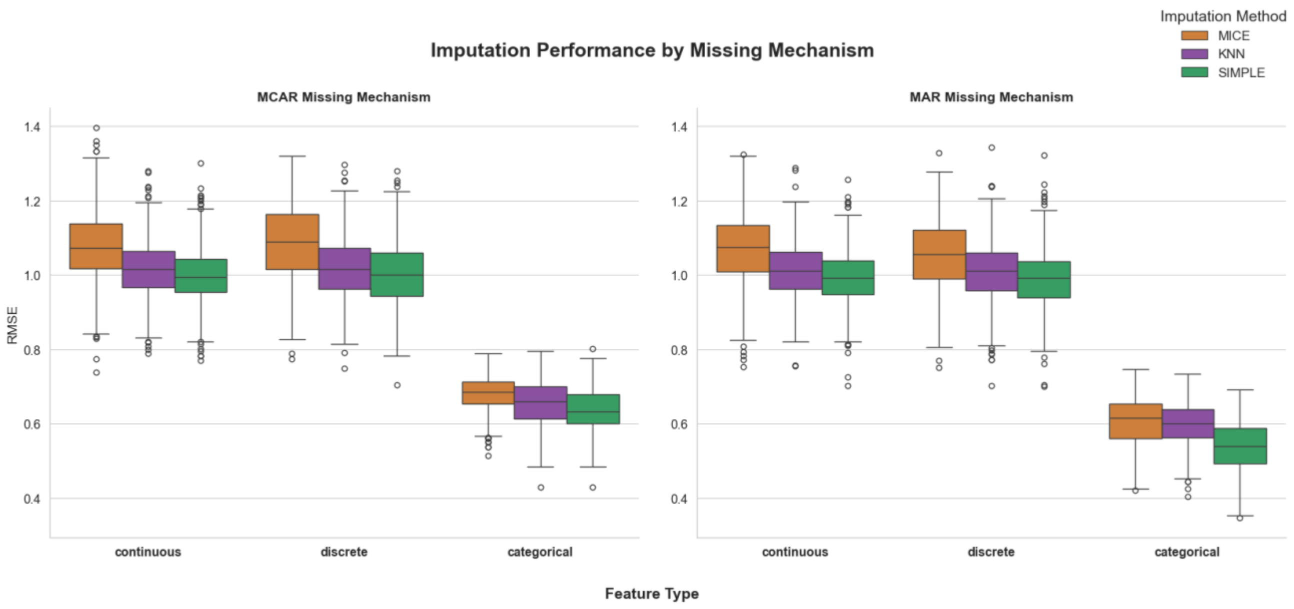


Figure C.1.a: Distribution of Imputation performance based on RMSE stratified by methods, variable type and missing mechanism MAR and MCAR. MICE exhibited the widest spread for numerical variables in both mechanisms, except for categorical variables under MCAR. Under MAR generally narrower RMSE distributions than under MCAR, particularly for categorical variables, suggesting greater consistency in imputation accuracy when missingness was conditionally dependent on observed data.

C.1.b Distribution of imputation performance by feature type and missing rate (RMSE)

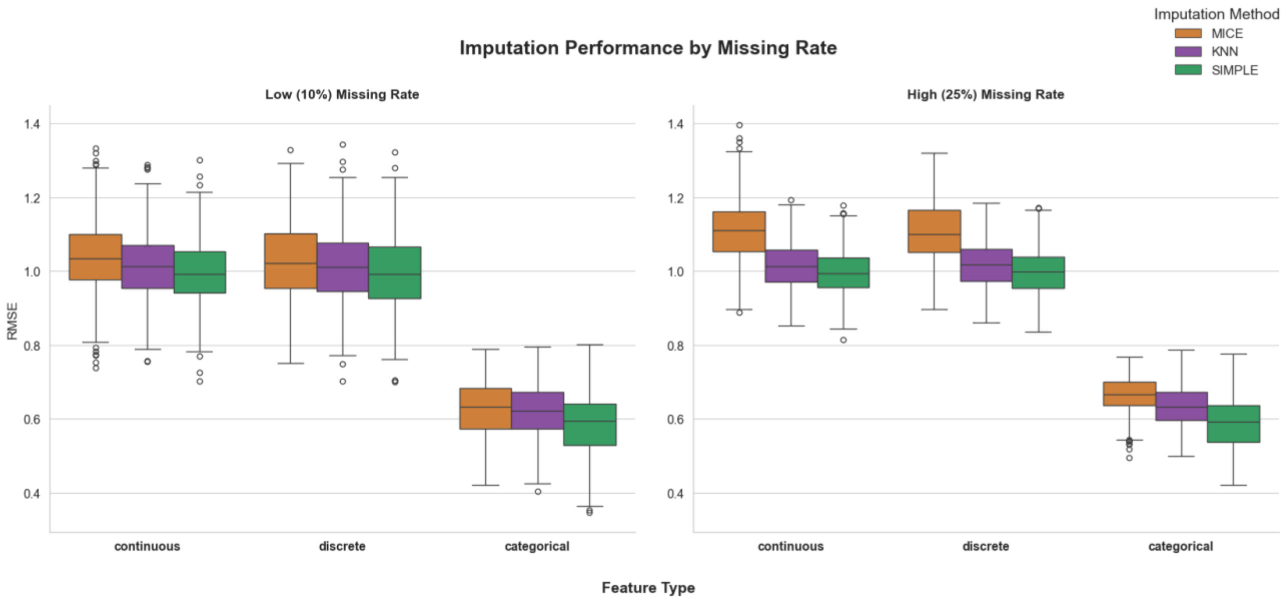


Figure C.1.b: Distribution of Imputation performance based on RMSE stratified by methods, variable type (10%, left) and high (25%, right) missing data rates. MICE consistently showed the widest spread in RMSE values for numerical variable types, with performance decreasing as missingness increased. All imputers demonstrated more stable results (narrower distributions) with high missing rate.

C.2. Classification:

C.2.a Distribution of F1 Score by classifier, imputation method, and missing rate

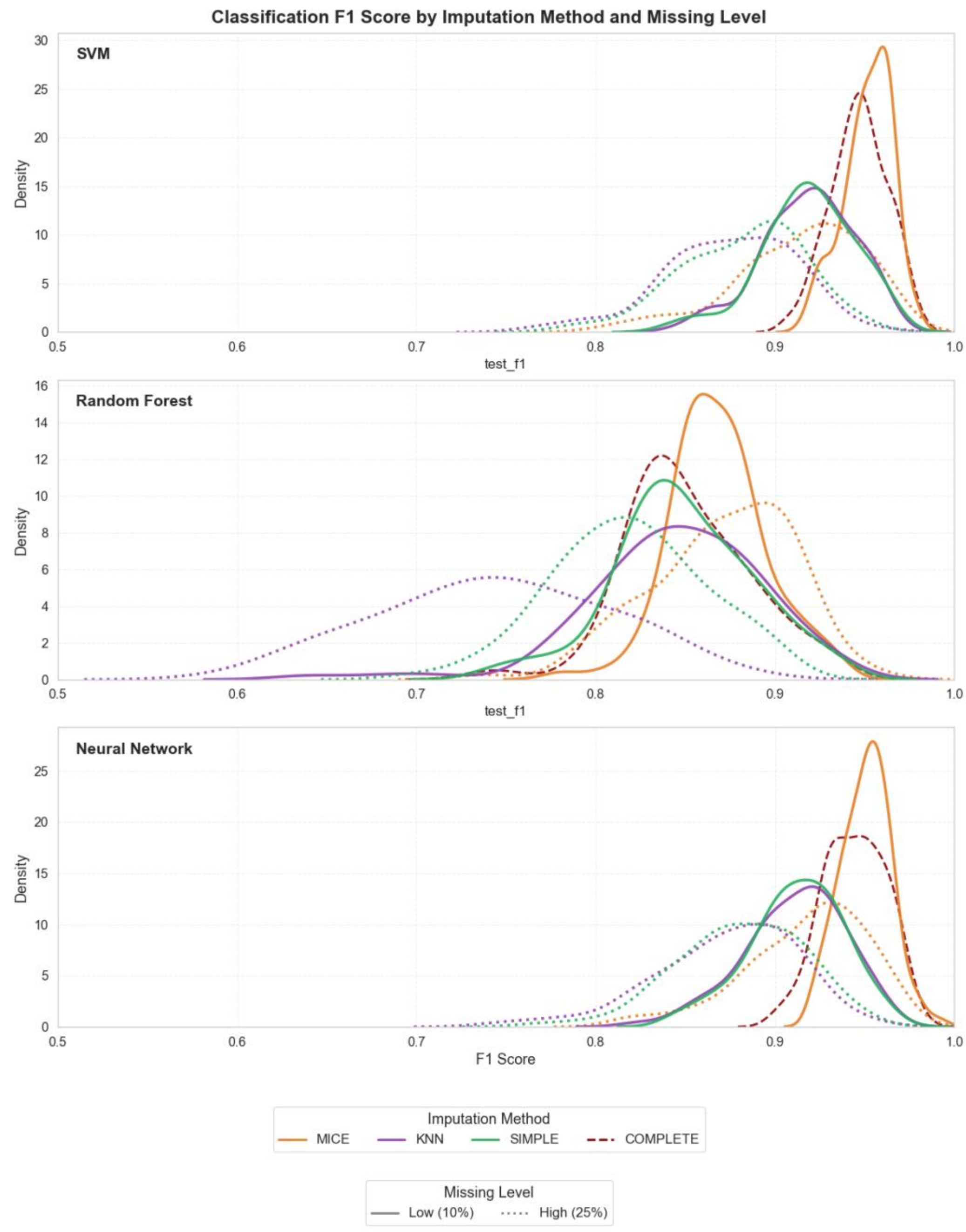


Figure C.2.a Distribution of F1 Score by classifier, imputation method, and missing rate. MICE generally achieved the highest F1 scores with most classifier combinations, particularly with SVM and Neural Network at low missing rates, overperforming even the complete dataset. Simple Imputer consistently outperformed KNN, especially at higher missingness, and demonstrated stability with Random Forest between missing levels. KNN showed the greatest decline due to increased missingness. MICE at high missing rates improved Random Forest performance.

C.2.a Train–test performance comparison by imputation method, classifier and missingness level

Table C.2.a reports train and test accuracy and F1 scores for each classifier - imputer combination on low (10%) and high (25%) missingness levels. Results are averaged over 100 simulation runs. Visible larger gaps between Random Forest train and test results, particularly with MICE and KNN imputers indicated overfitting.

Classifier	Imputation	Train Accuracy (Low 10%)	Train Accuracy (High 25%)	Train F1 (Low 10%)	Train F1 (High 25%)	Test Accuracy (Low 10%)	Test Accuracy (High 25%)	Test F1 (Low 10%)	Test F1 (High 25%)
SVM	Complete Data	0.9506	0.9506	0.9636	0.9636	0.9269	0.9269	0.9469	0.9469
	MICE	0.9616	0.9797	0.9718	0.9853	0.9337	0.8833	0.9519	0.9139
	KNN	0.9332	0.8946	0.9511	0.9241	0.8875	0.8262	0.9184	0.8744
	Simple	0.9099	0.8613	0.9342	0.8999	0.8875	0.8351	0.9182	0.8815
Random Forest	Complete Data	0.8032	0.8032	0.8761	0.8761	0.7623	0.7623	0.8507	0.8507
	MICE	0.8506	0.9243	0.9036	0.9432	0.7945	0.8265	0.8682	0.8705
	KNN	0.8480	0.8979	0.9015	0.9241	0.7590	0.6697	0.8445	0.7432
	Simple	0.8038	0.8284	0.8757	0.8698	0.7606	0.7586	0.8485	0.8180
Neural Network	Complete Data	0.9934	0.9934	0.9947	0.9947	0.9236	0.9236	0.9443	0.9443
	MICE	0.9963	0.9869	0.9970	0.9905	0.9323	0.8883	0.9508	0.9174
	KNN	0.9762	0.9124	0.9818	0.9377	0.8775	0.8242	0.9109	0.8731
	Simple	0.9495	0.8630	0.9614	0.9023	0.8783	0.8322	0.9118	0.8805

Table C.2.a: Comparison of train and test accuracy and F1 scores for each classifier–imputer combination under low (10%) and high (25%) missingness, averaged over 100 simulation runs.