VILNIUS UNIVERSITY

FACULTY OF MATHEMATICS AND INFORMATICS

DATA SCIENCE STUDY PROGRAMME

Master's thesis

Methods of Multiple Imputation

Daugialypiai praleistų reikšmių įrašymo metodai

Iveta Silkauskaitė

Supervisor : Assoc. Prof. Dr. Viktor Skorniakov

Vilnius
2025

Summary

Missing data is a major problem affecting decision-making processes and analysis results. The thesis reviews existing literature on multiple imputation applications and gives insights about the usage of MICE and other methods for data imputation in Python. The study consists of a simulation study and case study where real-world data was analysed and simulation results were confirmed. Imputation methods are compared by different data scenarios: data size, missingness rate, missingness type. The thesis could help practitioners while considering which data imputation method in which case to choose. The study showed that the most effective and universal method currently is MICE with Linear regression, Bayesian Ridge and Lasso estimators.

Keywords: Missing Values; Imputation; Multiple Imputation; MICE.

Santrauka

Praleistos reikšmės duomenyse yra didelė problema, kuri daro įtaką sprendimų priėmimui ir analizės rezultatams. Darbe apžvelgiama esama literatūra apie daugialypio praleistų reikšmių įrašymo metodų taikymą ir pateikiamos įžvalgos apie MICE ir kitų įrašymo metodų taikymą Python. Tyrimą sudaro simuliacinis tyrimas ir atvejo analizė, kurios metu buvo analizuojami realūs duomenys ir patvirtinami simuliacinio tyrimo rezultatai. Praleistų reikšmių įrašymo metodai lyginami pagal skirtingus duomenų scenarijus: duomenų dydį, praleistų reikšmių kiekį, tipą. Šis darbas gali padėti svarstant, kokį praleistų reikšmių įrašymo metodą, kokiu atveju pasirinkti. Tyrimas parodė, kad šiuo metu efektyviausias ir universaliausias metodas yra MICE su tiesinės regresijos, Bajeso Ridge ir Lasso įverčiais.

Raktiniai žodžiai: Praleistos reikšmės; Įrašymas; Daugialypis įrašymas; MICE.

Contents

1 Introduction

Missing data is a common issue across various fields. If not properly addressed, regardless of the dataset size and survey type, they affect the reliability of statistical analysis and can lead to biased or inconsistent results. Missingness can occur from nonresponse, data collection errors, data unavailability, privacy issues, etc, also leading to reduced statistical power, and inaccurate conclusions. Traditional methods, such as deletion or simple mean imputation, often result in distorted outcomes, as these methods fail to account for the structure of the missing data. The choice of imputation method can drastically influence the final results. It is also important to evaluate whether the missing attributes are significant enough to justify the time spent on data preprocessing and consider the types of variables and the quantity of missing data.

This research will explore multiple imputation methods and apply them to various datasets under different scenarios to demonstrate the impact of missing data. The study will include experiments when assumptions about missing data are violated—whether the data is Missing Completely at Random (MCAR), Missing at Random (MAR), or Missing Not at Random (MNAR).

Generative artificial intelligence (AI) models, such as *ChatGPT* and *Grammarly*, were used to improve the English language and correct writing mistakes.

2 Goal and Objectives

The thesis aims to investigate and compare the performance of different multiple imputation methods on various scenarios and datasets. To achieve this goal, I will:

- Conduct a literature review on multiple imputation methods and their applications with different data scenarios.
- Compare the performance and accuracy of different multiple imputation methods under various missing data scenarios (Missing Completely at Random (MCAR), Missing at Random (MAR), and Missing Not at Random (MNAR))
- Perform experiments on both simulated and real-world data to assess the practical impact and compare processing time and accuracy with different data types.
- Provide practical guidelines for selecting and implementing suitable methods based on the characteristics of the data.

By accomplishing these objectives, the thesis aims to offer valuable insights for researchers and practitioners in applying multiple imputation techniques effectively.

3 Literature review

Multiple imputation (MI) was first introduced by Donald B. Rubin, who raised the question of why data is missing - whether it occurs randomly or due to unknown factors. The author classified missing data into three mutually exclusive categories: Missing at Random (MAR), Missing not at Random (MNAR), and Missing Completely at Random (MCAR) [14]. Understanding differences is crucial for addressing missing data and for choosing the way of treating missing values, because in some scenarios incomplete dataset can introduce significant bias and compromise the integrity of statistical analysis. Before starting analysis you always have to ask yourself - if there is a reason why your data is incomplete [17].

Whether data is missing at random or not has a huge impact on how we should start our statistical analysis. The wrong way of dealing with missing data also can quickly lead to a substantial reduction in the sample size and consequent statistical information [4]. While the deletion method would not have a huge impact if data is missing completely at random, in other missingness scenarios results could change drastically.

Even though it is hard to determine the missingness type in real-world data, different studies show, that with different mechanisms, we have to choose different imputation methods and even classifiers. Unai Garciarena and Roberto Santana conducted an experiment [9] to test whether the scores of classifiers for each missingness type are significantly different from each other. Complex analyses were performed to compare classification scores, imputation methods, and missingness mechanisms. The final conclusion emphasized the importance of identifying the missing data pattern whenever possible, because it may influence the choice of imputation method and classification scores.

Various statistical tests might be used for the identification of the missingness mechanism. In the article [8] five clinical trial datasets were used for experimentations and four methods were used to separate mechanisms: Little's test, Listing and Schlittgen (LS) test, Ridout's logistic regression method and Fairclough's logistic regression method. Additionally, since the data involved questionnaires, comparisons were made between immediate and reminder responses. Little's test were recommended for assessing MCAR, while a logistic regression procedure could provide a more detailed analysis. Also, reminder systems were highlighted as valuable tools, because they increase the response rates significantly, as they were found to be equivalent to the immediate responses. However, the tests only determined whether the data were more likely to be MCAR or MAR.

The missing data issue especially arises in the medical field, where data are especially expensive and sensitive. Various imputation methods could play a huge role and make a huge impact. Liu, Chia-Hui and Tsai, Chih-Fong and Sue, Kuen-Liang and Huang, Min-Wei demonstrated that combining feature selection and imputation is a better choice for many medical datasets [13]. As the researchers suggest, performing feature selection to filter unrepresentative features would make the imputation process more effective, and more efficient. Also, the imputation model would be trained by lower dimensional data which could give better estimations and save time on computation. Researchers also experiment with how missing data affects classifier accuracy and the experimental results show

that combining feature selection with imputation methods can make the classifier perform better than the baseline classifier without feature selection.

Before selecting an imputation method it is suggested not only conduct feature selection but also think about data normalization or standardization which can profoundly impact the imputation process, which seldom depends on the imputation technique [10]. Data preprocessing such as outlier handling has a huge impact on imputed data. The imputed values are sometimes unreliable due to the presence of outliers. So outlier handling should be done before imputation methods.

Another consideration after data preprocessing is missingness mechanisms and assumptions about missing data, as these can affect which method should be used to achieve the best results. In the article [16], where authors experiment with deep learning methods and compare them to conventional methods, the results show, that Generative Adversarial Imputation Networks (GAIN) appear reasonable with data that are MCAR, but become poor in cases where data are MAR or MNAR. Authors suggest using traditional methods such as MICE and missforest for tabular data with a limited sample size (n < 30,000). MICE and missforest outperform recently proposed deep generative methods - GAIN and VAE in experiments and no obvious improvement were seen between using embedding or onehot encoding for categorical variables. Authors conclude that missforest and MICE methods are overall the best for now since their results are more stable, the accuracy is high, and they have lower computational cost compared to deep learning methods.

Some researchers report that the easiest way is to complete all the missing data as MAR to some degree because MAR resides in the middle of this continuum. Rubin's MAR assumption is typically a natural starting point if we wish to move beyond a complete records analysis [6].

In the article analysing Multiple Imputation Ensembles (MIE) [1], researchers as one of their most important findings emphasize, that even in scenarios of increased uncertainty, it is possible to obtain results similar or in some cases better than those obtained with the complete data, if the right imputation technique is applied. This is a significant finding as it shows that missing data becomes less of a problem when working with MCAR data.

Although multiple imputation may be time and memory-intensive, particularly with large datasets, its advantages in terms of representing the uncertainty as well as the ability to introduce diversity for the ensemble classifiers enable us to improve classification accuracy for scenarios with large levels of missing data.

In article *Multiple Imputation Through XGBoost* [5] the authors claim that standard MI implementations, such as mice-default, do not automatically account for interaction effects among variables in data. Applying tree-based algorithms such as CART and Random Forests to MI has been shown to alleviate this problem.

For evaluating the quality of imputation methods researchers use different approaches. In the article *Multiple Imputation Ensembles (MIE) for Dealing with Missing Data* [1] researchers use the normalized Euclidean distance. They also divide analysis by the feature type (i.e. numerical, categorical, or mixed) as imputation may work differently for different data types. Finally, they analyse the efficiency of the imputation method based on different data types and they relate this to the performance of different classifiers. Other scientists conclude that, the precision and accuracy of machine

learning imputation algorithms depend strongly on the type of data being analysed, and that there is no clear indication that favors one method over the other.

While multiple imputations seem to be a good way to deal with missing data because of high accuracy, and reduced bias compared to listwise deletion or other traditional methods, some authors emphasize some limitations as well. Scientists especially raise ethical considerations. There is a long history of minoritized individuals having their social identifiers assigned in ways that do not align with their identities emphasizing that imputation might have ethical problems [17].

Another limitation is that the imputed values are treated as known with certainty and treated on an equal footing with the values for the same variable for other subjects for whom the variable was observed and recorded and not imputed [2].

In summary, numerous studies highlight the importance of addressing missingness types and their impact on the selection of imputation methods, which can sometimes even affect the choice of classifiers. Since the problem of missing data arises across various fields, it is crucial to develop effective strategies for handling it properly.

4 Missing data mechanisms

Rubin [3] established the missing data theory and categorized it to three types of missingness. To define missingness, let Y be the entire data matrix that can be decomposed into Y_{obs} and Y_{mis}, which denote the observed and missing data. Let R denote a missingness matrix defined by

$$R := \begin{cases} 0, & \text{if Y is missing} \\ 1, & \text{if Y is observed} \end{cases}$$

This matrix shows if the corresponding entries in Y are observed (1) or missing (0).

Let ψ contain the parameters of the missing data model, then the general expression of the missing data model is $Pr(R = 0|Y_{obs}, Y_{mis}, \psi)$.

4.1 Missing Completely at Random (MCAR)

Missing observations are completely unrelated to both the observed or unobserved measurements, the absence of data occurs randomly. This type can occur when we have data corruption, data loss or survey participant unintentionally skips a question. The probability of MCAR is defined as

$$Pr(R = 0|Y_{obs}, Y_{mis}, \psi) = Pr(R = 0|\psi)$$

If the data are MCAR, the analysis and the estimated parameters remain unbiased.

With such missingness type, listwise or case deletion also could be a choice for handling missing data.

4.2 Missing at Random (MAR)

In MAR, the missingness can be explained by certain observed features in the dataset. Although the data is missing systematically, it is still said to be random because the missingness is unrelated to the unobserved values.

In the MAR mechanism, the missingness of data can be predicted using other observed variables in the study, but not from the missing data itself. In other words, the probability of missingness depends on the observed data, but not on the specific missing values.

This type can occur because of privacy and sensitivity of data. The probability of MAR is defined as

$$Pr(R = 0|Y_{obs}, Y_{mis}, \psi) = Pr(R = 0|Y_{obs}, \psi)$$

4.3 Missing Not at Random (MNAR)

MNAR occurs when the probability of missingness is directly related to the missing values themselves or other unobserved data. In this case, the missing data is not random and is associated with

specific reasons or patterns. Since the reason for missingness cannot be fully explained by the observed data, it is considered the most challenging type of missing data to address.

Addressing MNAR is also termed a non-ignorable process that requires complex techniques, often involving additional data collection, or other assumptions to ensure the integrity and accuracy of the statistical analysis.

Data are MNAR if

$$Pr(R = 0|Y_{obs}, Y_{mis}, \psi)$$

does not simplify and the probability of being missing also depends on unobserved information, including Y_{mis} itself.

4.4 Little's MCAR test

In Little's test of MCAR, the data $y_i, (i = 1, 2, \ldots, n)$ are modeled as p-dimensional multivariate normal distributions with a mean vector μ and covariance matrix Σ, where some components of y_i's are missing. When the normality assumption is not satisfied, Little's test remains asymptotically valid for quantitative random vectors y_i's but is not appropriate for categorical variables. Suppose there are J missing-value patterns among all y_i's. For each pattern j, let o_j and m_j represent the index sets of the observed and missing components, respectively, and $p_j = |o_j|$ be the number of observed components in pattern j. Furthermore, let μ_{o_j} and Σ_{o_j} be the p_j × 1-dimensional mean vector and the $p_j \times p_j$ covariance matrix of only the observed components for the jth missing pattern, and let y_{o_j} (p_j × 1) represent the observed sample mean for the jth missing pattern. Finally, let $I_j \subseteq \{1,2,\ldots,n\}$ be the index set of pattern j in the sample, with $n_j = |I_j|$; then $\sum_{j=1}^{J} n_j = n$. Little's χ^2 test statistic for MCAR is defined as:

$$d_0^2 = \sum_{j=1}^{J} n_j(\bar{y}_{o_j} - \mu_{o_j}^T)\Sigma_{o_j}^{-1}(\bar{y}_{o_j} - \mu_{o_j}) \tag{1}$$

The idea is that if the data are MCAR, then conditional on the missing indicator r_i, the following null hypothesis holds:

$$H_0 : y_{o,i}|r_i \sim N(\mu_{o_j}, \Sigma_{o_j}) \quad \text{if} \quad i \in I_j, 1 \leq j \leq J \tag{2}$$

where μ_{o_j} is a subvector of the mean vector μ. Instead, if the null hypothesis is not true, then conditional on the missing indicator r_i, the means of the observed y's are expected to differ across patterns, leading to the alternative hypothesis:

$$H_1 : y_{o,i}|r_i \sim N(\nu_{o_j}, \Sigma_{o_j}) \quad \text{if} \quad i \in I_j, 1 \leq j \leq J \tag{3}$$

where ν_{o_j}, $j = 1, 2, \cdots, J$ are mean vectors of each pattern j. Rejecting the null hypothesis H_0 is sufficient for rejecting the MCAR assumption, but it is not a necessary condition [12].

5 Methods

5.1 MICE

Multivariate Imputation by Chained Equations (MICE) is a powerful imputation technique for handling missing data. It uses regression models based on the observed data by iteratively imputing missing values, thereby preserving the relationships between variables and reducing bias introduced during the imputation process.

In Python, the MICE technique can be implemented using the *IterativeImputer()* function from the *sklearn.impute* module. This function allows to select from various estimators, such as a simple Linear Regression, Bayesian Ridge (default), K-Nearest Neighbors, to such advanced models as XGBoost. The process involves identifying missing values, initializing the imputer, fitting and transforming the data, and finally replacing the missing values with the generated imputations.

Scikit-learn's *IterativeImputer()* refines the imputed values iteratively, using regression models to capture relationships between features, with each iteration refining the imputed values until convergence or a specified maximum number of iterations is reached.

The *IterativeImputer()* is especially useful where missing values are strongly related to other features in the data. By using the available information in the other features, it often achieves more accurate and meaningful imputations compared to univariate methods like mean or median imputation.

While MICE is a flexible and robust method for handling missing data, it can also be computationally intensive, especially for large datasets and with more computationally expensive, complex estimators.

A more detailed explanation of how MICE algorithm works:

- The missing values are replaced with initial estimates, such as the mean, mode or median of the observed data, which is a temporary replacement to provide a starting point for the iterative process.
- For the specific column in which values are being imputed, the initially imputed values are changed back to missing.
- Regression model predicts column's values. A variable with missing values is temporarily treated as the target variable, the other variables are used as predictors. The missing values are then replaced with predictions generated by the model.
- The process repeats for the k iterations.
- The process repeats for each variable with missing data.
- Multiple Imputation: To account for the uncertainty in the imputations, the above process might be repeated m times, resulting in m complete datasets with slightly varying imputations. This approach is not implemented in sklearn's *IterativeImputer()*, so the process should be

repeated with a loop. R function *mice()* from *mice* package has a parameter m which you can change according to the need, standard values usually vary from 5 to 20.

- Once the m complete datasets are created, statistical analyses are performed separately on each dataset. The results are then combined.

In MICE algorithm we have both - the number of iterations, which ensures stability, and the number of imputations, which capture variability and account for uncertainty. Even when starting with the same initial imputed values, we get slightly different results due to the addition of residuals (random error) to the predicted values. Furthermore, in many implementations of MICE, imputations are drawn from a posterior predictive distribution, additionally, when generating multiple datasets, MICE uses different random seeds for each imputation which gives different results due to randomness.

5.2 Linear Regression Estimator

Linear Regression is a simple and widely used algorithm that assumes a linear relationship. The method is modeling the relationship between dependent and independent variables. It minimizes the residual sum of squares to find the best-fit line. Linear Regression is efficient, suitable for datasets with linear relationships, and is easy to interpret [11]. In the thesis, linear regression is used as an estimator in *IterativeImputer()* function.

5.3 Lasso Estimator

Lasso (Least Absolute Shrinkage and Selection Operator) is a linear regression technique that adds an L1 penalty term to the loss function. This regularization penalizes the absolute values of coefficients, encouraging sparsity by shrinking some coefficients to exactly zero. As a result, Lasso performs variable selection and reduces model complexity, which makes it more effective for high-dimensional datasets and prevents overfitting [11]. When used as the estimator in the *IterativeImputer()* function, Lasso regression predicts missing values by modeling the variable with missing data as a linear combination of other features and L1 penalty helps to reduce multicollinearity and overfitting by selecting the most relevant predictors. The iterative process updates missing values and refines the imputation until convergence.

5.4 Desicion Trees

A Decision Tree is a supervised machine learning algorithm used for classification and regression tasks. It splits the dataset into subsets based on feature values, creating a tree-like structure of decision rules. Each internal node represents a feature split, branches represent outcomes, and leaf nodes represent predictions. Decision Trees are intuitive, easy to interpret, and handle both numerical and categorical data **??**. The method is used as the estimator in the *IterativeImputer()* function.

5.5 Random Forest

Random Forest is an ensemble learning algorithm used for classification and regression tasks. It builds multiple decision trees during training and aggregates their predictions through majority voting (for classification) or averaging (for regression). By using bootstrap sampling and random feature selection at each split, Random Forest reduces overfitting and improves model generalization. It is well-suited for modeling non-linear relationships and handling complex interactions between variables and also provides insights into feature importance [7]. When used as the estimator in the *IterativeImputer()* function, Random Forest predicts missing values by treating each feature with missing data as a dependent variable and using the other features as predictors. The ensemble nature of Random Forest helps capture non-linear relationships and reduces the risk of overfitting. This iterative process continues until the imputed values stabilize, ensuring robust and accurate imputations for datasets with complex structures. Additionally, a simple custom approach was created to implement the Random Forest method by creating a class. The method identifies rows with and without missing values, uses rows without missing values to train a model, predicts the column based on the others and finally applies the model to rows with missing values to generate predictions and the output is the complete dataset.

5.6 XGBoost

XGBoost (Extreme Gradient Boosting) is a powerful machine learning algorithm based on the gradient boosting framework. It builds an ensemble of decision trees sequentially, where each tree corrects the errors of its predecessor by minimizing a specified loss function using gradient descent. XGBoost incorporates advanced features like regularization, tree pruning, and parallel processing, making it highly efficient and capable of handling large datasets with speed and high accuracy. XGBoost is known for its efficiency and ability to model complex, non-linear relationships making it widely used in machine learning tasks and practical applications [5]. In the thesis, XGBoost is used as the estimator in the *IterativeImputer()* function, additionally, a simple custom approach was created to implement the XGBoost method by creating a class. Similar to the implementation of Random Forest, this method also firstly identifies rows with and without missing values, on non-missing rows XGBoost models are trained and, finally, rows with missing values are predicted using this model.

5.7 Light GBM

LightGBM (Light Gradient Boosting Machine) is a gradient boosting framework designed for efficiency and speed. The method is known for its exceptional performance in managing large-scale datasets and high-dimensional features. LightGBM incorporates learning techniques and feature discretization to improve model robustness, reduce memory usage, speed up training, and perform well in classification, regression, and ranking tasks **??**. The method is used as the estimator in the *IterativeImputer()* function.

5.8 Gradient Boosting Trees

Gradient Boosting is an ensemble learning technique that builds a series of decision trees, where each tree is trained to correct the errors of the previous ones. It minimizes a specified loss function in a gradient descent framework. Each tree is trained on the residual errors of the prior model, improving overall accuracy [11]. Gradient Boosting is highly effective and can capture complex, non-linear relationships between features. The method is used as the estimator in the *IterativeImputer()* function.

5.9 KNN

K-Nearest Neighbors (KNN) is a non-parametric method that imputes missing values by identifying the k most similar observations (neighbors) in the dataset based on a distance metric, such as Euclidean distance [16]. It assumes that similar observations have similar values for the feature with missing data. The KNN imputation method has been extensively studied for data imputation, with literature demonstrating its high performance compared to other imputation techniques [6]. KNN is simple to implement, effective for smaller datasets but can be computationally intensive for large datasets. The method is used with *KNN()* function.

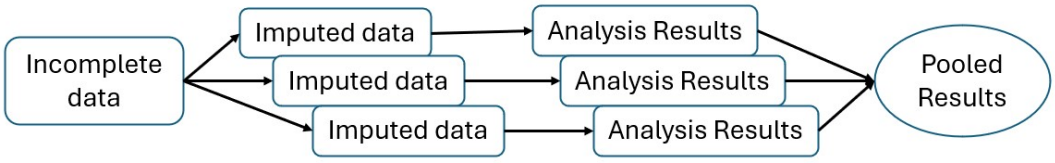
6 Multiple Imputation process

Multiple imputation is a flexible and robust approach for addressing missing data. It creates several different plausible imputed datasets to account for the uncertainty introduced by missing values, thereby minimizing biases that simpler methods introduce. The imputed values are drawn from a distribution that is specifically modeled for each missing entry and this ensures that the imputations reflect the variability in the data. Each dataset is analysed individually, and the results are pooled to produce estimates [15].

The process consists of three main steps:

- **Create imputed datasets:** In the first step, multiple datasets are generated by replacing missing values with imputed values. These imputations rely on statistical models to provide plausible replacements for the missing data.
- **Analyse imputed datasets:** The second step includes estimating the parameters of interest for each of the imputed datasets. The results differ in each of the imputed datasets because of the variation caused because of the uncertainty about what value to impute.
- **Pool the results:** The last step combines or pools the results from all imputed datasets to produce a single set of parameter estimates. The pooling step typically involves calculating the overall variance, which accounts for both within-dataset variability (arising from the data itself) and between-dataset variability (arising from the imputation process). It is done to ensure that the final results reflect the uncertainty caused by missing values. Greater variability across the imputed datasets indicates greater uncertainty due to missing data.

In 1 figure. multiple imputation process is presented.



1 figure. *Multiple Imputation process*

7 Simulation study

Data simulations and statistical analyses were conducted using Python. The simulation consists of four sequential stages:

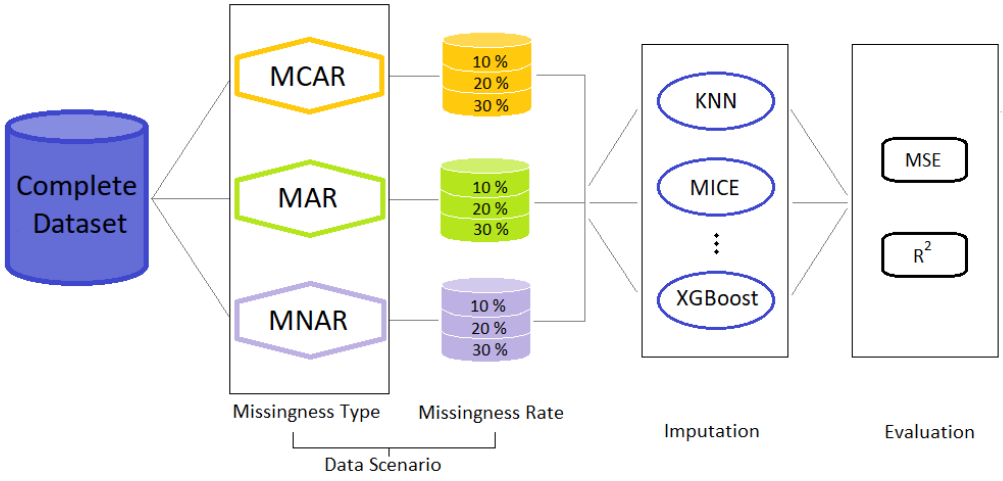
1. Data generation: simulate complete datasets with different scenarios and sizes.
2. Amputation: introduce missing values to complete datasets according to specific rules to generate missing values.
3. Imputation: fill in the missing values of the simulated incomplete datasets by different imputation methods.
4. Analysis and evaluation: perform statistical analysis, calculate evaluation metrics to estimate accuracy of imputation and regression accuracy.

Experiments were executed/repeated 10 times, to get more robust results, however, the performance of each experiment was not significantly different, and the average performance is reported.

Data was generated using function *make_regression* from *sklearn.datasets*. Data were generated with all three missingness types - MCAR, MAR and MNAR changing missingness rate.

Six different sizes datasets (100, 300, 500, 700, 1000, 5000) with five different missingness ratios (10 %, 15 %, 20 %, 25 %, 30 %) in each dataset with different missingness types were analysed. A total of 9900 experiments were conducted: three missingness types × five missingness rates × six dataset sizes × eleven imputation methods x repeated 10 times.

In 2 figure. experimentations framework is presented.



2 figure. *Experimentations framework*

8 Evaluation indicators

The main approach to test the accuracy between methods was calculating the error between the imputed and actual data. Mean Square Error (MSE): MSE is the average squared difference between the estimated and true values, defined as:

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

where m is the number of missing values in the dataset, y_i is the true value, and $\hat{y}_i$ is the estimated value. The lower the value, the more accurate the imputation to the true value.

Additionally, simple linear regression models were built and the coefficient of determination, denoted R^2 was calculated to test the model's accuracy. R^2 is a number that tells how well the independent variable(s) in a statistical model explains the variation in the dependent variable. A higher R^2 score shows smaller differences between the observed data and the fitted values, it ranges from 0 to 1. The formula for calculating R^2 is:

$$R^2 = 1 - \frac{SS_{residual}}{SS_{total}}, \tag{4}$$

where

- $SS_{residual}$ is the sum of squared differences between the observed and predicted values:

$$SS_{residual} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{5}$$

  here y_i is observed value and $\hat{y}_i$ is predicted value.

- SS_{total} is the total sum of squares:

$$SS_{total} = \sum_{i=1}^{n}(y_i - \overline{y_i})^2 \tag{6}$$

  here $\overline{y_i}$ is a mean of the observed values.

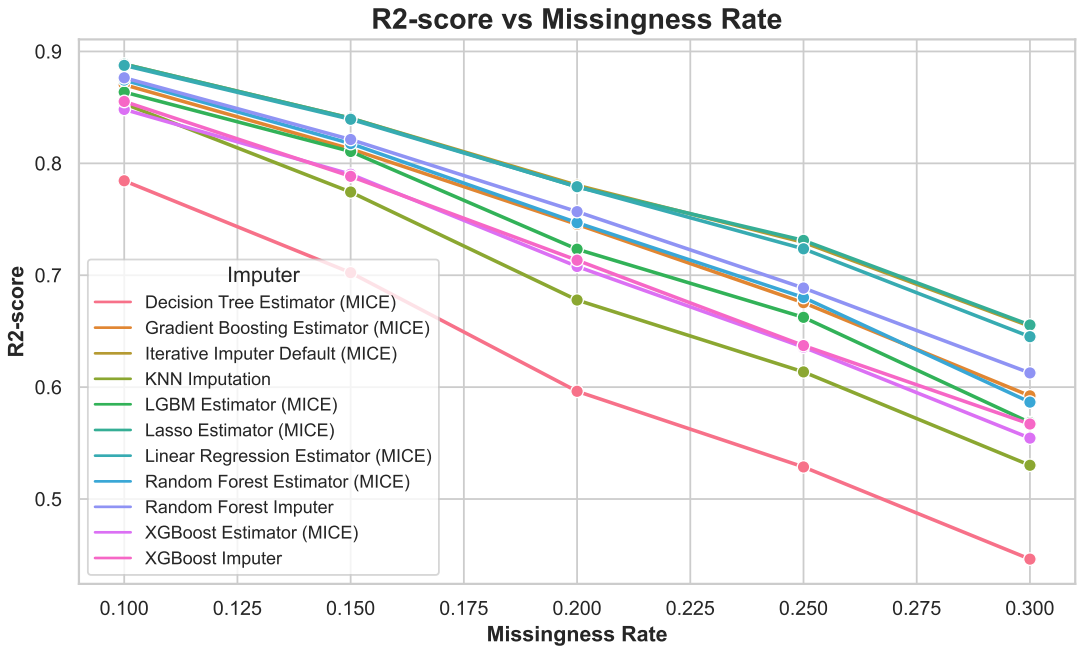
9 Software

To conduct experiments I decided to use Python due to its ease of integration and the ability to create custom methods, while R focuses more on statistical analysis. However, during the experimentation process, I found out that R might be a better choice since it offers a wider range of implemented methods, also more information about how to apply multiple imputations techniques. Python, on the other hand, is more focused on using machine learning algorithms for missing value imputations. To apply imputation methods I mostly used *IterativeImputer()* function, which is an analog to R's *mice()* function from the *mice* library. The main difference, and a key limitation of *IterativeImputer()* function in Python, is that it lacks a parameter to directly specify the number of multiple imputations. So to apply multiple imputations in Python, you need to manually loop through the process m times to generate m complete dataset which can then be used for further analysis or modeling.

10 Results

An experimental study was conducted to evaluate the performance of different imputation methods, with a focus on their suitability for different data scenarios. The goal was to determine which methods work the best under specific conditions with various missingness rates and dataset sizes. In the 3 table. all results are presented across these dataset specifications. The study revealed that while the most effective and optimal imputation method varies slightly depending on the specific parameters and scenario, certain methods consistently outperformed others across all the conditions. Specifically, MICE with linear regression, Lasso, and the default - Bayesian Ridge estimator were found as the most effective and accurate approaches. These methods demonstrated the ability to handle a wide range of cases effectively because they were achieving the best results not only with the lowest MSE, and the highest R^2 scores but also by their efficient imputation times, which makes them a good practical choices for diverse datasets.

Despite its limited application in existing literature, Lasso estimator within the MICE method proved to be the most effective for the largest datasets, consistently delivering good results, regardless of what missingness rates they had.



3 figure. *Missingness rate and R^2 - score*

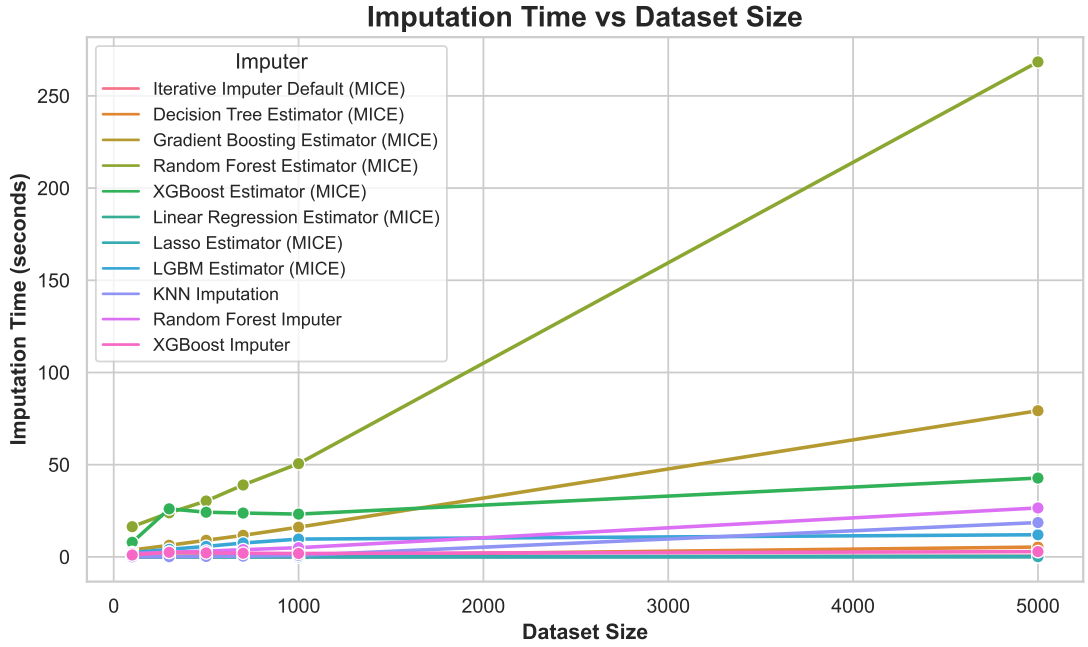
For smaller datasets, Random Forest imputation method could also be the one to think about while choosing an imputation method. Since all tree-based algorithms can capture complex, nonlinear relationships between variables, this characteristic is a significant advantage in scenarios where traditional linear assumptions may not hold. However, Random Forest had a slower computation time compared to other methods, which might limit its usage in time-sensitive applications or with larger datasets.

For cases with a high missingness rate, the MICE method with Lasso or the default - Bayesian

Ridge estimator proved to be the most robust, maintaining higher accuracy and reliability compared to other methods.

After conducting multiple experiments to test how different methods and estimators work on different data scenarios, I found that imputation accuracy, particularly measured by MSE is mostly impacted by the missingness rate. In contrast, the size of the dataset was found less significant on both MSE and R^2 scores.

After conducting a more detailed analysis using a simple linear regression model on simulated data I found a clear pattern: with increasing missingness rate, we see a rapid decrease in the R^2 score. For example, models that achieved $R^2 \geq 0.9$ with a missingness rate of 0.1 decreased to less than 0.7 when the missingness rate was 0.3. This trend illustrates the profound challenges posed by missing data, even when advanced imputation techniques are applied.



4 figure. *Dataset Size and Imputation Time*

As we can see in 4 figure., imputation time increases significantly with dataset size. The biggest increase can be seen for the Random Forest Estimator. However, all tree-based estimators for MICE method seems to be computationally expensive. Custom Random Forest and XGBoost imputers show much better results regarding imputation time, also similar accuracy as in *IterativeImputer()* function with those estimators. So in those cases where you want to apply the tree-based imputation method, you need to think about implementing it by yourself rather than using *IterativeImputer()*.

In summary, the experimentation study demonstrated that currently choosing MICE with such estimator as Linear Regression, Bayesian Ridge (default) or Lasso would be the most effective approach regardless of the missingness rate, type, or dataset size.

11 Case study

The data was collected using a web-scraping method from one of Lithuania's most popular car listing websites, https://autogidas.lt.

Due to the large number of different groups of categorical variables, some groups have been combined according to their similarity to avoid those groups with a very small number of observations. Categorical variables were encoded because most imputation methods can handle only numerical variables.

Also, only records where the steering position is on the left were included in the dataset. The final dataset contains 7246 observations and has 14 features.

Descriptive statistics and missingness rates are summarized in 1 table. Four variables - mileage, power, weight and the number of doors - contain missing values, each with a different missingness rate. Variable *Weight* has the highest proportion of missing values, reaching 80 %.

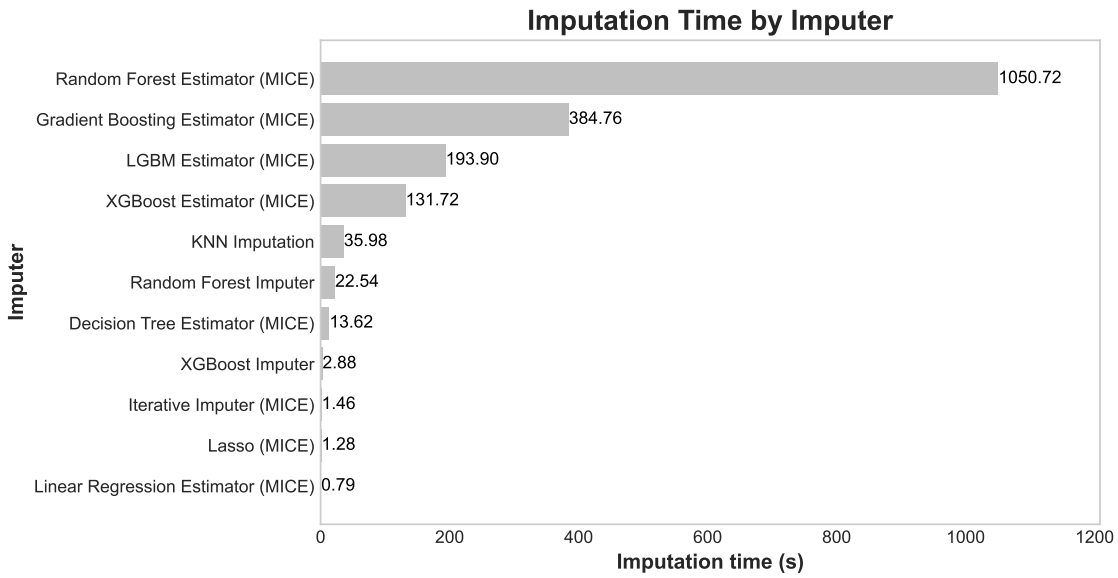
1 table. *Descriptive statistics of case study data*

Variable	Mean (SD) / No. of categories	No. of observations with missing data	Percentage of missing data
Make - categorical	72 categories	0	0 %
Model - categorical	756 categories	0	0 %
Year - numerical	15.96 (6.64)	0	0 %
Fuel Type - categorical	4	0	0 %
Mileage - numerical	257775.95 (286207.74)	1994	27.52 %
Defects - categorical	2 categories	0	0 %
Gearbox - categorical	3 categories	0	0 %
Engine (l) - numerical	2.14 (0.65)	0	0 %
Power (kW) - numerical	110.53 (45.06)	1253	17.29 %
Body Type - categorical	10 categories	0	0 %
Weight - numerical	2028.95 (396.22)	5982	82.56 %
Driven Wheels - categorical	3 categories	0	0 %
Doors - numerical	3.84 (0.56)	518	7.15 %
Price - numerical	5408.15 (7598.97)	0	0 %

The variables *Make* and *Model* of the car did not have missing values and were excluded, due to their large number of classes, which makes it difficult to estimate and analyse them properly, especially for rare types.

To determine the type of missingness in the dataset - MCAR, MAR or MNAR - I applied Little's MCAR test. This test was recommended in the literature [8], and it assesses whether the data is MCAR or not. Since Python is not so advanced for statistical analysis, R package *naniar* and its function *mcar_test()* were used to conduct Little's test to test if data is MCAR or not. The results of Little's MCAR test showed that data is not MCAR with significance level $\alpha = 0.05$ as the p-value was less than α.

For imputation evaluation, I developed a simple linear regression model to predict the price of a car and evaluate the accuracy of the prediction using various imputation approaches. Given the



5 figure. *Imputation time by imputation method*

incomplete dataset, it is not possible to measure the error between imputed and actual observations, so the imputation evaluation process consisted only of model creation and accuracy comparison. The linear regression model predicts the price of a car, and to improve the performance, the target variable - price - is log-transformed to address its right-skewed distribution.

Even though testing imputation by creating a simple model to identify the most effective imputation method is not an ideal approach for imputation evaluation, the absence of actual values limits us and we can't compare actual and imputed values and calculate error between them.

As shown in 5 figure., MICE method with a Random Forest estimator required the longest time. MICE method with a simple Linear Regression estimator was the fastest.

2 table. *Model evaluation using different imputation methods*

Method	MSE	R^2 score	Imputation time
Iterative Imputer (MICE)	0.12	0.90	1.46
Linear Regression Estimator (MICE)	0.11	0.90	0.79
Lasso Estimator (MICE)	0.12	0.89	1.28
Decision Tree Estimator (MICE)	0.26	0.77	13.62
KNN Imputation	0.25	0.78	35.98
Random Forest Imputer	0.26	0.77	22.54
Random Forest Estimator (MICE)	0.26	0.77	1050.72
XGBoost Estimator (MICE)	0.26	0.76	131.72
XGBoost Imputer	0.27	0.76	2.88
Gradient Boosting Estimator (MICE)	0.26	0.76	384.76
LGBM Estimator (MICE)	0.27	0.75	193.90

The 2 table. shows the results, including MSE, R^2 score and imputation time for each imputation method used. Once again, the default MICE method with a Bayesian Ridge regression estimator showed one of the best results. However, MICE method with a Linear Regression estimator achieved even better results and was also the fastest approach.

In MICE, tree-based regressor estimators required significantly more time to impute values, with Random Forest estimator (MICE) imputation taking 1050.72 seconds. The custom Random Forest method showed the same level of accuracy, but completed the imputation much faster, in only 22.54 seconds, which is more than 46 times less.

The worst accuracy was observed for the LGBM regressor estimator in MICE, which achieved an R^2 score of only 0.75.

To highlight the importance of imputation, additionally, model evaluations were conducted using complete dataset and dataset without columns containing missing values. Since the *Weight* column has more than 82 % missing values, only the remaining 18 % of the dataset could be used. Also, a dataset with columns that do not have missing values was tested. A linear regression model was applied, and the results showed, that accuracy decreased significantly. The dataset without columns containing missing values had an R^2 score $= 0.625002$, while the dataset with complete data had R^2 score $= 0.721228$. Both values are considerably lower than those obtained with imputed data, underscoring that imputation is highly beneficial and can improve prediction significantly.

As recommended in various research, data was scaled with Python's function *StandardScaler()* from *sklearn*, and also min-max normalization was applied $\frac{x-x_{min}}{x_{max}-x_{min}}$, where x_{min} is minimum value of variable and x_{max} is maximum value. However, an increase in accuracy was not seen. On the other hand, results might vary because of the specifics of the dataset.

In summary, since the type of missingness was determined to be not MCAR, it is usually more challenging to impute values since there might be unobserved factors that may influence the missingness. Nevertheless, imputation proved to be highly effective in reaching better accuracy for predicting a car's price on real-world data, with the best methods achieving a R^2 score of 0.9. and an imputation time of around one second.

12 Conclusions

The results of the simulation experiments and case study demonstrate that while tree-based algorithms are beneficial in finding complex and nonlinear relationships between variables, but well-known MICE algorithm currently stands out as the best approach for imputing values, including accuracy, implementation process, and computational time. MICE method is a powerful approach for handling missing data, with flexibility for using various estimators makes it suitable for diverse datasets and could help to achieve better model accuracy in those cases where missing data is observed. This is especially critical in contexts such as medical experiments or other cases where data collection is expensive, and every data point is crucial. MICE method implemented with *IterativeImputation()* function from *sklearn* shows a good performance, especially with such estimators as Linear Regression, Bayesian Ridge (default) and Lasso. These estimators demonstrated good performance in both small and large datasets, regardless of the missingness rate. Currently, Random Forest may be used for smaller datasets because of the computation time and because only for smaller datasets accuracy seemed to be better compared to other methods. As recommended in the literature, standardization did not show a significant advantage in this study and it did not improve the accuracy. However, this outcome may vary across different data scenarios.

As highlighted in the simulation study, the missingness rate has a huge impact on the model's accuracy, so selecting the most suitable approach for addressing missing values are very important step in preparing data for analysis.

13 Limitations and Future work

While this study provided valuable insights about multiple imputation process and different estimators that could be implemented and used in Python's *IterativeImputer()* function or made by custom classes and functions, several limitations remain, that could be analysed in future research.

One area that requires further exploration is the impact of the number of imputations on accuracy, as this study did not evaluate how this parameter might influence the results across different datasets and scenarios.

Another limitation is that this study focuses on numerical data, leaving handling categorical values unexplored. Expanding the analysis to include categorical data would give a wider range of insights and provide practitioners with better guidance while choosing between imputation techniques, particularly since real-world datasets usually contain mixed data types.

References and sources

[1] A. Aleryani, W. Wang, B. de la Iglesia. “Multiple imputation ensembles (MIE) for dealing with missing data.” In: *SN Computer Science* 1.3 (2020). URL: https://link.springer.com/article/10.1007/s42979-020-00131-0.

[2] P. C. Austin, I. R. White, D. S. Lee, S. van Buuren. “Missing Data in Clinical Research: A Tutorial on Multiple Imputation.” In: *Canadian Journal of Cardiology* 37.9 (2021), pages 1322–1331. ISSN: 0828-282X. URL: https://www.sciencedirect.com/science/article/pii/S0828282X20311119.

[3] S. van Buuren. *Flexible Imputation of Missing Data*. A Chapman & Hall book. CRC Press, Taylor & Francis Group, 2018. ISBN: 9781138588318. URL: https://books.google.lt/books?id=bLmItgEACAAJ.

[4] J. R. Carpenter, M. Smuk. “Missing data: A statistical framework for practice.” In: *Biometrical Journal* 63.5 (2021), pages 915–947. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/bimj.202000196.

[5] Y. Deng, T. Lumley. “Multiple Imputation Through XGBoost.” In: *Journal of Computational and Graphical Statistics* 33.2 (2024), pages 352–363. URL: https://doi.org/10.1080/10618600.2023.2252501.

[6] T. Emmanuel, T. Maupong, D. Mpoeleng, T. Semong, B. Mphago, O. Tabona. “A survey on missing data in machine learning.” In: *Journal of Big Data* 8 (2021). https://doi.org/https://doi.org/10.1186/s40537-021-00516-9.

[7] R. Feng, D. Grana, N. Balling. “Imputation of missing well log data by random forest and its uncertainty analysis.” In: *Computers Geosciences* 152 (2021), page 104763. ISSN: 0098-3004. https://doi.org/https://doi.org/10.1016/j.cageo.2021.104763. URL: https://www.sciencedirect.com/science/article/pii/S0098300421000704.

[8] S. Fielding, P. M. Fayers, C. R. Ramsay. “Investigating the missing data mechanism in quality of life outcomes: a comparison of approaches.” In: *Health and Quality of Life Outcomes* 17 (2009). URL: https://hqlo.biomedcentral.com/articles/10.1186/1477-7525-7-57.

[9] U. Garciarena, R. Santana. “An extensive analysis of the interaction between missing data types, imputation methods, and supervised classifiers.” In: *Expert Systems with Applications* 89 (2017), pages 52–65. ISSN: 0957-4174. https://doi.org/https://doi.org/10.1016/j.eswa.2017.07.026. URL: https://www.sciencedirect.com/science/article/pii/S095741741730502X.

[10] M. K. Hasan, M. A. Alam, S. Roy, A. Dutta, M. T. Jawad, S. Das. “Missing value imputation affects the performance of machine learning: A review and analysis of the literature (2010–2021).” In: *Informatics in Medicine Unlocked* 27 (2021), page 100799.

[11] T. Hastie, R. Tibshirani, J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.)* 2009. URL: https://link.springer.com/book/10.1007/978-0-387-84858-7.

[12] C. Li. "Little's Test of Missing Completely at Random." In: *The Stata Journal* 13.4 (2013), pages 795–809. https://doi.org/10.1177/1536867X1301300407. URL: https://doi.org/10.1177/1536867X1301300407.

[13] C.-H. Liu, C.-F. Tsai, K.-L. Sue, M.-W. Huang. "The Feature Selection Effect on Missing Value Imputation of Medical Datasets." In: *Applied Sciences* 10.7 (2020). ISSN: 2076-3417. https://doi.org/10.3390/app10072344. URL: https://www.mdpi.com/2076-3417/10/7/2344.

[14] D. B. Rubin. "Inference and missing data." In: *Biometrika* 63 (3 1976), pages 581–592. https://doi.org/https://doi.org/10.1093/biomet/63.3.581.

[15] J. A. C. Sterne, I. R. White, J. B. Carlin, M. Spratt, P. Royston, M. G. Kenward, A. M. Wood, J. R. Carpenter. "Multiple imputation for missing data in epidemiological and clinical research: potential and pitfalls." In: *BMJ* 338 (2009). ISSN: 0959-8138. https://doi.org/10.1136/bmj.b2393. URL: https://www.bmj.com/content/338/bmj.b2393.

[16] Y. Sun, J. Li, Y. Xu, T. Zhang, X. Wang. "Deep learning versus conventional methods for missing data imputation: A review and comparative study." In: *Expert Systems with Applications* 227 (2023), page 120201. ISSN: 0957-4174. URL: https://www.sciencedirect.com/science/article/pii/S0957417423007030.

[17] A. D. Woods, D. Gerasimova, B. Van Dusen, J. Nissen, et al. "Best practices for addressing missing data through multiple imputation." In: *Infant and Child Development* 33.1 (2024), e2407. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/icd.2407.

Appendix 1. Simulation Results

3 table. *Simulation Results (sorted descending by R^2 score)*

Missingness Rate	Size	Method	MSE	R^2 score	Imputation Time
0.1	100	Linear Regression Estimator (MICE)	1.05328	0.89940	0.07362
0.1	100	Iterative Imputer Default (MICE)	1.02204	0.89800	0.11423
0.1	100	Random Forest Imputer	1.17197	0.89619	1.78684
0.1	100	Lasso Estimator (MICE)	1.00429	0.89608	0.01748
0.1	100	KNN Imputation	1.31250	0.89058	0.02854
0.1	100	Random Forest Estimator (MICE)	1.18961	0.88939	16.86466
0.1	100	XGBoost Imputer	1.41189	0.88323	1.04204
0.1	100	LGBM Estimator (MICE)	1.19360	0.87542	2.47558
0.1	100	Gradient Boosting Estimator (MICE)	1.35827	0.86288	3.86389
0.1	100	XGBoost Estimator (MICE)	1.44570	0.86268	8.55955
0.1	100	Decision Tree Estimator (MICE)	2.10776	0.80705	0.17651
0.1	300	Lasso Estimator (MICE)	1.05952	0.87974	0.02186
0.1	300	Iterative Imputer Default (MICE)	1.05941	0.87834	0.09883
0.1	300	Linear Regression Estimator (MICE)	1.06699	0.87373	0.07179
0.1	300	Random Forest Imputer	1.16223	0.85900	2.62961
0.1	300	KNN Imputation	1.37895	0.85587	0.10187
0.1	300	LGBM Estimator (MICE)	1.33397	0.85144	4.33043
0.1	300	Random Forest Estimator (MICE)	1.16893	0.85080	24.92505
0.1	300	Gradient Boosting Estimator (MICE)	1.20380	0.84879	6.91103
0.1	300	XGBoost Imputer	1.34919	0.84292	2.39344
0.1	300	XGBoost Estimator (MICE)	1.37550	0.82290	24.91805
0.1	300	Decision Tree Estimator (MICE)	2.02367	0.77041	0.36426
0.1	500	Iterative Imputer Default (MICE)	1.03783	0.88196	0.10166
0.1	500	Lasso Estimator (MICE)	1.03712	0.88195	0.02296
0.1	500	Linear Regression Estimator (MICE)	1.04860	0.88010	0.07023
0.1	500	Random Forest Estimator (MICE)	1.16709	0.86365	34.70784
0.1	500	Random Forest Imputer	1.16379	0.86064	3.50760
0.1	500	Gradient Boosting Estimator (MICE)	1.18949	0.85770	10.06729
0.1	500	LGBM Estimator (MICE)	1.32340	0.84446	6.37174
0.1	500	KNN Imputation	1.47384	0.83560	0.22703
0.1	500	XGBoost Estimator (MICE)	1.40667	0.83297	25.74441
0.1	500	XGBoost Imputer	1.39985	0.82970	2.25004
0.1	500	Decision Tree Estimator (MICE)	2.15060	0.75923	0.52397

Continued on next page

Table 3 table. – continued from previous page

Missingness Rate	Size	Method	MSE	R^2 score	Imputation Time
0.1	700	Lasso Estimator (MICE)	1.03383	0.88615	0.02321
0.1	700	Iterative Imputer Default (MICE)	1.03481	0.88602	0.10611
0.1	700	Linear Regression Estimator (MICE)	1.04198	0.88531	0.07226
0.1	700	Gradient Boosting Estimator (MICE)	1.15302	0.87690	13.11093
0.1	700	Random Forest Imputer	1.16057	0.87472	4.23685
0.1	700	Random Forest Estimator (MICE)	1.17108	0.87451	42.78030
0.1	700	LGBM Estimator (MICE)	1.31198	0.85897	8.27952
0.1	700	KNN Imputation	1.45066	0.85327	0.36751
0.1	700	XGBoost Estimator (MICE)	1.41098	0.84897	26.84449
0.1	700	XGBoost Imputer	1.41439	0.84465	2.20397
0.1	700	Decision Tree Estimator (MICE)	2.10951	0.79234	0.70549
0.1	1000	Lasso Estimator (MICE)	1.05936	0.89786	0.02045
0.1	1000	Iterative Imputer Default (MICE)	1.06006	0.89751	0.10595
0.1	1000	Linear Regression Estimator (MICE)	1.06259	0.89671	0.07156
0.1	1000	Gradient Boosting Estimator (MICE)	1.13855	0.88796	18.01281
0.1	1000	Random Forest Estimator (MICE)	1.17140	0.88708	58.98410
0.1	1000	Random Forest Imputer	1.16642	0.88624	5.75414
0.1	1000	LGBM Estimator (MICE)	1.31534	0.86955	10.10703
0.1	1000	XGBoost Imputer	1.37412	0.86264	1.69774
0.1	1000	XGBoost Estimator (MICE)	1.39167	0.85468	20.62727
0.1	1000	KNN Imputation	1.51832	0.84474	0.65974
0.1	1000	Decision Tree Estimator (MICE)	2.14501	0.78715	1.00582
0.1	5000	Lasso Estimator (MICE)	1.05599	0.89005	0.08657
0.1	5000	Iterative Imputer Default (MICE)	1.05616	0.89004	0.26926
0.1	5000	Linear Regression Estimator (MICE)	1.05729	0.88999	0.22430
0.1	5000	Gradient Boosting Estimator (MICE)	1.07858	0.88749	92.20181
0.1	5000	LGBM Estimator (MICE)	1.13552	0.88226	12.00807
0.1	5000	Random Forest Imputer	1.13210	0.88208	31.54711
0.1	5000	Random Forest Estimator (MICE)	1.13883	0.88101	329.01047
0.1	5000	XGBoost Imputer	1.24979	0.86874	2.95727
0.1	5000	XGBoost Estimator (MICE)	1.26910	0.86694	40.97241
0.1	5000	KNN Imputation	1.55633	0.83966	17.46148
0.1	5000	Decision Tree Estimator (MICE)	2.12810	0.79033	6.30269
0.2	100	Iterative Imputer Default (MICE)	1.11190	0.79846	0.10305
0.2	100	Linear Regression Estimator (MICE)	1.17112	0.79810	0.08935
0.2	100	Lasso Estimator (MICE)	1.10093	0.79042	0.01833

Continued on next page

Table 3 table. – continued from previous page

Missingness Rate	Size	Method	MSE	R^2 score	Imputation Time
0.2	100	Random Forest Imputer	1.32088	0.78567	1.74351
0.2	100	LGBM Estimator (MICE)	1.34872	0.77562	2.35197
0.2	100	Random Forest Estimator (MICE)	1.38041	0.77015	16.29360
0.2	100	XGBoost Imputer	1.60580	0.74601	0.99532
0.2	100	Gradient Boosting Estimator (MICE)	1.57073	0.73897	3.78220
0.2	100	XGBoost Estimator (MICE)	1.60525	0.72918	7.84696
0.2	100	KNN Imputation	1.60026	0.69700	0.03246
0.2	100	Decision Tree Estimator (MICE)	2.14697	0.61718	0.16297
0.2	300	Iterative Imputer Default (MICE)	1.13785	0.79375	0.10590
0.2	300	Lasso Estimator (MICE)	1.12939	0.79340	0.02134
0.2	300	Linear Regression Estimator (MICE)	1.16416	0.78828	0.08357
0.2	300	Random Forest Imputer	1.28256	0.76913	2.52382
0.2	300	Random Forest Estimator (MICE)	1.32365	0.75544	24.21167
0.2	300	Gradient Boosting Estimator (MICE)	1.36083	0.74244	6.21212
0.2	300	LGBM Estimator (MICE)	1.48562	0.72009	4.09439
0.2	300	XGBoost Imputer	1.48053	0.71612	2.68732
0.2	300	KNN Imputation	1.61655	0.69538	0.12331
0.2	300	XGBoost Estimator (MICE)	1.54211	0.69454	27.25270
0.2	300	Decision Tree Estimator (MICE)	2.18305	0.59279	0.29227
0.2	500	Iterative Imputer Default (MICE)	1.14572	0.75699	0.10733
0.2	500	Lasso Estimator (MICE)	1.14289	0.75665	0.02302
0.2	500	Linear Regression Estimator (MICE)	1.15842	0.75619	0.08317
0.2	500	Random Forest Imputer	1.27598	0.71960	3.06139
0.2	500	Gradient Boosting Estimator (MICE)	1.32620	0.71925	8.94570
0.2	500	Random Forest Estimator (MICE)	1.33145	0.71818	29.19397
0.2	500	XGBoost Estimator (MICE)	1.55241	0.67891	24.53625
0.2	500	LGBM Estimator (MICE)	1.48502	0.66760	5.62833
0.2	500	XGBoost Imputer	1.55559	0.66245	2.15609
0.2	500	KNN Imputation	1.66720	0.63719	0.25548
0.2	500	Decision Tree Estimator (MICE)	2.21025	0.56373	0.41672
0.2	700	Linear Regression Estimator (MICE)	1.12888	0.77588	0.08169
0.2	700	Iterative Imputer Default (MICE)	1.12024	0.77551	0.10770
0.2	700	Lasso Estimator (MICE)	1.11825	0.77529	0.02262
0.2	700	Random Forest Imputer	1.25252	0.75042	3.93619
0.2	700	Gradient Boosting Estimator (MICE)	1.25017	0.74917	11.58522
0.2	700	Random Forest Estimator (MICE)	1.28071	0.74627	39.04982

Continued on next page

Table 3 table. – continued from previous page

Missingness Rate	Size	Method	MSE	R^2 score	Imputation Time
0.2	700	LGBM Estimator (MICE)	1.46242	0.71825	7.51746
0.2	700	XGBoost Imputer	1.51117	0.70492	2.04225
0.2	700	XGBoost Estimator (MICE)	1.50443	0.70413	23.56137
0.2	700	KNN Imputation	1.66587	0.68001	0.40824
0.2	700	Decision Tree Estimator (MICE)	2.15651	0.60803	0.57368
0.2	1000	Lasso Estimator (MICE)	1.12090	0.78781	0.02588
0.2	1000	Iterative Imputer Default (MICE)	1.12203	0.78769	0.09777
0.2	1000	Linear Regression Estimator (MICE)	1.12733	0.78651	0.09094
0.2	1000	Gradient Boosting Estimator (MICE)	1.21703	0.76523	16.11457
0.2	1000	Random Forest Imputer	1.23312	0.76450	4.86844
0.2	1000	Random Forest Estimator (MICE)	1.26665	0.75506	49.42384
0.2	1000	LGBM Estimator (MICE)	1.42520	0.72176	9.81714
0.2	1000	XGBoost Estimator (MICE)	1.47178	0.72076	24.41583
0.2	1000	XGBoost Imputer	1.46557	0.71575	1.89283
0.2	1000	KNN Imputation	1.67308	0.68066	0.81570
0.2	1000	Decision Tree Estimator (MICE)	2.17118	0.60283	0.80209
0.2	5000	Lasso Estimator (MICE)	1.12557	0.77006	0.08469
0.2	5000	Iterative Imputer Default (MICE)	1.12582	0.77002	0.27680
0.2	5000	Linear Regression Estimator (MICE)	1.12719	0.76979	0.23229
0.2	5000	Gradient Boosting Estimator (MICE)	1.19355	0.75771	79.36056
0.2	5000	Random Forest Imputer	1.22398	0.75170	26.37992
0.2	5000	Random Forest Estimator (MICE)	1.29951	0.73664	264.60414
0.2	5000	LGBM Estimator (MICE)	1.29480	0.73632	12.02624
0.2	5000	XGBoost Imputer	1.31567	0.73524	2.58428
0.2	5000	XGBoost Estimator (MICE)	1.39266	0.71948	38.95492
0.2	5000	KNN Imputation	1.62167	0.67680	19.08751
0.2	5000	Decision Tree Estimator (MICE)	2.17673	0.59310	5.28863
0.3	100	Lasso Estimator (MICE)	1.16854	0.67998	0.01743
0.3	100	Iterative Imputer Default (MICE)	1.20788	0.67580	0.16094
0.3	100	Random Forest Imputer	1.36451	0.66197	1.71814
0.3	100	Linear Regression Estimator (MICE)	1.35839	0.63041	0.11861
0.3	100	XGBoost Imputer	1.67529	0.61722	0.98904
0.3	100	Gradient Boosting Estimator (MICE)	1.61171	0.60611	3.56872
0.3	100	LGBM Estimator (MICE)	1.44190	0.60460	2.28376
0.3	100	KNN Imputation	1.60055	0.57865	0.03624
0.3	100	Random Forest Estimator (MICE)	1.46515	0.57841	15.98084

Continued on next page

Table 3 table. – continued from previous page

Missingness Rate	Size	Method	MSE	R^2 score	Imputation Time
0.3	100	XGBoost Estimator (MICE)	1.67685	0.55502	7.31852
0.3	100	Decision Tree Estimator (MICE)	2.18322	0.47150	0.15372
0.3	300	Iterative Imputer Default (MICE)	1.19095	0.64111	0.14665
0.3	300	Lasso Estimator (MICE)	1.18804	0.64066	0.02311
0.3	300	Linear Regression Estimator (MICE)	1.22434	0.63998	0.11691
0.3	300	Random Forest Estimator (MICE)	1.42070	0.58644	22.08065
0.3	300	Random Forest Imputer	1.37196	0.58302	2.33755
0.3	300	LGBM Estimator (MICE)	1.56853	0.55942	3.85296
0.3	300	Gradient Boosting Estimator (MICE)	1.47110	0.54369	5.79184
0.3	300	XGBoost Estimator (MICE)	1.58709	0.54105	24.94014
0.3	300	XGBoost Imputer	1.65794	0.52073	2.31913
0.3	300	KNN Imputation	1.74430	0.50919	0.14405
0.3	300	Decision Tree Estimator (MICE)	2.20543	0.42646	0.28171
0.3	500	Lasso Estimator (MICE)	1.25349	0.63390	0.02313
0.3	500	Iterative Imputer Default (MICE)	1.26208	0.63172	0.13518
0.3	500	Linear Regression Estimator (MICE)	1.28521	0.62495	0.10704
0.3	500	Random Forest Imputer	1.42496	0.57600	2.82059
0.3	500	Random Forest Estimator (MICE)	1.50303	0.56402	27.43830
0.3	500	Gradient Boosting Estimator (MICE)	1.48113	0.55897	8.06137
0.3	500	XGBoost Imputer	1.68990	0.52755	2.27754
0.3	500	LGBM Estimator (MICE)	1.70217	0.51904	5.08670
0.3	500	XGBoost Estimator (MICE)	1.68636	0.51395	24.74103
0.3	500	KNN Imputation	1.77518	0.49531	0.28411
0.3	500	Decision Tree Estimator (MICE)	2.29013	0.42137	0.38883
0.3	700	Lasso Estimator (MICE)	1.20672	0.65340	0.02104
0.3	700	Iterative Imputer Default (MICE)	1.21337	0.65251	0.13443
0.3	700	Linear Regression Estimator (MICE)	1.22746	0.65061	0.09725
0.3	700	Random Forest Imputer	1.37085	0.61940	3.52695
0.3	700	Gradient Boosting Estimator (MICE)	1.39797	0.61469	10.45766
0.3	700	Random Forest Estimator (MICE)	1.44135	0.60369	35.17978
0.3	700	LGBM Estimator (MICE)	1.59601	0.57141	6.70783
0.3	700	XGBoost Imputer	1.61993	0.56863	1.77213
0.3	700	XGBoost Estimator (MICE)	1.59846	0.56782	20.88564
0.3	700	KNN Imputation	1.70401	0.52605	0.45259
0.3	700	Decision Tree Estimator (MICE)	2.21047	0.45975	0.49880
0.3	1000	Iterative Imputer Default (MICE)	1.22895	0.68164	0.13432

Continued on next page

Table 3 table. – continued from previous page

Missingness Rate	Size	Method	MSE	R^2 score	Imputation Time
0.3	1000	Lasso Estimator (MICE)	1.22705	0.68120	0.02796
0.3	1000	Linear Regression Estimator (MICE)	1.24017	0.68070	0.11005
0.3	1000	Random Forest Imputer	1.38601	0.63882	4.34155
0.3	1000	Gradient Boosting Estimator (MICE)	1.41260	0.62981	14.24822
0.3	1000	Random Forest Estimator (MICE)	1.48474	0.61687	44.20826
0.3	1000	XGBoost Estimator (MICE)	1.60492	0.59805	25.63662
0.3	1000	LGBM Estimator (MICE)	1.62922	0.58674	8.91731
0.3	1000	XGBoost Imputer	1.61276	0.58114	1.98166
0.3	1000	KNN Imputation	1.74469	0.54594	0.94198
0.3	1000	Decision Tree Estimator (MICE)	2.22682	0.46219	0.69175
0.3	5000	Lasso Estimator (MICE)	1.22330	0.64413	0.07514
0.3	5000	Iterative Imputer Default (MICE)	1.22418	0.64401	0.29105
0.3	5000	Linear Regression Estimator (MICE)	1.22637	0.64394	0.25227
0.3	5000	Gradient Boosting Estimator (MICE)	1.38934	0.60003	67.15375
0.3	5000	Random Forest Imputer	1.37130	0.59695	21.70527
0.3	5000	XGBoost Imputer	1.40821	0.58688	3.08761
0.3	5000	Random Forest Estimator (MICE)	1.53912	0.57013	211.76849
0.3	5000	LGBM Estimator (MICE)	1.54462	0.56889	11.92893
0.3	5000	XGBoost Estimator (MICE)	1.57671	0.55071	49.22171
0.3	5000	KNN Imputation	1.67830	0.52590	18.97883
0.3	5000	Decision Tree Estimator (MICE)	2.22454	0.43648	4.33380

Appendix 2. Python code - Simulation study

```
# Random Forest
class RandomForestImputer:
    def __init__(self, n_estimators=100):
        self.models = {}
        self.n_estimators = n_estimators

    def fit_transform(self, X):
        random_state = random.randint(1, 1000)
        X = np.array(X, dtype=float)
        X_imputed = np.copy(X)
        for col in range(X.shape[1]):
            missing = np.isnan(X[:, col])
            if np.any(missing):

                not_missing = ~missing
                y = X[not_missing, col]
                X_train = X[not_missing, :]
                X_test = X[missing, :]

                X_train = np.delete(X_train, col, axis=1)
                X_test = np.delete(X_test, col, axis=1)

                # Handle NaN in predictors
                imputer = SimpleImputer(strategy="mean")
                X_train = imputer.fit_transform(X_train)
                X_test = imputer.transform(X_test)

                # Train Random Forest
                model = RandomForestRegressor(
                    n_estimators=self.n_estimators,
                    random_state=random_state
                    )
                model.fit(X_train, y)
                X_imputed[missing, col] = model.predict(X_test)
                self.models[col] = model

        return X_imputed

# XGBoost
class XGBoostImputer:
    def __init__(self, n_estimators=100):
        self.models = {}
        self.n_estimators = n_estimators

    def fit_transform(self, X):
        random_state = random.randint(1, 1000)
```

```
        X = np.array(X, dtype=float)
        X_imputed = np.copy(X)
        for col in range(X.shape[1]):
            missing = np.isnan(X[:, col])
            if np.any(missing):

                not_missing = ~missing
                y = X[not_missing, col]
                X_train = X[not_missing, :]
                X_test = X[missing, :]

                X_train = np.delete(X_train, col, axis=1)
                X_test = np.delete(X_test, col, axis=1)

                # Handle NaN in predictors
                imputer = SimpleImputer(strategy="mean")
                X_train = imputer.fit_transform(X_train)
                X_test = imputer.transform(X_test)

                # Train XGBoost
                model = XGBRegressor(
                    n_estimators=self.n_estimators,
                    random_state=random_state,
                    use_label_encoder=False,
                    eval_metric='rmse'
                    )
                model.fit(X_train, y)
                X_imputed[missing, col] = model.predict(X_test)
                self.models[col] = model

        return X_imputed


def introduce_missingness(df, missing_rate=0.1, mechanism="MCAR"):

    if isinstance(df, np.ndarray):
        df = pd.DataFrame(df)

    np.random.seed(random.randint(1, 1000))
    df_missing = pd.DataFrame(df.copy())
    n_samples, n_features = df_missing.shape

    # total number of missing cells
    total_cells = n_samples * n_features
    total_missing = int(np.round(missing_rate * total_cells))

    missing_count = 0

    if mechanism == "MCAR":
```

```
        # randomly choose missing cells
        while missing_count < total_missing:
            row = np.random.randint(0, n_samples)
            col = np.random.randint(0, n_features)
            if pd.isna(df_missing.iloc[row, col]):
                continue
            df_missing.iloc[row, col] = np.nan
            missing_count += 1

    elif mechanism == "MAR":
        # set threshold as the mean of the first column
        threshold = df.iloc[:, 0].mean()

        # set missingness based on the threshold, but not in first column
        valid_cols = [col for col in df_missing.columns if col != df.columns[0]]
        while missing_count < total_missing:
            row_candidates = df_missing[df_missing.iloc[:, 0] > threshold].index
            if len(row_candidates) == 0:
                break
            row = np.random.choice(row_candidates)
            col = np.random.choice(valid_cols)
            if pd.isna(df_missing.loc[row, col]):
                continue
            df_missing.loc[row, col] = np.nan
            missing_count += 1

    elif mechanism == "MNAR":
        # set missingness based on the values of each feature itself
        valid_cols = list(df_missing.columns)
        while missing_count < total_missing:
            col = np.random.choice(valid_cols)
            threshold = df[col].mean()  # threshold is the mean of the column itself
            row_candidates = df_missing[df_missing[col] > threshold].index
            if len(row_candidates) == 0:
                break
            row = np.random.choice(row_candidates)
            if pd.isna(df_missing.loc[row, col]):
                continue
            df_missing.loc[row, col] = np.nan
            missing_count += 1

    else:
        raise ValueError("Invalid missingness mechanism")

    return df_missing

# perform multiple imputation
def perform_multiple_imputation(missing_data, imputer, n_imputations=5):
    start_time = time.time()
```

```
    imputed_datasets = []

    for _ in range(n_imputations):
        imputed_data = imputer.fit_transform(missing_data)
        imputed_datasets.append(imputed_data)
    end_time = time.time()
    duration = end_time - start_time
    return imputed_datasets, duration

# combine multiple imputations results
def combine_imputation_results(imputed_datasets, original_data, missing_mask):
    scores = []

    for imputed_data in imputed_datasets:
        score = mean_squared_error(
            original_data[missing_mask], imputed_data[missing_mask]
            )
        scores.append(score)
    return scores, np.mean(scores), np.var(scores)

# evaluate regression model
def evaluate_regression_model(imputed_data, target_data):
    X = imputed_data
    y = target_data

    # Split data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
        )

    # train a simple linear regression model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # evaluate with cross-validation and test set
    cross_val_scores = cross_val_score(model, X_train, y_train, cv=5,
                                       scoring='mean_squared_error')
    test_score = mean_squared_error(y_test, model.predict(X_test))
    test_r2 = r2_score(y_test, model.predict(X_test))

    return {
        "cross_val_mean_mse": np.mean(cross_val_scores),
        "test_mse": test_score,
        "test_r2": test_r2
    }

# Experiment
def run_experiment(
        missingness_types, missing_rates, sizes,
```

```
    imputers, n_imputations=5, n_repeats=5
    ):
all_results = []

for repeat in range(n_repeats):
    print(f"Starting repeat {repeat + 1}/{n_repeats}...")
    results = []

    for size in sizes:
        X, y = make_regression(n_samples=size, n_features=5, noise=0.1)

        for missingness_type in missingness_types:
            print(f"Starting missingness type {missingness_type}...")
            for missing_rate in missing_rates:
                # ampute dataset
                start_missingness = time.time()
                missing_data = introduce_missingness(X, missing_rate, missingness_type)

                end_missingness = time.time()
                missingness_duration = end_missingness - start_missingness

                missing_mask = np.isnan(missing_data)

                for imputer_name, imputer in imputers.items():
                    # perform multiple imputations
                    imputed_datasets, imputation_duration = perform_multiple_imputation(
                        missing_data, imputer, n_imputations
                        )

                    # evaluate imputations
                    start_evaluation = time.time()
                    scores, mean_score, variance_score = combine_imputation_results(
                        imputed_datasets, X, missing_mask
                        )
                    end_evaluation = time.time()
                    evaluation_duration = end_evaluation - start_evaluation

                    # train and test regression model for each imputed dataset
                    regression_results = []
                    for imputed_data in imputed_datasets:
                        regression_result = evaluate_regression_model(imputed_data, y)
                        regression_results.append(regression_result)

                    # aggregate results
                    mean_cross_val_mse = np.mean(
                        [res["cross_val_mean_mse"] for res in regression_results]
                        )
                    mean_test_mse = np.mean(
                        [res["test_mse"] for res in regression_results]
```

```
                            )
                        mean_test_r2 = np.mean(
                            [res["test_r2"] for res in regression_results]
                            )

                        # Results
                        results.append({
                            "repeat": repeat + 1,
                            "size": size,
                            "missingness_type": missingness_type,
                            "missing_rate": missing_rate,
                            "imputer": imputer_name,
                            'scores': scores,
                            "mean_score": mean_score,
                            "variance_score": variance_score,
                            "mean_cross_val_mse": mean_cross_val_mse,
                            "mean_test_mse": mean_test_mse,
                            "mean_test_r2": mean_test_r2,
                            "missingness_time": missingness_duration,
                            "imputation_time": imputation_duration,
                            "evaluation_time": evaluation_duration,
                            "total_time": missingness_duration +
                            imputation_duration + evaluation_duration
                        })

            all_results.extend(results)

    results_df = pd.DataFrame(all_results)
    return results_df

imputers = {
    "Iterative Imputer Default (MICE)":
    IterativeImputer(max_iter=10),
    "Decision Tree Estimator (MICE)":
    IterativeImputer(estimator=DecisionTreeRegressor(), max_iter=10),
    "Gradient Boosting Estimator (MICE)":
    IterativeImputer(estimator=GradientBoostingRegressor(), max_iter=10),
    "Random Forest Estimator (MICE)":
    IterativeImputer(estimator=RandomForestRegressor(), max_iter=10),
    "XGBoost Estimator (MICE)":
    IterativeImputer(estimator=XGBRegressor(), max_iter=10),
    "Linear Regression Estimator (MICE)":
    IterativeImputer(estimator=LinearRegression(), max_iter=10),
    "Lasso Estimator (MICE)":
    IterativeImputer(estimator=Lasso(), max_iter=10),
    "LGBM Estimator (MICE)":
    IterativeImputer(estimator=LGBMRegressor(), max_iter=10),
    "KNN Imputation":
    KNN(k=5),
```

```
    "Random Forest Imputer":
    RandomForestImputer(),
    "XGBoost Imputer":
    XGBoostImputer(),
}

# Parameters
missingness_types = ["MCAR", "MAR", "MNAR"]
missing_rates = [0.1, 0.15, 0.2, 0.25, 0.3]
sizes = [100, 300, 500, 700, 1000, 5000]
n_imputations = 5
n_repeats = 10 # Number of repeats for accuracy

experiment_results = run_experiment(
    missingness_types, missing_rates, sizes, imputers, n_imputations, n_repeats
    )

# save results
experiment_results.to_csv("imputation_experiment_results.csv", index=False)
```

Appendix 3. Python code - Case Study

```
import numpy as np
import pandas as pd
import random
import time
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error, r2_score
from fancyimpute import KNN, IterativeImputer
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import Lasso
from lightgbm import LGBMRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, cross_val_score

data = pd.read_csv('cars.csv')
from sklearn.preprocessing import OrdinalEncoder
# Encode categorical values
encoder = OrdinalEncoder()
categorical_cols = ['make', 'fuel', 'body_type', 'gearbox', 'defects']
data[categorical_cols] = encoder.fit_transform(data[categorical_cols])

# Random Forest
class RandomForestImputer:
    def __init__(self, n_estimators=100):
        self.models = {}
        self.n_estimators = n_estimators

    def fit_transform(self, X):
        random_state = random.randint(1, 1000)
        X = np.array(X, dtype=float)
        X_imputed = np.copy(X)
        for col in range(X.shape[1]):
            missing = np.isnan(X[:, col])
            if np.any(missing):

                not_missing = ~missing
                y = X[not_missing, col]
                X_train = X[not_missing, :]
                X_test = X[missing, :]

                X_train = np.delete(X_train, col, axis=1)
                X_test = np.delete(X_test, col, axis=1)
```

```
                # Handle NaN in predictors
                imputer = SimpleImputer(strategy="mean")
                X_train = imputer.fit_transform(X_train)
                X_test = imputer.transform(X_test)

                # Train Random Forest
                model = RandomForestRegressor(
                    n_estimators=self.n_estimators,
                    random_state=random_state
                    )
                model.fit(X_train, y)
                X_imputed[missing, col] = model.predict(X_test)
                self.models[col] = model

        return X_imputed

# XGBoost
class XGBoostImputer:
    def __init__(self, n_estimators=100):
        self.models = {}
        self.n_estimators = n_estimators

    def fit_transform(self, X):
        random_state = random.randint(1, 1000)
        X = np.array(X, dtype=float)
        X_imputed = np.copy(X)
        for col in range(X.shape[1]):
            missing = np.isnan(X[:, col])
            if np.any(missing):

                not_missing = ~missing
                y = X[not_missing, col]
                X_train = X[not_missing, :]
                X_test = X[missing, :]

                X_train = np.delete(X_train, col, axis=1)
                X_test = np.delete(X_test, col, axis=1)

                # Handle NaN in predictors
                imputer = SimpleImputer(strategy="mean")
                X_train = imputer.fit_transform(X_train)
                X_test = imputer.transform(X_test)

                # Train XGBoost
                model = XGBRegressor(
                    n_estimators=self.n_estimators,
                    random_state=random_state,
                    use_label_encoder=False,
                    eval_metric='rmse'
```

```
                    )
                model.fit(X_train, y)
                X_imputed[missing, col] = model.predict(X_test)
                self.models[col] = model

        return X_imputed

# perform multiple imputations
def perform_multiple_imputation(missing_data, imputer, n_imputations=5):
    imputed_datasets = []

    for _ in range(n_imputations):
        imputed_data = imputer.fit_transform(missing_data)
        imputed_datasets.append(imputed_data)
    return imputed_datasets

# evaluate regression model
def evaluate_regression_model(imputed_data, target_column):
    X = imputed_data.drop(columns=[target_column])
    y = imputed_data[target_column]

    # pplit data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
        )

    # train a simple linear regression model
    model = LinearRegression()
    model.fit(X_train, y_train)

    # evaluate
    cross_val_scores = cross_val_score(
        model, X_train, y_train, cv=5, scoring='root_mean_squared_error'
        )
    test_score = mean_squared_error(y_test, model.predict(X_test))
    test_r2 = r2_score(y_test, model.predict(X_test))

    return {
        "cross_val_mean_mse": -np.mean(cross_val_scores),
        "test_mse": test_score,
        "test_r2": test_r2
    }

# experiment
def run_experiment_2(original_data, imputers, target_column, n_imputations=5):

    data = original_data.to_numpy()

    results = []
```

```
    for imputer_name, imputer in imputers.items():

        # perform multiple imputations
        start_imputation = time.time()
        imputed_datasets = perform_multiple_imputation(data, imputer, n_imputations)
        end_imputation = time.time()
        imputation_duration = end_imputation - start_imputation

        # train and test regression model for each imputed dataset
        regression_results = []
        for imputed_data in imputed_datasets:
            imputed_df = pd.DataFrame(imputed_data, columns=original_data.columns)
            regression_result = evaluate_regression_model(imputed_df, target_column)
            regression_results.append(regression_result)

        # aggregate results
        mean_cross_val_mse = np.mean(
            [res["cross_val_mean_mse"] for res in regression_results]
            )
        mean_test_mse = np.mean(
            [res["test_mse"] for res in regression_results]
            )
        mean_test_r2 = np.mean(
            [res["test_r2"] for res in regression_results]
            )

        # results
        results.append({
            "imputer": imputer_name,
            "imputation_time": imputation_duration,
            "mean_cross_val_mse": mean_cross_val_mse,
            "mean_test_mse": mean_test_mse,
            "mean_test_r2": mean_test_r2
        })

    results_df = pd.DataFrame(results)
    return results_df

# Imputers
imputers = {
    "Iterative Imputer Default (MICE)":
    IterativeImputer(max_iter=10),
    "Decision Tree Estimator (MICE)":
    IterativeImputer(estimator=DecisionTreeRegressor(), max_iter=10),
    "Gradient Boosting Estimator (MICE)":
    IterativeImputer(estimator=GradientBoostingRegressor(), max_iter=10),
    "Random Forest Estimator (MICE)":
    IterativeImputer(estimator=RandomForestRegressor(), max_iter=10),
    "XGBoost Estimator (MICE)":
```

```
    IterativeImputer(estimator=XGBRegressor(), max_iter=10),
    "Linear Regression Estimator (MICE)":
    IterativeImputer(estimator=LinearRegression(), max_iter=10),
    "Lasso Estimator (MICE)":
    IterativeImputer(estimator=Lasso(), max_iter=10),
    "LGBM Estimator (MICE)":
    IterativeImputer(estimator=LGBMRegressor(), max_iter=10),
    "KNN Imputation":
    KNN(k=5),
    "Random Forest Imputer":
    RandomForestImputer(),
    "XGBoost Imputer":
    XGBoostImputer(),
}

data['price_log'] = np.log(data['price'])
data = data[['fuel', 'body_type', 'gearbox', 'mileage', 'defects', 'doors',
             'price_log', 'weight', 'years', 'engine_size', 'power']]
# run experiment
experiment_results = run_experiment_2(
    data, imputers, target_column="price_log", n_imputations=5
    )
print(experiment_results)
```

Appendix 4. R code - Little's MCAR test

```
library(naniar)

# Load data
data <- read.csv("cars.csv")

# Perform Little's MCAR test
result <- mcar_test(data)
print(result)

if (result$p.value > 0.05) {
  cat("Fail to reject null hypothesis - data is MCAR")
} else {
  cat("Reject null hypothesis - data is not MCAR")
}
```