SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

Faculty of Informatics and Information Technologies

Reg. No.: FIIT-5212-98938

Analysis and Replacement of Missing Values in Machine Learning

Bachelor thesis

Study programme: Informatics
Study field: Computer Science
Training workplace: Institute of Informatics, Information Systems and Software Engineering
Thesis supervisor: Ing. Peter Krammer

Bratislava 2022 **Olesia Melnychyn**

Slovak University of Technology in Bratislava
Ústav informatiky, informačných systémov a softvérového inžinierstva

Faculty of Informatics and Information Technologies
Academic year: 2021/2022
Reg. No.: FIIT-5212-98938

BACHELOR THESIS TOPIC

Student: **Olesia Melnychyn**
Student's ID: 98938
Study programme: Informatics
Study field: Computer Science
Thesis supervisor: Ing. Peter Krammer
Head of department: doc. Ing. Valentino Vranić, PhD.

Topic: **Analýza chýbajúcich hodnôt a ich dodefinovanie v strojovom učení**

Language of thesis: English

Specification of Assignment:

V súčasnosti výrazne narastá objem dostupných údajov, ako aj potreba ich spracovania a analýzy. Dostupné údaje však obvykle nedosahujú optimálne vlastnosti; často sú zaťažené výraznými chybami, resp. dochádza k výpadkom, či prerušeniam meraní čoho dôsledkom sú chýbajúce hodnoty v datasetoch. V niektorých prípadoch dokonca chýbajúce hodnoty nie sú náhodne, ale sú dané technickými obmedzeniami jednotlivých meracích zariadení (napr. z dôvodu obmedzených rozsahov meracích prístrojov). Chýbajúce hodnoty tak predstavujú problém, ktorý často komplikuje úspešné nasadenie neurónových sietí, či iných typov modelov. Existujú rozličné techniky na narábanie, či doplnenie chýbajúcich hodnôt; väčšina z nich má však svoje výhody a nevýhody, resp. je vhodná len pre špecifické prípady. Cieľom práce je navrhnúť univerzálnu hybridnú metódu kombináciou viacerých vhodne zvolených a nastavených existujúcich metód (resp. ich modifikácií), tak aby došlo k zvýšeniu presnosti dodefinovaných hodnôt, a použité čiastkové metódy aby tak vzájomne kompenzovali svoje nedostatky. Analyzujte metódy narábania a dodefinovania chýbajúcich hodnôt a vypracujte ich prehľad. Navrhnite hybridnú metódu na dodefinovanie chýbajúcich hodnôt na základe existujúcich metód (optimálne aby kombinovala výhody jednotlivých metód a potláčala nevýhody). Implementujte a otestujte funkčnosť a stabilitu navrhnutej metódy; vykonajte ladenie metódy. Validujte metódu na viacerých dátových množinách (so simulovanými chýbajúcimi hodnotami). Vypočítajte štandardné miery chyby (root mean squared error, relative absolute error) dodefinovaných hodnôt, a vyhodnoťte metódu.

Length of thesis: 40

Deadline for submission of Bachelor thesis: 16. 05. 2022
Approval of assignment of Bachelor thesis: 23. 11. 2021
Assignment of Bachelor thesis approved by: doc. Ing. Valentino Vranić, PhD. – Study programme supervisor

Čestne vyhlasujem, že som túto prácu vypracovala samostatne, na základe konzultácií a s použitím uvedenej literatúry.

V Bratislave, 14.05.2022

Olesia Melnychyn

Acknowledgements

First of all, I would like to thank my supervisor Ing. Peter Krammer for mentoring me during this work. The knowledge I gained from our consultations helped me a lot in writing it. I also very much appreciate your eagerness to help with any problem that might have appeared on the way.

I would also like to express my gratitude to my friends that stay with me in the most stressful moments and bear my character.

Most of all I would like to thank my parents for all their support on this long way and for making this all possible. Also great thank you to them for helping me with wise advice in difficult situations both at school and in life. Knowing that they will never let me down and believe in me keeps motivating me to try even harder.

Olesia Melnychyn

Anotácia

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný program: Informatika

Autor: Olesia Melnychyn
Bakalárska práca: Analýza chýbajúcich hodnôt a ich dodefinovanie v strojovom učení
Vedúci bakalárskej práce: Ing. Peter Krammer
Máj 2022

Potreba kompletných a spoľahlivých dát rastie spolu s rozvojom oblasti, akou je Umelá Inteligencia. Nanešťastie je z rôznych dôvodov takýto stav dát veľmi zriedkavý, čo môže byť spôsobené chybou hardvéru, softvéru, ako aj chybou používateľa. Keďže cieľom Strojového Učenia a iných oblastí Inteligentnej Analýzy Údajov je vytvoriť vhodný model predikcie, je dôležité mať dostatočný dataset. Preto vymazanie záznamov s chýbajúcimi hodnotami nemusí byť postačujúcim riešením. Kvôli tomuto sa používajú rôzne metódy dodefinovania hodnôt, ale neexistuje dokonalý prístup, keďže každý má svoje výhody a nevýhody a navyše aj prípady ich použitia.
Z tohto dôvodu sa táto práca zameriava na vytvorenie hybridnej metódy dodefinovania hodnôt, ktorá dokáže kombinovať výhody už existujúcich techník, s cieľom maximalizovania ich výkonnosti. V tomto bakalárskom projekte sa bližšie pozrieme na už existujúce metódy imputácie, ich výhody a nevýhody, ako aj na metódy ich evaluácie. Následne navrhneme a implementujeme ich kombináciu a vyladíme parametre, ktoré určujú akú metódu v konkrétnom prípade použiť. Pri rozhodovaní zoberieme do úvahy aj štatistické vlastnosti hodnôt v stĺpci.

Annotation

Slovak University of Technology Bratislava
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES
Degree course: Informatics

Author: Olesia Melnychyn
Bachelor's Thesis: Analysis and Replacement of Missing Values in Machine Learning
Supervisor: Ing. Peter Krammer
2022, May

The need for complete and reliable data grows along with the development of such field as Artificial Intelligence. Unfortunately, it is a rare case when the data is such for different reasons. It can be because of the hardware or software error, as well as a user one. Since the aim of Machine Learning or other areas of Intelligent Data Analysis is to create a proper predicting model, it is significant to have a sufficient dataset, and therefore deleting records with missing values is hardly a good solution. Imputation methods are used for this reason, but there is no perfect approach since each has its own benefits and drawbacks.
Therefore, this work aims to create a hybrid imputation method that can combine the benefits of already existing imputation techniques to maximize their performance. So, in this work, we take a close look at existing imputation methods, as well as design and implement their combination, as well as tune the parameters that define which method to use in the particular case.

Contents

List of Acronyms

CCA Complete Case Analysis

CCPP Combined Cycle Power Plant

CMC Concept Most Common

CoD coefficient of determination

Corr Pearson correlation coefficient

DLI Datawig deep learning imputation methods

Em Expectation-maximization

FKMI Fuzzy K-means Clustering

GUI Graphical User Interface

HIMP Hybrid imputation method

HPM-MI Hybrid prediction model with missing value imputation

IDE Integrated Development Environment)

JAMA Java Matrix Package

Java ML Java Machine Learning Library

JSAT Java Statistical Analysis Tool

KNN K-Nearest Neighbors

KMI K-means Clustering Imputation

LLSI Local Least Squares Imputation

LOCF Last Observation Carried Forward

MAE mean absolute error

MAPE mean absolute percentage error

MAR Missing at Random

MCAR Missing Completely at Random

MCMC Markov chain Monte Carlo

MC Most Common Method

MICE Multiple Imputation by Chained Equations

ML Machine Learning

MNAR Missing Not at Random

MPE Mean Percentage Error

MSE Mean-Squared Error

NOCB Next Observation Carried Backward

PCA Principle Component Analysis

RAE Relative absolute error

RF Random Forest

RMSE root mean-squared error

RRSE root relative squared error

RSE relative squared error

R^2 R-squared

SVD Singular Value Decomposition

SVDI Singular Value Decomposition Imputation

SVMI Support Vector Machines Imputation

SvrFcmGa Fuzzy C-means SvrGa imputation

UTC Coordinated Universal Time

WKNN Weighted Imputation with K-Nearest Neighbor

1 Introduction

It is a well-known fact that data in the contemporary world is invaluable. Information, along with the ability to analyze it and get some results from it, is a powerful tool for different goals, such as providing a market with a demanded product or service, or, from a different view, making it demanded by creating a target-consumer advertisements. The other field where data analysis plays a significant role is in medicine: there already exist tools that make an attempt to help doctors to diagnose and prescribe treatment for various diseases. The further application is science: with the help of gathered data, scientists try to spot some trends in a variety of occurrences and predict their future behavior, such as the frequency of natural phenomenons, or the most topical - the growth of diseases like COVID-19.

As a result, it comes as no surprise that data is in such demand. Furthermore, these gathered records should be good enough for creating a reliable prediction model, namely, they should be accurate and complete. Otherwise, it can have a detrimental effect on the outcomes of the model trained on them, that is the predictions will be unrealistic and, therefore, useless. Unfortunately, the case of data being perfect is very rare, and, thus, each data analyst faces a challenge of finding a way to alleviate the impact of such data on the future estimator.

For this reason, there exist various techniques how to handle these incomplete data and make them more appropriate for future investigations. For instance, the part of data that is not proper can be removed and only the suitable part is used for training, but evidently, it is not the most sensible decision since all the data is important. Due to this, a common way of handling missing values is to define them. In some cases, these gaps in a dataset can be easily replaced with the average or the most common value, but more often there is a need to create

an additional model to predict the missing values. There is an expansive range of such techniques, but all of them have their advantages and disadvantages and are advised to use only in specific use cases when their main assumptions are met.
Hence, this work aims to find a combination of these methods - the so-called hybrid imputation method - in order to reduce the effect of the drawbacks and take full advantage of the benefits of each one.
The first chapter makes a closer examination of existing methods of handling missing values, as well of the tools, such as libraries and frameworks, which make it easier and more time and memory efficient. The second chapter is going to explain the design along with the implementation of the suggested solution, its main aspects, and the parameters used for making a choice of method used in the particular case. The third chapter summarises the work done and evaluates the performance of the design method, and besides that, makes a look into the future of the result of this work.

2 Problem Analysis

This chapter provides an overview of the problem of handling missing values in datasets. First of all, it describes the nature of such values by classifying them into three different types and explains the reasons why such occurrences take place. Then follows an examination of two main ways of managing these gaps divided into sections, namely removal of data and imputation of missing values. Both approaches comprise further ranges of techniques used, for instance, data removal is divided into listwise deletion, pairwise deletion, and dropping variables. Moreover, in each section, we make a more detailed inspection including the main assumptions and applications along with the pros and cons of the most commonly used methods. Also, at the end of this chapter, we examine common statistical measures which are used to describe the dataset and also common metrics used for evaluation the performance of the designed methods, as well as their meaning and significance.

2.1 Problem of Missing Values

Unfortunately, not all the data is complete and, as a result, every data study faces a problem of missing values. This may happen because of a software, hardware, human error, or also, in case of some questionnaires or forms, the user might want not to provide some part of information. Usually, these values are represented by *?, NaN, Na, null* or an empty cell, mainly depending on the programming language [1]. It is now a common challenge in the world of data science, which may not only affect the accuracy of predictions but also completely invalidate the whole study because of the selection bias [2]. And although some of the existing models, such as CART, are resistant to missing values [3], still it is a problem

which should be tackled, since most already implemented models or methods in libraries, such as skLearn *etc.*, do not support gaps in a dataset and will throw an error when coming across one. Furthermore, it is often a case that we do not want to reduce an already small dataset and lose any significant data for analysis and training a final model, since, thus, the estimator might become irrelevant.

Missing data may appear due to different reasons and, hence, have a different impact on the study results. For this reason, there are outlined three different types of missing values (*see Figure 1*).

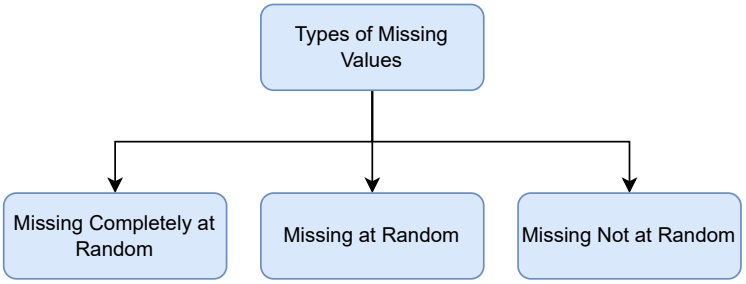


Figure 1: Types of missing values

The first one is *Missing Completely at Random (MCAR)*. This one describes the case when the value is absent independently from both values which were meant to be collected and a series of other observations [2]. This type is used as a main assumption for the majority of datasets in machine learning and is an ideal situation since the data remains unbiased and with a minimal effect on the statistical power of the designed models.

The second type of missing data is *Missing at Random (MAR)*. This one is used to define the case when the other variables are related to the missingness of a value, but in the meantime, it does not depend on its hypothetical value [4]. Likewise, on the occasion of the first type, it is safe to delete the samples with the occurrences of missing values of this type. But there is a problem that such type can rarely be proved to be the case.

Missing Not at Random (MNAR) is the last category and all the data which is considered to be neither missing completely at random nor missing at random

belong to this type [2]. This case may cause some complications for the data scientists. The reason for this is that it cannot be safely deleted, as it may lead to a bias in a model. The only way to avoid this bias is to use an additional model for predicting missing values, but it is appropriate to mention that it does not necessarily give better results.

2.2 Handling Missing Data

Since there exist different problems caused by mistakes or gaps in datasets, as well as types of missing values, the ways of their handling also diverse drastically. The method used for this purpose usually depends not only on the type of missing data (MCAR, MNAR or MAR described above) but also on other characteristics of the observations. For example, some of the techniques focus on numerical data (which may be both continuous or discrete), the others are suitable only for categorical types. At the same time, these methods are divided into separate groups – the ones which substitute the missing value, called *value imputation methods*, and those that delete a whole sample in case one or more of the values are absent. There is also another way called *encoding missingness*. This one is about setting the model to ignore the missing values, mainly by introducing a further value which will be treated as an extra category [3]. Judging from the principal, it is obvious that this one can only be used for categorical data and, therefore, is out of the scope of this work. But the first two techniques of handling missing values will be examined more profoundly further in this chapter.

2.2.1 Data Removal

The data removal method is the easiest way of handling missing values. The principle is to delete the entire predictor and/or the whole sample which contains at least one missing value. While using this approach, one should be aware of the consequences of such data transformation, as it may lead to biasing the model and

worsening the results. These aftereffects are related to the sample of dataset left after applying the removal as well as the amount of data available for training a model.

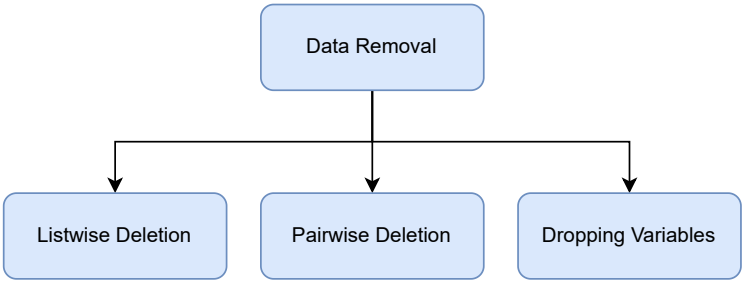
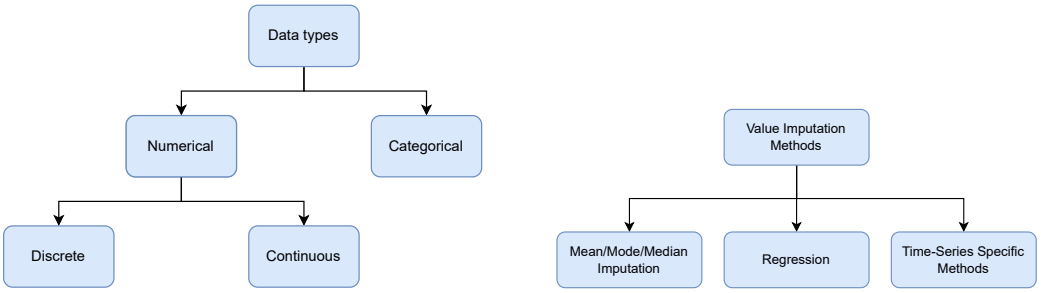


Figure 2: Data Removal

Although removing data is pretty straightforward, there also exist different ways to apply this method to a dataset. These techniques can be seen in Figure 2. The most common is *listwise deletion*, or, as it is also referred to *Complete Case Analysis (CCA)*. This approach is about deleting each row that has at least one missing value and a complete, but smaller, the dataset is an output. Listwise deletion is mainly based on the assumptions that the missing data hold a small part of the dataset (not more than 5-6%) and their deletion will not bias the model. Additionally, the values should be of missing-at-random type [5]. Since these requirements are hardly ever met, it is usually not recommended to apply this method during data preprocessing.

The other practice is *pairwise deletion*. In this case, missing values are simply ignored in an original dataset and the analysis is carried out on separate samples consisting of different sets of observations and having the rows with missing values removed. Thus, this way maximizes the size of the dataset available for training an estimator. But it also has its drawbacks, namely, different parts of a model will be trained on different number of samples, which, in consequence, will complicate the interpretation [4].

The last method is *dropping variables*. The principle is to delete the whole column in case some part of the data is missing. Of course, it is not the best way to



(a) (b)

Figure 3: Data Types (a) and Value Imputation Methods (b)

handle missing data, but might be appropriate in a case when the most part (for instance, 60%) of the values in a column is missing and also the values in the column is not significant [4].

2.2.2 Value Imputation Methods

The other way of handling data is using *value imputation methods*. These are used to get the most out of the dataset and not to reduce its size, since it may have bad consequences on the whole study. In a nutshell, these methods analyze the statistics and/or relationships between the predictors and observations in order to be able to predict the value which is missing.
For different data types (Figure 3a) various of methods (Figure 3b) can be used, and this is to be described in this section.

Mean, Mode, Median

The basic imputation methods are filling gaps with *mean*, *mode* or *median*. In case of all of them, missing values are replaced with the mean, mode, or median in accordance. All three are easy and fast to perform, which is a significant advantage of these methods. The assumption for using the mean, mode, and median value imputation methods is that the missing data is of MAR type. Also, they are pretty feasible to use mostly in cases when the probability that the values lacking are

close to the average, most frequent, or halfway of the range [5]. Otherwise, it might affect a the statistical power of a model trained over such data, as they do not take into account the relationships among the columns in a dataset. Moreover, it fails to appreciate variance (there might occur an over-representation of a value) and biases the model. Hence, these methods are not for general use, but mainly for the cases when a small part (around 5-6% of data) is missing [5, 6]. Notwithstanding, the mode value imputation technique can be used for all data types, while the other two are not applicable for categorical data, they are only meant for numerical one.

Regressions

Another value imputation method is with using some type of regression. This one comprises many different ways, which can be seen in Figure 4. This one is reasonably considered to give better results than previous methods, as it takes the relationships between the predictors and values to be predicted into consideration and, moreover, the predictions may be done based on not only one, but more variables. For these types of methods, one should firstly fit a regression estimator by explicitly separating the variable to be estimated and the predictors based on which to predict it. The result of such a process is the coefficients, which then are used for calculating missing values [6].

The most known type of regression and the one we will describe first is *Linear Regression*. It is noteworthy that this regression type is a subitem of *Polynomial Regression* since it is the same polynomial but with a degree equal to 1. Firstly, one should select the best predictor for a missing value using the correlation matrix. Then after training a model, we receive the equation coefficients, which afterward are used for predicting the data lacking. The formula of this regression is the following:

$$\mathrm{y} \;=\; \beta_0 \;+\; \beta_1 x \;+\; \epsilon, \tag{1}$$

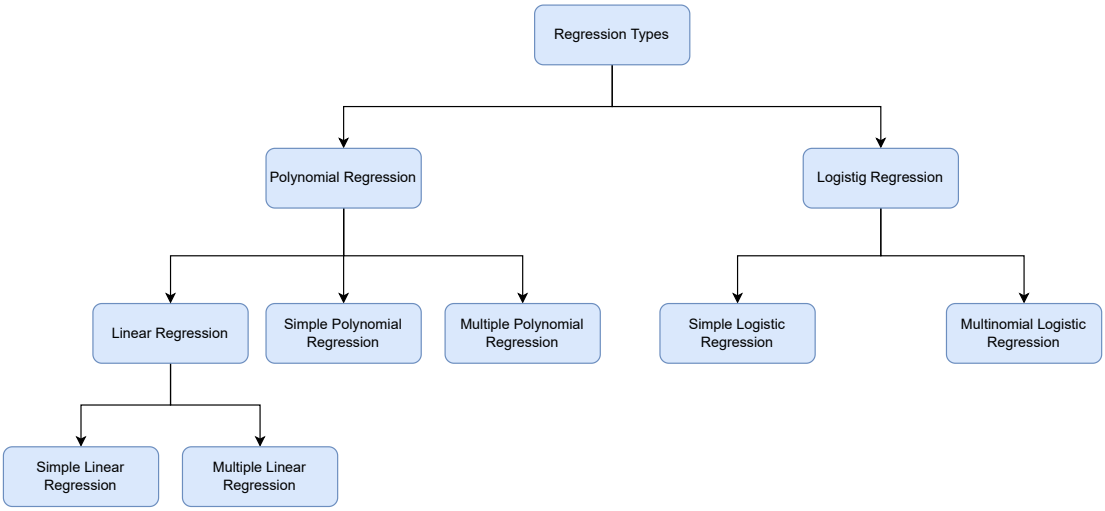


Figure 4: Types of Regression

where y is dependent variable, x - predictor, β_0 - constant term, β_1 - coefficient for an explanatory variable and ϵ - error of the model, also called residuals.
There is also a modified type of this regression called *Multiple Linear Regression*, which makes a prediction based on more than one column in a dataset. In this case, the formula is as follows:

$$y = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n + \epsilon, \tag{2}$$

where y is dependent variable, x_p - predictors, β_0 - constant term, β_p - coefficient for each explanatory variable and ϵ - error of the model.
Additionally, it should be pointed out that the linear type of regression can only be used with numerical data. But although both variants described are considered to be good ways of substituting invalid data, these methods also have their disadvantages, which in some cases might cancel out their benefits. As a result of the prediction on other variables, the standard error within predicted values might be decreased or even lost. Also, it should be taken into consideration that this type of regression attempts to find a linear relation between the variables, whereas it might not be the actual type of relationships in the given dataset [4].

The other type is *Polynomial Regression*. This one makes an attempt to find curvilinear dependency since it is rarely the case when it is just linear. The definition is as follows:

$$y \ = \ \beta_0 \ + \ \beta_1 x^1 \ + ... + \beta_n x^n \ + \ \epsilon, \tag{3}$$

where y_i is dependent variable, x_i - predictor, β_0 - constant term, β_p - coefficient for an explanatory variable, n - the degree of the regression and ϵ - error of the model.

This type might be a more feasible way of handling missing data, but when using such a model for predicting, the degree of the equation should be chosen carefully. Since too low a degree might lead to underfitting and too high one might result in overfitting [7].

After having described polynomial and multiple regressions, there appears a question if those two can be somehow combined since it is rare when the dependent variable depends on one predictor or more than one but of a first degree. For this case, there exists *Multiple Polynomial Regression*. The formula of this regression has the form:

$$y \ = \ C_0 \ + \ \alpha_1 x_1^1 \ + ... + \alpha_n x_1^n \ + ... + \ \beta_1 x_p^1 \ + ... + \beta_n x_p^n \ + \ \epsilon, \tag{4}$$

where y is dependent variable, x_p - predictor, C_0 - constant term, α_p, β_p - coefficients for an explanatory variable, n - the degree of the regression and ϵ - error of the model.

It is obvious that this model might provide far better results since it considers different aspects, but among the drawbacks of this method is *multicolinearity*, which occurs when the predictors are interdependent and, as a result, the coefficients might deviate [8].

The other type of regression we are going to look at is *Logistic Regression*. This method is used for predicting a column with discrete values based on other predictors. It is noteworthy that, unlike the previous type, it can also be used for categorical data. Since the purpose of using this method is to predict the probabil-

ity of some value, the most used outcome is binary, such as yes/no, true/false *etc.* The definition of this regression:

$$P(y) = \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}} \tag{5}$$

where y_i is dependent variable, $P(y_i)$ - probability of dependent variable, x_i - predictor, C_0 - constant term, $beta_0$ and $\beta_p 1$ - coefficients.

There also exists a generalization called *Multinomial Logistic Regression* which opts for solving problems with more than two possible outcomes. This means that more categories are possible, and, therefore, this regression is commonly applied in cases of classification problems [9]. In this case, the probability is calculated for every possible category:

$$\begin{aligned} P(y=1) &= \frac{e^{\beta_1 x}}{1+\sum_{k=1}^{K-1} e^{\beta_k x}} \\ &\dots \\ P(y=K-1) &= \frac{e^{\beta_{K-1} x}}{1+\sum_{k=1}^{K-1} e^{\beta_k x}} \end{aligned} \tag{6}$$

where y is dependent variable, $P(y=k)$ - probability of dependent variable being of k category, x - predictor, K - number of categories and β_k - coefficient.

There are more methods for value imputation of regression kind which are not going to be described, but the main ones were already covered so to explain the principle of how this type works.

Time-Series Specific Methods

The last methods we are going to mention are about missing values related to the *longitudinal data*. This type of data need a special handling of data since it might have a particular tendency over time.

One of the ways to deal with such data is *Linear Interpolation*. This method tries to create a curve by predicting data points within the given range. It can be a reasonable choice for the data which is recorded relatively often, but is not suitable

for those which are gathered seasonally [4].
The other approaches are *Last Observation Carried Forward (LOCF)* and *Next Observation Carried Backward (NOCB)*. These cases take into consideration the same observations at different time stamps. But it is noteworthy that both LOCF and NOCB might bias the model if the data has some obvious tendency [4].

2.3 Descriptive Statistics

In most cases of making a decision on which imputation method should be used, descriptive statistics comes into mind. As an example, we can take mean imputation, which is considered to be appropriate if the values in the column are close to the mean and the standard deviation is relevantly small. In this section, we are going to describe such measures that describe the data in a dataset.
The first and the most common measure is the mean. It is also referred to as average and its main problem is that it is relatively more influenced by extreme values [10]. Its formula looks as follows:

$$mean \quad = \quad \frac{x_1 \ + ... + \ x_n}{n} \tag{7}$$

The second common statistic is the median, which represents a half-point of the data. It is also called 50th percentile and considered to better represent the data than the mean since it is less sensitive to outliers [10]. It is also 2nd (middle) quartile. Both percentiles and quartiles are *quantiles* which split data in 100 and 4 equal parts respectively. In order to compute quantile, data should be ordered from 1 to n, where n is a number of values. Then, the qth quantile is calculated with interpolation between two values on both sides of the rank, which is equal to $q * (n + 1)$.
Variance (s^2), and Standard Deviation (s) are next measures which are pretty often used for describing dataset. Both are closely related since standard deviation

is calculated by obtaining the square root of variance. Speaking of the characteristics of these values, they describe how much data differ from the mean and therefore are sensitive to outliers [10]. The formulas for both these measures are the following:

$$variance \quad = \quad \frac{(x_1 - \overline{x})^2 \ + ... + \ (x_n - \overline{x})^2}{n} \tag{8}$$

$$standard\ deviation \quad = \quad \sqrt{variance} \tag{9}$$

Kurtosis and skewness are further statistical measures of the data distribution represented by formulas (10) and (11). The first one describes how many values are in tails. In case when a big part of data is placed in tails, which are then called *heavy*, kurtosis is positive [10]. Otherwise, when it is negative, tails are called *light*. The second measure describes the difference between these tails, namely how much one is heavier or lighter than the other [10].

$$kurtosis \quad = \quad \frac{1}{n}\frac{(x_1 - \overline{x})^3 \ + ... + \ (x_n - \overline{x})^3}{s^3} \tag{10}$$

$$skewness \quad = \quad \frac{1}{n}\frac{(x_1 - \overline{x})^3 \ + ... + \ (x_n - \overline{x})^4}{s^4} \tag{11}$$

These are not the only existing measures of descriptive statistics, but the main ones which we are going to consider when making a choice of imputation method used in a particular case.

2.4 Metrics

As with everything in Machine Learning, the usage of value imputation methods begs a question of whether it is worth applying them and if they perform better

than a simple listwise deletion. For this purpose, different metrics are used. In case of categorical data, they are *True Positive*, *False Positive*, *True Negative*, *False Negative*, which are then used for calculating accuracy, precision, recall *etc.* These are the main ways to evaluate the results. But since this work deals with numerical data, methods for evaluation described in this section will diverse (*see Figure 5*). In case of this type, it is hard to say whether the prediction is correct or incorrect, it is more about how far the prediction is from the expected value, usually referred to as *error*. There are many formulas for calculating different errors and their meaning are diverse as well.

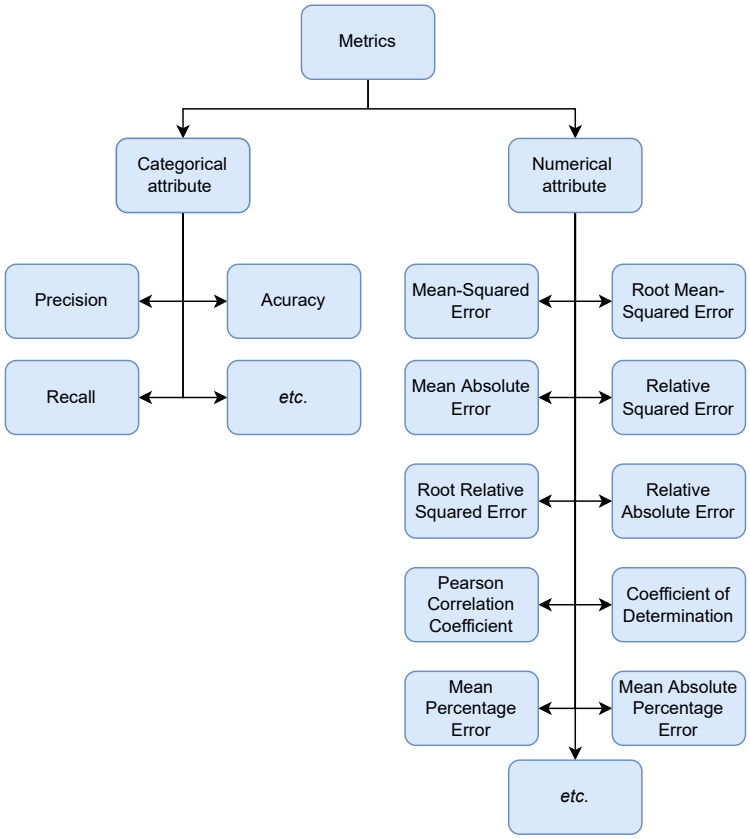


Figure 5: Metrics

Before getting down to the errors themselves, it would be appropriate to add what the letters in the formulas mean. In all formulas $p_1, p_2, ..., p_n$ stand for the values

of predictions, $a_1, a_2, ..., a_n$ represent the actual values to compare with. Also in all errors below, n is for the number of observations used. At the same time $\overline{a}$ has a different meaning in some formulas, namely in all cases, except correlation coefficient, it stands for the mean over the actual data, while in case of the mentioned exception it stands for the mean over predicted values.

The most used error is *Mean-Squared Error (MSE)*. Also, it is the easiest method to evaluate the results [11]. The definition can be seen in formula (12). The smaller the value is, the closer predictions are to the real values. The disadvantage of this error is that it exaggerates the effect of outliers, namely, it punishes larger errors more. Also, it is worth mentioning that the order of MSE unit is higher than the error unit since it is squared. In order to get down to the same one, *root mean-squared error (RMSE)* is used [12]. This metric is described by fromula (13).

$$MSE = \frac{(p_1 - a_1)^2 + ... + (p_n - a_n)^2}{n} \tag{12}$$

$$RMSE = \sqrt{\frac{(p_1 - a_1)^2 + ... + (p_n - a_n)^2}{n}} \tag{13}$$

The other way to evaluate the performance of an estimator is *mean absolute error (MAE)*. It is just an average of the individual error, but without taking sign into consideration. Unlike mean-squared error, this one treats all error sizes evenly and, as a result, is not that much sensitive to outliers [11]. The definition of MAE has the form:

$$MAE = \frac{|p_1 - a_1| + ... + |p_n - a_n|}{n} \tag{14}$$

In some cases, such as when an error of 10 for a prediction of 100 has the same importance as 0.1 for 1, absolute errors are pointless, since they will exaggerate the value of a bigger error. Hence, on such occasions, it is appropriate to use an

alternative – relative errors. Speaking of this kind of error, it is considered that a feasible model should not have the value of more than one, which means that it is better than a trivial predictor and is reasonable to use. An example of such a metric is *relative squared error (RSE)*. In the formula (15), an error is related to what the value could be if a simple predictor (in our case it is a mean over training data) was used and, thus, normalizing the error [11]. Accordingly to the case of MSE and RMSE, *root relative squared error (RRSE)* can be used. The last is described by formula (16).

$$RSE = \frac{(p_1 - a_1)^2 + ... + (p_n - a_n)^2}{(a_1 - \overline{a})^2 + ... + (a_1 - \overline{a})^2} \tag{15}$$

$$RRSE = \sqrt{\frac{(p_1 - a_1)^2 + ... + (p_n - a_n)^2}{(a_1 - \overline{a})^2 + ... + (a_1 - \overline{a})^2}} \tag{16}$$

Another error that can also be used for measuring the performance of an estimator is *Relative absolute error (RAE)*. It is just a sum of absolute errors normalized in the same way as RSE, namely with an error which would be if a simple predictor which gives average values was used for predicting [11]. As mentioned above, a good model will result in a value closer to zero, whilst the one with poor performance will have a metric value greater than one. The definition of this metric is as follows:

$$RAE = \frac{|p_1 - a_1| + ... + |p_n - a_n|}{|a_1 - \overline{a}| + ... + |a_1 - \overline{a}|} \tag{17}$$

Pearson correlation coefficient (Corr), can also be useful for evaluating the results of a model which deals with numerical data. It expresses a linear relationship between two variables by taking the covariance and standard deviation of each of them into consideration, which can be seen in formula (18). Its value is in a range from -1 to 1. The extremes stand for the cases when the results correlate

perfectly, either negatively, in case of -1, or positively, in case of 1. The half-point, 0, means that there is no correlation between the predictions and the actual values. Obviously, for reasonable models, negative values of the coefficient, as well as zero, should not occur and, unlike other measures, the higher the value is, the better the model is considered to be. Also as to this kind of performance metric, it is appropriate to add that it is slightly diverse from all others [11]. It is because it shows whether the values move in the same direction and at the same speed, and, therefore, there might be a situation when the predictions are a multiplied by actual values, which is not what is expected from a model, but the correlation coefficient will show that it is a best fit.

$$\begin{aligned} Corr &= \frac{S_{PA}}{\sqrt{S_P S_A}}, \quad \text{where} \quad S_{PA} = \frac{\sum_i (p_i - \overline{p})(a_i - \overline{a})}{n-1}, \\ & S_P = \frac{\sum_i (p_i - \overline{p})^2}{n-1}, \quad S_A = \frac{\sum_i (a_i - \overline{a})^2}{n-1} \end{aligned} \tag{18}$$

R-squared (R^2), or *coefficient of determination (CoD)*, is the next way to evaluate an estimator. This metric represents the proportion between the variance of predicted values and the variance of actual ones, as it is shown in (19). While the correlation coefficient describes the relationship between expected values and results, R-squared shows how much the variance of predictions relates to the variance of real ones. This means that when, for instance, R^2 is equal to 0.25, then about a quarter of the estimated values can be explained by the original ones [13].

$$CoD = 1 - \frac{(a_1 - p_1)^2 + ... + (a_n - p_n)^2}{(a_1 - \overline{a})^2 + ... + (a_n - \overline{a})^2} \tag{19}$$

The last error we are going to describe is *Mean Percentage Error (MPE)*. This metric measures the accuracy of the prediction model in the units of percent. It is also commonly used and works well for datasets that do not contain zeros [14]. Also, for the cases with values below zero, it might be appropriate to use *mean absolute percentage error (MAPE)*, which performs measurement independently from the sign of a value. The definitions of both MPE and MAPE are in formulas (20)

and (21) respectively.

$$MPE = \frac{100}{n} \sum_i \frac{a_i - p_i}{a_i} \tag{20}$$

$$MAPE = \frac{100}{n} \sum_i \frac{|a_i - p_i|}{|a_i|} \tag{21}$$

Having examined so many measures, there appears a question which one is the most reliable to use. But there is no simple answer to which measure is the best since there is no one-size-fits-all. Moreover, it is not easy to say which metric should be used in a given situation, as that can only be determined by examining the further application itself, namely what is a purpose, what is a cost of each error in this particular case, and which error is more preferred to be kept as low/high as possible. For the reason that, for example, root-squared error punishes outliers more than small errors, whilst absolute measures do not do it, it only depends on a study which one would be the most relevant to use. Also, there is a question of units, meaning that using root-mean-squared error will take the evaluation results to the same dimensionality as the values to be predicted. In the meantime, relative errors evaluate an estimator from the simple predictor's point of view and answer the question of whether and to what extent it is reasonable to use the model [11]. But anyway, the best model gives the best predictions independently from which measurements were used to assess its performance. And since there is no particular definition of the application of the method design during this project, it was decided that the most appropriate metrics would be the most used ones, namely RMSE and MAPE.

2.5 Tools (Frameworks and Libraries)

Many different tools in various programming languages are used to perform predictions. Mostly, machine learning programs are written in Python. The most significant reason for this is a great choice of libraries primarily targeting this language, such as NumPy, SciPy, and scikit-learn, and also such powerful frameworks as TensorFlow and Apache Spark. Also, simplicity and conciseness play an important role in making the final decision in favor of this language [15]. Since it is an interpreted language and the types are defined during the run time, the drawback of Python is its speed.

The other option is Java. This one is usually chosen due to its benefits in enterprise development, typing, and also great speed [16], which is why it was decided to write this project in Java. What deters data scientists from Java is its inappropriateness for visualization.

Also, a well-known language is MatLab. Its greatest advantage is loads of built-in functionality for Machine Learning. Moreover, it goes along with its IDE, which simplifies work. Namely, although not being the most beautiful, the visualization in MatLab still attracts beginners because of being simple to use and interactive. Moreover, there exist many toolboxes for machine learning and statistics. But the disadvantage of this language is a high memory usage as well as relatively low speed of execution due to the fact that it is, like Python, an interpreted one [17]. But at the same time, MatLab is faster than Python because of the great optimization in a variety of toolboxes, such as Statistics and Machine Learning, Optimization, and Parallel Computing Toolboxes.

Nevertheless, programming languages are not the only thing to be decided when starting with artificial intelligence, the others include libraries, platforms, and frameworks, some of which we are going to describe further in this section.

Weka

Weka is one of the most common software used for data science in Java. It is free and written in Java, therefore, compatible with any platform. Moreover, this software has its own GUI. It is a good choice for data preparation and analysis, as well as modeling and other data mining algorithms. Weka is especially good for classification problems, but can also be used for other purposes, such as regression, feature selection, prediction of time series, clustering *etc.* [18].
This software also supports visualization, although it has some limitations. Among the drawbacks of Weka there is a fact that it is not constantly updated with state-of-the-art techniques [19]. Therefore, data scientists might sometimes need to use further tools along with Weka. The other disadvantage is limited GUI documentation, which at the same time is not getting updated with the growth of Weka, making work more complicated. The other con is the problem of memory: when working with big datasets, the program might throw OutOfMemoryError [19]. To prevent this from happening, a data scientist usually has to divide the dataset into smaller parts and use *incremental learning*. The other solution is using *sparse matrix* instead of *dense* ones. Another disadvantage is that some functions are not accessible from GUI and as a result there might be a need to switch to the command line to perform some tasks [19].

Scikit-Learn

Scikit-Learn is one of the most commonly used libraries for ML in Python. One of its biggest advantages is its functionality which is good for analysis and modeling [20]. Scikit-learn is mainly known for its supervised learning algorithms, namely Decision Trees, Bayesian methods, SVM and Linear Regression, but it also supports unsupervised ones. As well as Weka, it supports classification, clustering, regression *etc.* The other great benefit is its high compatibility with NumPy, Matplotlib, and SciPy, due to the reason that it was built upon them [20]. It has also a simple user interface and a high level of learnability. Moreover, it gets constantly

updated because of a great number of contributors and the immense international community [20]. The documentation is also pretty readable. The main drawback is that it is not really suitable for deep learning and also GPU data processing is not supported by this library [20].

Apache Commons Math

Apache Commons Math is a library used in Java for statistical analysis as well as ML. One of the advantages is that it is well documented. It implements most common functionality, such as curve fitting and clustering algorithms. Moreover, it provides data scientists with statistical methods, namely different types of regression, statistical tests, and descriptive statistics methods as well as calculating a correlation coefficient and covariance [21]. The main disadvantage is not so great community, which makes it more complicated to solve a problem or understand how it works.

JSAT

Java Statistical Analysis Tool (JSAT) is another library for Java language, which contains a great collection of algorithms for ML. It was partly written for self-education and the intention was also to make it more intuitive and faster than Weka. As well as Apache Commons Math, it does not have any external dependencies. It is considered to be relatively fast in case of small and medium datasets. One of the advantages is that parallel execution is supported by a great part of functionality [22]. It implements many types of classifiers, different algorithms for clustering, regression models as well as methods for natural language processing problems. It also provides data scientists with methods for data preprocessing, such as visualization, descriptive statistics *etc.*

Java ML

Java Machine Learning Library (Java ML) is also a library for ML in Java programming language. It has a great functionality for this purpose, such as algorithms for classification problems, regression, clustering, feature selection, and data preprocessing. One of the advantages is that it works with most of the file types with the requirement that it has one record per line [23]. Java ML is also fully documented and provides a user with tutorials and examples of usage. Moreover, to simplify the work of the data scientist, all algorithms strictly follow a simple standard interface [24].

JAMA

Another Java-oriented library is Java Matrix Package (JAMA). It provides functionality for performing linear algebra calculations. The aim is to provide simple and intuitive functionality for beginners which want to work with matrices. The library can manipulate with the real, dense matrix. JAMA provides functionality for elementary matrix operations, such as addition, multiplication, transpose, *etc.*, as well as for more complex computations. For instance, there are five fundamental decompositions available, such as SVD, LU, QR, Cholesky, symmetric, and non-symmetric eigenvalue decompositions. Furthermore, JAMA is capable of solving equations and calculating derived quantities (determinant, inverse, *etc.*) [25].

2.6 Related Works

It comes as no surprise that this work is not the only one to attempt to create a hybrid imputation method, to reap the benefits of the existing ones. Therefore, in this section, we would like to briefly point out some of such works. The studies that are described below, in contrast to our goal, are not focused only on numerical data, most aim to imputation the mix of data types.

A Hybrid Imputation Method for Multi-Pattern Missing Data: A Case Study on Type II Diabetes Diagnosis [26]

This study proposes a Hybrid imputation method (HIMP), which goal is to tackle all three types of missing values, namely MCAR, MAR and MNAR described earlier in this chapter.

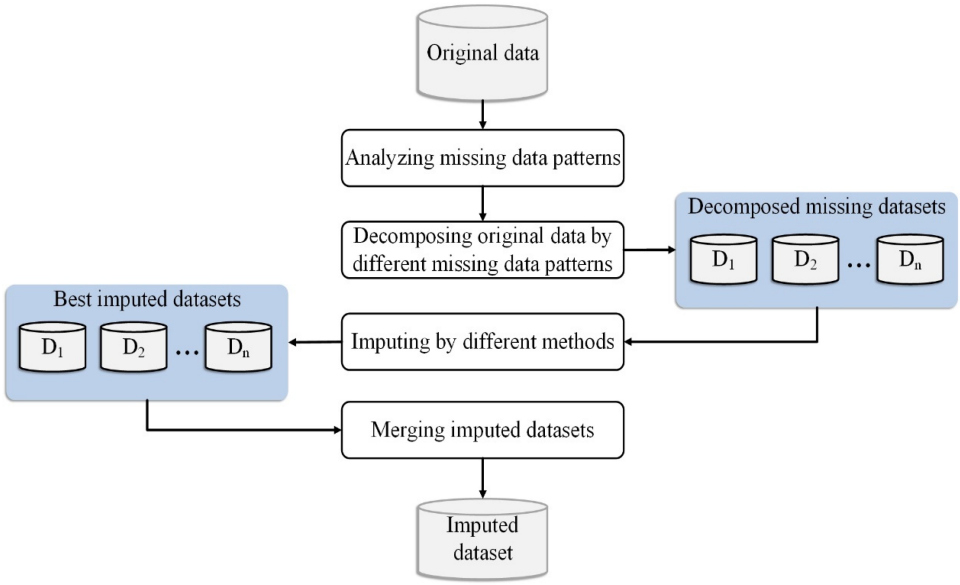


Figure 6: Hybrid Imputation Method for Multi-Pattern Missing Data [26]

The flow of the algorithm proposed can be split into four main parts: analysis, decomposition, imputation, and merge (see Figure 6). So, firstly, the dataset is analyzed in order to detect different missing value types. Then the data is decomposed into three parts accordingly to the pattern o missing data. After that, each of the dataset parts is imputed using a different approach. For the lacking data of MNAR type, appropriate constant global values, such as "non-determined" or "unknown", are used for imputation. Imputation of missing values of MAR and MCAR pattern is applied to a resulted dataset from the previous approach. Then, HIMP uses single imputation methods, such as K-Nearest Neighbors (KNN) and *Hot-Deck*, for missing completely at random values and multiple imputation methods, namely *Markov chain Monte Carlo (MCMC)*, Multiple Imputation by

Chained Equations (MICE), and *Expectation-maximization (Em)* for values missing at random. In both cases, the results of each method are evaluated by using different classifiers and the dataset generated by the best approach for each type are merged so to form a final dataset.
The evaluation shows that the proposed method is more effective in imputation data in the dataset with all three missing values types than other methods, namely MICE, KNN and *Fuzzy C-means SvrGa imputation (SvrFcmGa)*.

A Hybrid Modified Deep Learning Data Imputation Method for Numeric Datasets [27]

As well as in our case, this study aims to create a hybrid imputation method. But the approach, called *RF-DLI*, is based on deep learning and therefore uses diverse imputation techniques, namely *Random Forest (RF)* and *Datawig deep learning imputation methods (DLI)*.
Random Forest is a well-known supervised learning algorithm that consists of many decision trees and uses bagging and random feature selection. DLI a software package based on deep learning, which is used for imputation values of heterogeneous data types (a combination of both numerical and categorical data).
The process of data imputation in this study can be divided into three general steps. At first, Random Forest is used for determining the effect of each attribute on the imputed value. In this case RF is used as a feature selection algorithm, where the importance of the features is based on the calculated impurity value. In this case, the variance was chosen among others (except for this one, RF also uses Gini index and information gain) to work out the impurity. The effect of an attribute is determined as follows: the more the attribute reduces impurity, the more it is important. Then the most significant attributes (50% of the features) are chosen to be used during imputation. And finally, missing values are imputed by DLI with the use of these attributes.

According to the paper, it gives generally better results than other approaches, namely *KNN*, *MICE*, mean imputation method and *Principle Component Analysis (PCA)*.

Hybrid Prediction Model with Missing Value Imputation for Medical Data [28]

This work introduces *Hybrid prediction model with missing value imputation (HPM-MI)* (see Figure 7). Although the goal of this algorithm is prediction, it contains a hybrid imputation method which is worth mentioning. The main idea is to use the best among 11 imputation approaches chosen by simple K-means.

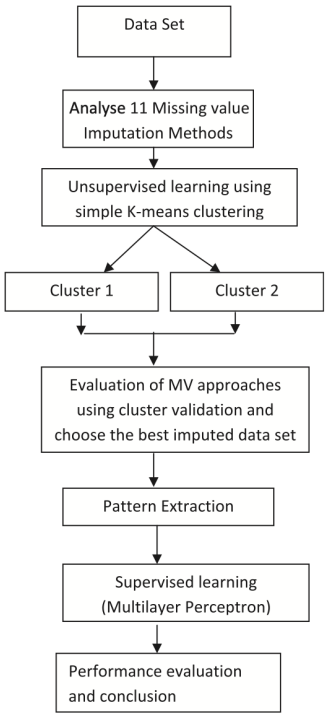


Figure 7: Hybrid Prediction Model with Missing Value Imputation [28]

At first, all 11 imputation methods are applied to the original dataset creating 11 new samples. The methods which are used for imputation are: case deletion, Most

Common Method (MC), Concept Most Common (CMC), KNN, Weighted Imputation with K-Nearest Neighbor (WKNN), K-means Clustering Imputation (KMI), Fuzzy K-means Clustering (FKMI), Support Vector Machines Imputation (SVMI), Singular Value Decomposition Imputation (SVDI), Local Least Squares Imputation (LLSI) and Matrix Factorization. Then simple K-means clustering is then used on dataset samples and then the results are validated and the best approach, the one with the least number of incorrectly classified instances, is chosen.

2.7 Proposed Design

After having a pretty close look at the main aspects of the problem of missing values, the design of the hybrid method to be implemented in the second part of this work becomes more clear. Regarding the programming language, it was decided to write in Java in order to reap the benefits of its computational speed. Also, the decision was laid on writing in pure Java using open-source libraries, such as JSAT, Apache Commons Math, *etc.* As to the methods used in the final hybrid one, it was set to combine all that is described above, namely mean, median, and mode imputation, different types of regression, and also techniques appropriate for time-series datasets. Moreover, if possible, we would like to improve the performance by using methods available in the libraries mentioned above. Speaking of combining existing methods, we will take descriptive statistics of a column in which missing value occurred, as well as its relationship to the predictors, into consideration to decide which method is the most appropriate in the particular case. For measuring the performance, it is planned to implement all the metrics described in the corresponding section, but the main goal is to maximize the values of RMSE, RAE and MAPE.

Of course, it is not a the final word and the design might be extended or changed during implementation. For example, we might use some additional methods for predicting values or further metrics for performance evaluation. Also, although the

main target is time-specific data, we would like to implement predicting missing values based on more than one predictor as a further improvement of the design, since values in some datasets, apart from time, may also depend on other aspects, such as geographic coordinates, height, *etc.*

3 Solution Description

This chapter describes the proposed solution to the problem, that was analyzed in the previous section. First of all, it provides a rough overview of the solution and its main items, datasets that were used for the implementation, input and output formats, as well as reasons for making choices. Along with the datasets we are going to explain the preprocessing part. After that, the main block of the implementation is described – in this section, we are going to outline the architecture to better understand the links between objects and the overall structure of the design. Further, the main objects will be described to explain the main principle. At last, we will list the libraries and tools used for the implementation, along with other technical details.

3.1 Datasets and Data Preprocessing

During implementation, we were using different datasets so as not to run into overfitting. Among these datasets are:

1. Combined Cycle Power Plant (CCPP) Dataset
2. Energy efficiency Dataset
3. Dataset from Slovak Hydrometeorological Institute
4. Yellowstone Park Elevation Dataset

Mainly, we were using the last one, since it was usable for the purposes of the multiple imputation (based on more than one predictor), namely in this case the independent variables were time, longitude, and latitude. The data were gathered hourly from 1st Jan, 2016, 00:00 and until 6th Mar, 2019, 11:00. Further in this section we are going to use it as an example for data preprocessing

and will refer to it as a Hydrometeorological dataset. So to work with the Hydrometeorological dataset as with a simple imputation case, we first needed to retrieve the data for different times, but with similar latitude and longitude. Since all the data are divided into different `.csv` files by time (more precisely, each file contains the data tracked at the beginning of an hour), we also had to collect all needed data in one file. In this step, we also dropped some unessential columns to ease our work. For all these tasks, we wrote a script `src/com/company/data/combine.py`. For multiple imputation, there exists a similar script – `src/com/company/data/combine_multiple.py` – this one does not try to find similar latitude and longitude. Also, to make processing better, we decided to reformat the time. Before, the time had ISO standard date format with UTC time, which is `yyyy-MM-dd'T'HH:mm:ss.SSS'Z'`, and we replaced values in this column with number of hours since the beginning date (1st Jan, 2016, 00:00). Then, to be able to test our method we had to simulate missing values. That was done with another script – `src/com/company/data/add_missing_values.py`. As it might be seen, both scripts are written in Python, but there is no strong reason for that, it is only because of the language's simplicity and syntax that makes it suitable for such small problems.

It is worth mentioning the format of input datasets. The program is meant to read or write into only `csv` files. The other point is that since the work aims to impute numerical data, the program does not accept datasets with other types of data than `int` or `double`. Otherwise, it will run into a `NumberFormatException`. This limitation is also related to the tool we are using for reading csv files, namely JSAT that does not support reading of any other data type than those which are listed above. But in terms of defining missing values, the requirements are not so strict: it can be either of the following, or even their combinations: *null*, *na*, *nan* (all three ignoring case), *?*, *0* (if in `config.properties` we define that 0 should be considered as a missing value) and just an empty cell (in csv files - two commas in a row). Also, the complete and incomplete datasets must have the same structure and corresponding records (same line – same record) for the

correct performance evaluation of the implemented hybrid method.

3.2 Hybrid Method Design

In this section, we would like to describe the design of the proposed method and explain the logic behind it. Namely, this section will tackle all three main components of the method – input, output and the method itself (see Figure 8). We will try to explain how the method behaves in different cases and why we defined it to do so.

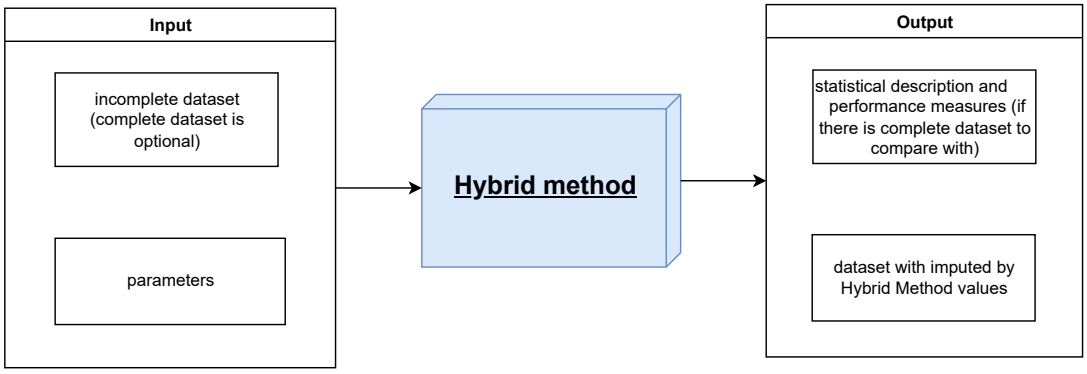


Figure 8: Performance measures of altitude column

3.2.1 Input

As it might be expected, `csv` files are not the only variant of input, more things should be provided by the user before the imputation is run. Such information (further, we will call them parameters) can be provided via console with the user typing the answers to the displayed questions. But since such a procedure is not very user-friendly, there exists another way of passing parameters to the program, namely through configuration file `config.properties`. There, with `input.useConfigValues = true`, the values are taken from a file and the program does not require user input from the console.

Now, a bit more detailed about the parameters. The first thing to be provided to the program is datasets. The complete dataset is optional and is used only for calculating performance measures of the designed hybrid method, but an incomplete one is essential. Then, the index of the predicted column is required. As the hint in the console states, with entering `-1`, all columns, except for the predictors, will be considered as predicted. There is also a check if the index is valid (is not exceeding the number of columns in the dataset.) Otherwise, `IndexOutOfBoundsException` will be thrown. After that, programs need the predictors' indexes. Here, the user can write more than one index and those which are not valid will be skipped. If no predictors are left after validation, then an exception (`InvalidDataException`) with the corresponding message is thrown. Afterward, in case it is an input in the console, the program asks whether to rewrite `config.properties` with provided parameters, so they can be used repeatedly in the future. Then, after the program finishes running, it requires a path to an output file. For how the console input looks like see Listing 1.
The other parameters, such as whether to print all info, whether 0 should be considered a missing value, and whether to save the outcome of the imputation in csv file, *etc.*, are coming from `config.properties` file. More on these can be found in Appendix C: User Guide.

3.2.2 Hybrid Method

The main component of the design is the method itself. This is basically a bunch of different methods taken from a variety of libraries or implemented by ourselves in combination with the conditions at the different places/stages of the program which define what method is going to be used. There are also other aspects, such as different ways of preparing the training dataset and the one to be predicted, *etc.* So, in this part, we would like to describe the flow of the process in the proposed solution and the conditions which define the behavior of the hybrid method. In this section, we will start from the rough overview and further will move to the

```
Enter filename with complete dataset (type "1" to use test
↪  filename, skip if there is not one)
\path\to\complete_dataset
Enter filename with incomplete dataset (type "1" to use test
↪  filename)
\path\to\incomplete_dataset
Number of missing values: 6932
Enter index of the column to be predicted (starting from 0)
↪  (type -1 to impute all dependent values)
-1
Enter index(es) of predictor(s) (starting from 0)
0 1 2
Store predicted and predictors indexes in config file? y/n
y
...
Enter filename of the output file (type "1" to use test
↪  filename)
\path\to\imputed_dataset
```

Listing 1: Console Input

explanation of some parts in more detail.

Main Flow

The main course of the data is pretty simple and is illustrated using a flow chart in Figure 9. At first, after scanning the input and creating a dataset object, the program calculates statistical measures for each column that is going to be predicted. The descriptive statistics for each column include such measures as well-known mean, variance, and standard deviation, and also kurtosis, skewness, Pearson correlation, percentiles, *etc.* This statistics is used further in the program in some if-statements, for instance, when choosing an imputation method to be used for predicting missing value.

Then starts the main cycle, namely, we are traversing the dataset, record by record. So, for each data record, we check if it has any missing values. If not, the pro-

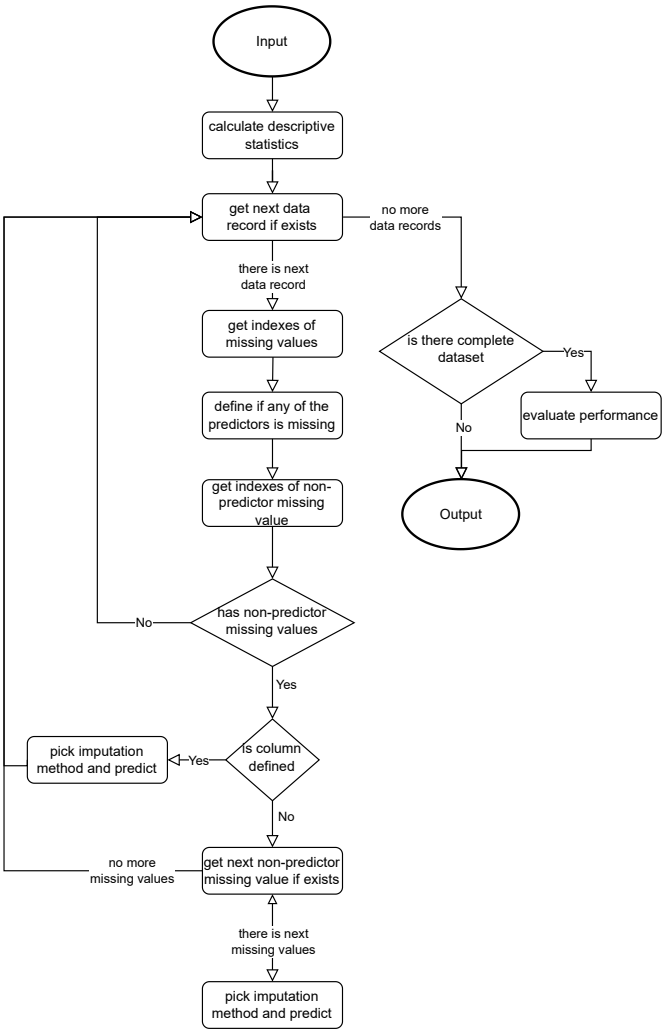


Figure 9: Main Flow

gram will continue traversing the dataset, since in such a case we do not need to predict anything. Otherwise, we identify indexes of missing values (there might be more than just one missing). Then, we determine if none of those is an index of any of the predictors and determine indexes of missing values that are not predictors. After that, if the column to be predicted is defined, we will check if that column is among those indexes and pass it further into the part where an imputation method is chosen and the value is predicted (this will be described in further sections). Otherwise, if a particular is not defined, we will pass all missing

values' indexes (that are not meant to be predictors) separately. In case check for missing predictor gave a positive result, the choice lies between mean and median imputation methods, otherwise, it is a more complicated process which will be described further in this section.
When all possible values in the dataset are imputed, the program continues to the part of saving data into a file and, in case a complete dataset has been provided, evaluates the results.

Multiple and Simple Imputation Methods

Now, we would like to describe the process of choosing between simple and multiple imputation methods. This part of the program flow is depicted in Figure 10. From the viewpoint of the previous flow chart (Fig. 9), we are at the "Pick imputation method and predict" step. More precisely, (a) describes the first part (choosing an imputation method) and (b) – the second (predicting missing value).
So, as it can be seen in the previous figure, we receive the missing value as an input for this process and the first step is to create an object which will hold the main data for imputation. The only logic here is to check how many predictors we have. In case there is at least one missing predictor, we define an empty array of predictors, otherwise, we use the ones that were provided by a user. In case there are one or no predictors in the composed main data object, we opt for simple imputation methods. There, we have to prepare a training dataset at first. For this purpose, we take n data records around the current one with no missing values in the predictor column (if there is a predictor column defined) as well as in the column of the current missing value. At this stage, we also collect data records with missing values that can be imputed along with the current. These are gathered during the search for training data by coming across one that has a gap in the same column as the main missing value. After these two datasets are prepared, we go on to choose the most suitable method. The logic of choosing the right method is going to be described in the next section, so far we will use it as a

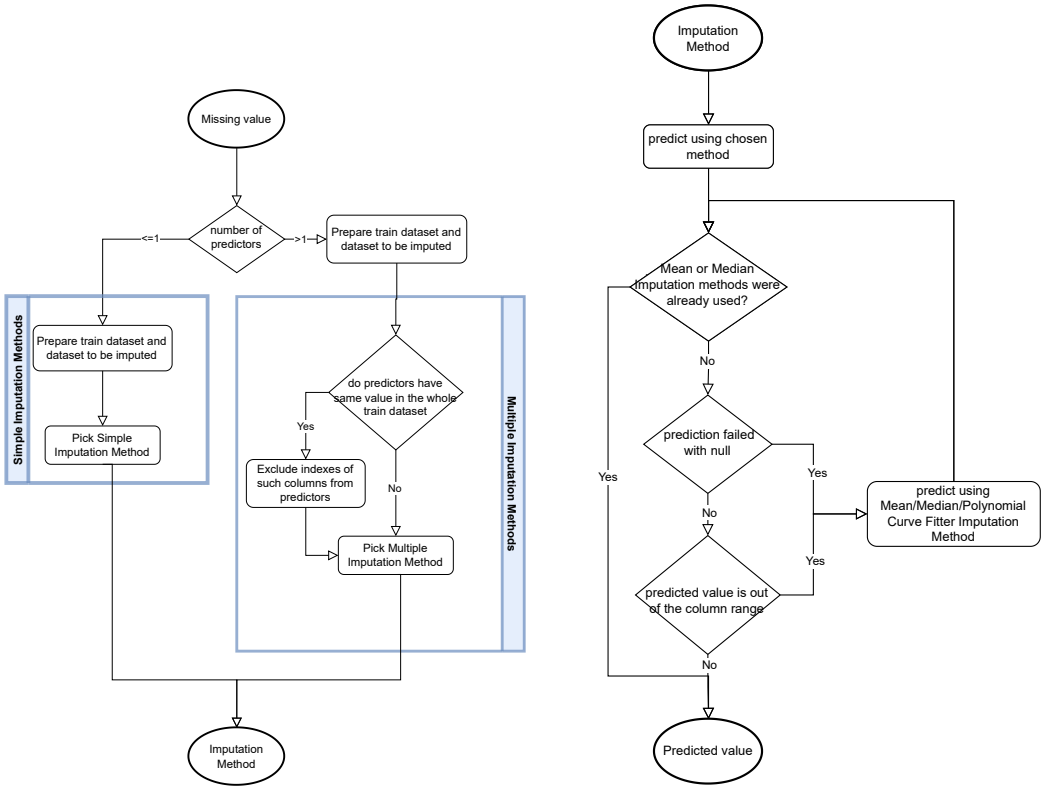


Figure 10: Choosing Simple and Multiple (a) and Predicting Missing Value (b)

black box.

Given there is more than one predictor, we are talking about multiple imputation methods. In this case, preparing two datasets is a bit different. Namely, 80 data records (there is no strong reason for this number, but the higher the number is, the bigger the time complexity is and there is no guarantee that it will improve the results) around the current one are ranked by euclidean distance and the closest 12 comprise training dataset. The distance is calculated as in equation 22 below, where n is a number of predictors, a_i and b_i is a normalized in some way predictor of one, respectively second, data item. In our case, we normalized our predictors to be in a range from 0 to 1 using the formula 23, where a is a predictor value and max_a and min_a are minimal, respectively maximum, values in the column

of a. Depending on configuration in `config.properties`, the data records can also get a weight using equation 24, where d is euclidean distance and ϵ is a small value to prevent dividing-by-zero error.

$$d = \sqrt{(a_1 - b_1)^2 + ... + (a_n - b_n)^2} \tag{22}$$

$$z = \frac{a - min_a}{max_a - min_a} \tag{23}$$

$$w = \frac{1}{\epsilon + d} \tag{24}$$

As well, as in the case of preparation datasets for simple imputation methods, here, we collect data records that make sense to be imputed using the same method and training dataset as the main missing value. Data records with a gap in the same column as the current, having no missing values in predictor columns, and having a smaller euclidean distance than the data record with the biggest one in the training dataset are considered to be suitable in this case. It is worth mentioning, that these data records are not always gathered, since after some testing we have noticed that it worsens the results and, therefore, we decided to do this only in case predicting each value separately would be too time-consuming (in the code, we check if there are more than 1 000 000 missing values in the dataset and if not, we predict one value at a time).

After we build our training dataset for multiple imputation, we check if there is a predictor column with the same values, namely all data records have the same value in the column. If so, we exclude the predictor from predictors, but only for this iteration, meaning the next missing value will have the original set of predictors. Then, we choose an imputation method that can be both simple and multiple.

After we define which method we are going to use for predicting, we try to predict the missing value (Fig. 10b). If it fails (e.g. the predicted value is null), the

program will try to predict it one more time, but this time using Polynomial Curve Fitter (if Multiple Polynomial Regression has been used previously) or simple mean/median imputation method (if that is not the one used before). The outcome of this is a predicted value that is saved in the dataset in place of the missing one.

Choosing Simple Imputation Method

The core of the hybrid algorithm is a selection of an appropriate method to be used for predicting missing values. In case of simple imputation, the choice lies between mean and median imputation methods, linear interpolation, linear regression, polynomial curve fitter, or imputation of the closest value. In order to decide between them, there is a bunch of conditions that are also shown in Figure 11. Since the most computationally efficient methods are to put mean or median, we check whether they would be plausible to use first. The check is based on how close the values in the predicted column vary from the mean, respectively median. As well as in all following conditions, there are predefined thresholds for that.
If neither check for mean nor median was valid, the method that follows is Linear Interpolation. This one is coming from Apache Commons Math (described in section 3.1) and requires values to be constantly increasing. In case the training dataset has all values decreasing, we reverse it and use this method for imputation. When interpolation is also not considered to be appropriate, the program checks whether linear regression or polynomial curve fitter would be useful. From JSAT we used the implementation of the first one, and the second is coming from Apache Commons Math. The checks, in this case, are based on the Pearson correlation coefficient. If neither seems to work, we decide whether imputation of the closest value would be appropriate based on some intervals of standard deviation and Pearson correlation. If not, the default method is selected, namely the mean imputation method.

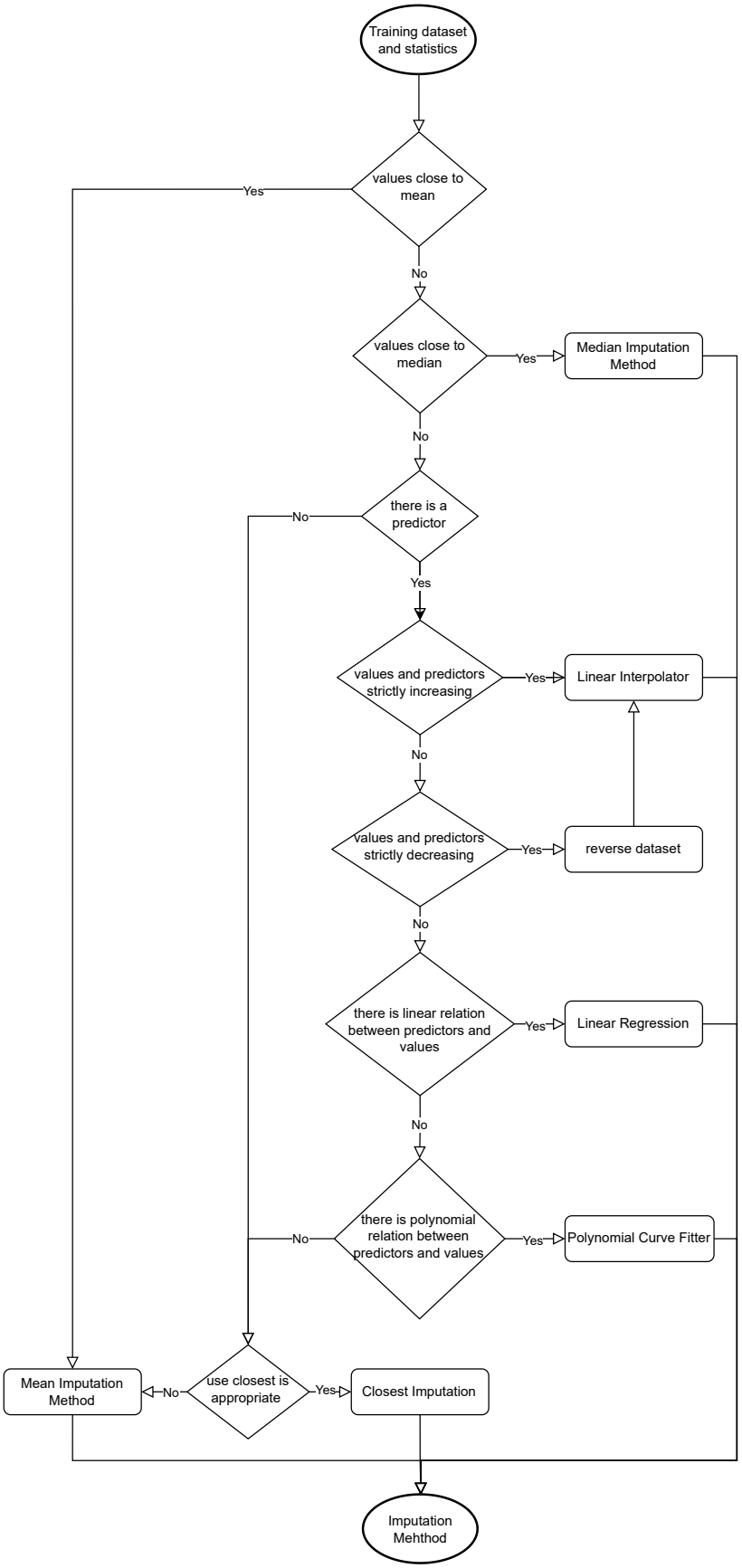


Figure 11: Choosing simple imputation method

Choosing Multiple Imputation Method

In case of multiple regression, the choice is smaller, namely between three methods – multiple linear or polynomial regression, or multi-dimensional imputation method (see Figure 12).

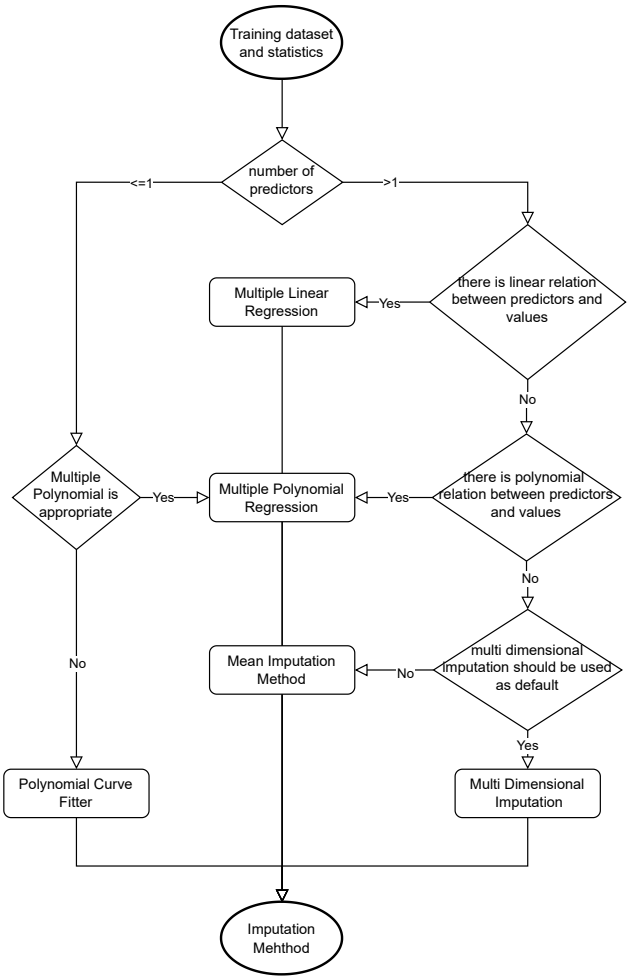


Figure 12: Choosing multiple imputation method

There are two different flows that depend on how many predictors we have, namely one or more. For the note, there is a possibility to have one predictor if we have excluded predictors in the previous step during preparing the dataset (this was already described in this section). If we have one predictor we first check if multiple

polynomial regression would be appropriate. This is done based on values of kurtosis and correlation coefficient, this time we use the one for multiple correlation (see Formula 25). This measure is within a range from 0 to 1, where the higher the value the more credible the linear relationship is. The coefficient of multiple correlation describes the linear relationship between a set of independent variables and a dependent one [29]. Then, if the check did not result in true, we use the default method for this flow, which is polynomial curve fitter.

$$R = \sqrt{(c^T R_{xx}^{-1} c)} \tag{25}$$

$$c = \begin{pmatrix} r_{x_1 y} \\ r_{x_2 y} \\ \vdots \\ r_{x_n y} \end{pmatrix}, \quad R_{xx} = \begin{pmatrix} r_{x_1 x_1} & r_{x_1 x_2} & \dots & r_{x_1 x_n} \\ r_{x_2 x_1} & \ddots & & \vdots \\ \vdots & & \ddots & r_{x_{n-1} x_n} \\ r_{x_n x_1} & \dots & r_{x_n x_{n-1}} & r_{x_n x_n} \end{pmatrix}$$

,where c is a vector of correlations $r_{x_i y}$ between the predictor variable x_i and the dependent variable y (predicted one), R_{xx} stands for the matrix of correlations between independent variables.

For the flow with more than one predictor, the decision-making mechanism is a bit different. The first two imputation methods are basically the same since linear regression is just a special type of polynomial, namely the one with the 1st degree. For deciding whether or not to use one of these methods we also use multiple correlation coefficient.

Multi-dimensional imputation method is a default one (if the `impute.useMultiDimensionalAsDefault` in `config.properties` is `true`, otherwise, the default is mean imputation method) used when neither check for linear, nor polynomial returned `true`. This one is not coming from a library and was implemented during the work on the thesis. The main idea is to predict the missing values based on each predictor separately. In our program, we choose the most appropriate simple imputation method following the algorithm described in the previous subsection for each predictor column. Then, the final predicted value is calculated as a mean

of all values that have been predicted by selected simple imputation methods.

3.2.3 Output

As well as input, the output of the hybrid method has also more than one form, there are three different ones. The first one is the file with imputed values, which was already described above. The other two forms are console output and file `resuls.txt`.
The console output, namely its verbosity, is controlled by the input. In case of more detailed output, the user can see which method along with its own parameters (e.g coefficients for the linear regression) was used for each missing value as well as the value predicted. Moreover, user can see the MAPE of the single prediction. The final evaluation (in case the complete dataset has also been provided) is displayed along with the statistical description of the columns that used to have missing values independently of the verbosity.
The `results.txt` file contains only the final evaluation of the hybrid method.

3.3 Technical Aspects

Now, we would like to explain the technical part in more detail. One of the first things to be mentioned is that the implementation was written mainly in Java (except for the scripts, mentioned in section 3.1, that were written in Python). The reason for that is that it is a strongly typed programming language and also has great speed.

3.3.1 Libraries

Talking about libraries and tools in the implementation, we used their `.jar` files which were added to the classpath. Among these are JSAT, JAMA and Apache Commons Math all mentioned in the section 2.5. Apache Commons Math was used for manipulation with a data matrix representation of the dataset and performing mathematical or statistical operations, as well as some of the imputation methods implemented in this library, namely `GaussianCurveFitter`, `LinearInterpolator`, *etc.* Java Matrix Package was used for different kinds of regressions, namely simple and multiple linear regression, as well as a polynomial. The most used library throughout the code is Java Statistical Analysis Tool. The `SimpleDataSet` class from this library stored the whole dataset (dataset with imitated missing values and complete one to compare the results with), as well as its small instances - training and test datasets for the imputation methods. Besides that, JSAT also takes care of reading and writing data from, respectively into, a file. Furthermore, some of the imputation methods are taken from this library.

3.4 Results

In this section, we would like to describe the results of the proposed hybrid method design. We will also mention the arrangements that were done before running the tests, so to ensure accurate results comparison.

Test set-up

So, at first, we want to outline the organization which took place before running tests. One of the things which are worth mentioning is that the designed hybrid method is able to predict missing values even if there are no predictors (if they are also missing). In such a case, mean or median imputation methods are used.

We had to disable this for the testing since not all the methods we are going to compare with can handle this.
For the testing, we have prepared three datasets, the test data of which can be found in Appendix C. So, the first one we tested on was a dataset from Slovak Hydrometeorological Institute which has three independent and five dependent variables. The second one is the Combined Cycle Power Plant dataset which has only one independent variable and three dependants. And the last one is the Yellowstone Park dataset, which has two predictors and one predicted value.
We tried to compare the results of our hybrid method with the results of each method that was used as its part. The preparation of the training dataset for each missing value was the same for both cases (except for mean and median imputation methods). We will show only some of the results in this section to describe the main point, all of them are in Appendix B.

Test results of Hydrometeorological Dataset

So, the first dataset we are going to compare is the weather one. Here, the predictors are time, longitude, and latitude and the values we were predicting are atmosphere temperature and pressure, relative humidity, wind speed, and direction.
The main trend in almost all the predicted columns was that the performance measures of the hybrid method were a bit better than the results of other methods. For example, for the atmosphere temperature, the evaluation can be seen in Figure 13. In the mentioned diagram, as well as in all others in this section, we showed only the results of the designed hybrid method along with the best and worst methods among others that we compared it with. In case of this predicted value, the worst was the technique using mean and the best was polynomial curve fitter (which used time as a predictor in the second degree). As can be seen, the mean is far worse than the other two, while polynomial curve fitter and hybrid are quite close to each other in all the metrics.

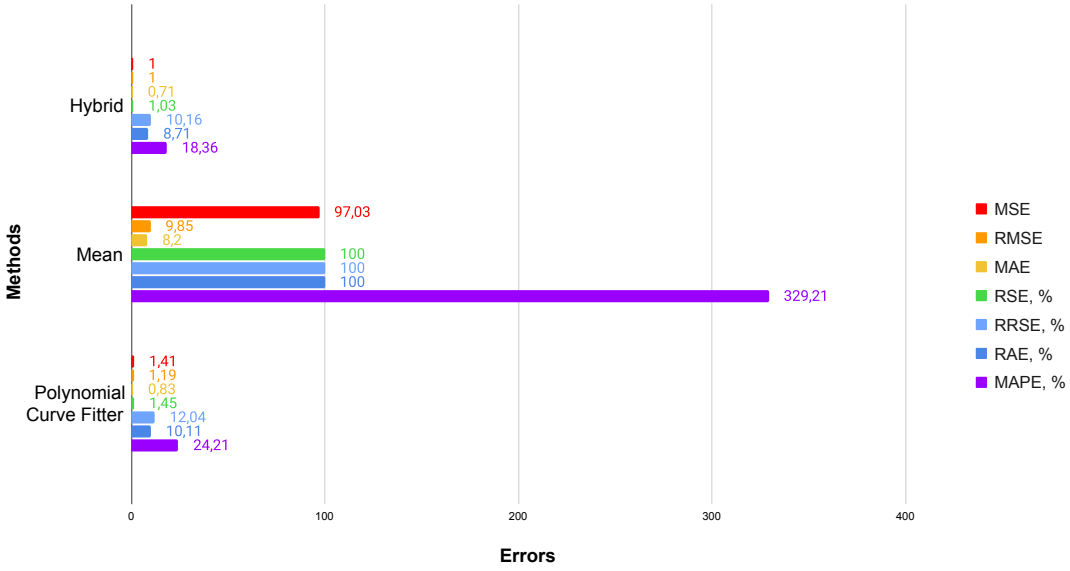


Figure 13: Performance measures of Atmosphere Temperature column

The other column that showed the same trend, namely, a great gap between the worst and hybrid and a small gap between the best, is relative humidity. Its valuation is in Figure 14. In this case, the best and worst methods were polynomial curve fitter (also using time as an independent value, but in the third degree) and median respectively.

All other columns showed the same trend, except for the wind direction column. Here, the results of our method were among the middle ones, but that is caused by the fact that it is a vector, which changes in an interval of 0 to 360, and even other methods failed to predict it (the smallest MAPE, which is the last error at the diagrams, was more than 200%).

Test results of CCPP Dataset

The second dataset is Combined Cycle Power Plant. Here, in all cases, the hybrid method showed middle results. This might be caused by the data, since among "winners" were such methods as mean and median imputation. The results of

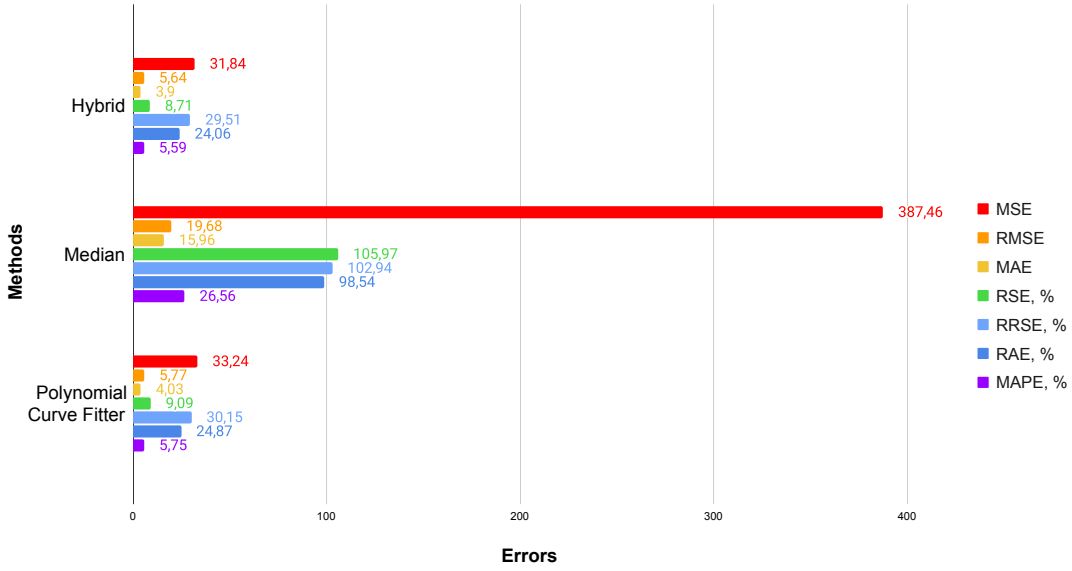


Figure 14: Performance measures of Relative Humidity column

linear regression and polynomial curve fitter (of both second and third degrees) had even worse performance measures than the hybrid one. For instance, in the Figure 15, you can see the evaluation of methods for the column which contained values of exhaust vacuum.

Test results of Yellowstone Park Dataset

The last dataset is the Yellowstone Park dataset. Here, we have only one predicted value, which is altitude, and the independent variables are GPS coordinates. The testing results are displayed in Figure 16. In contrast to previous diagrams, we have removed MSE, since its value for the mean imputation is far too big and as a result, other errors would be overshadowed. Namely, for hybrid method MSE is 200.08, for mean - 24998,53 and for polynomial curve fitter - 201,73. As you can see, the polynomial curve fitter (using latitude in the third degree) and our hybrid method are pretty much the same, but looking at the values of errors, it is noticeable that there was some small improvement in predictions.

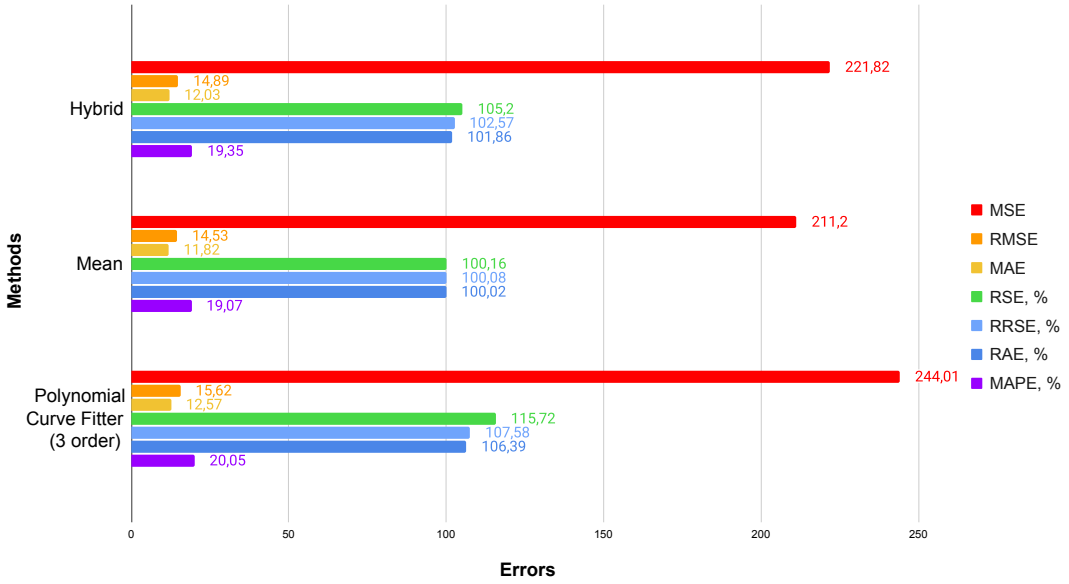


Figure 15: Performance measures of Exhaust Vacuum column

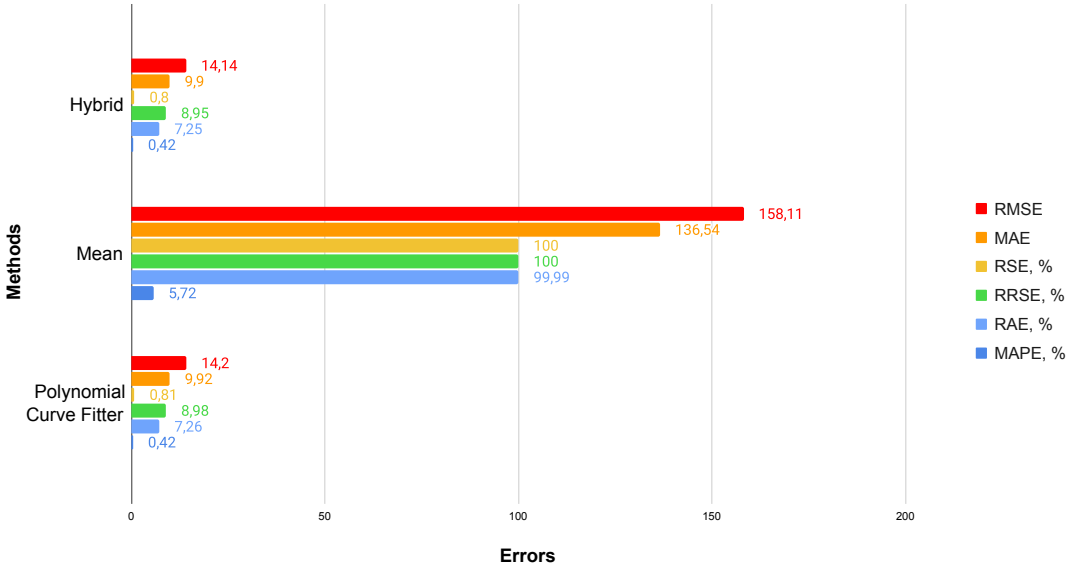


Figure 16: Performance measures of Altitude column

4 Conclusion

During this work, we made an effort to design a hybrid method for the replacement of missing values. We had a deep look into the world of machine learning, especially the data preprocessing part.

In the first chapter, we went through different kinds of missing values and the reason for their existence. We also had a look into the existing methods of handling missing data, such as different types of deletion (listwise, pairwise, *etc.*) and imputation methods (number of regressions, mead, median imputation, *etc*). In that part, along with the use cases of each of them, we also outlined their benefits and drawbacks, which lead us during the implementation of the hybrid method later. We also examined statistical measures which lately assist us in creating a sensible decision-making mechanism. Additionally, we went through metrics that helped us with evaluating the performance of the designed hybrid method. Moreover, in that chapter, we also described tools and frameworks that are used for the same purpose as the implemented method has and also described some examples of relevant works.

In the second chapter, we described our solution, starting from the rough overview and then slowly diving deeper into the implementation details. There, the structure of the input and output are described as well as the preconditions and requirements to be fulfilled so the program works correctly. Then, we described the logic of the main algorithm that makes the decision as well as listed the imputation methods that were used as a part of it. We also mentioned a few technical details in that part. As a part of the implementation, we also had to tune the parameters (mentioned as thresholds). Finally, we tested our design on three different datasets and showed achieved results by comparing performance measures, such as MAPE, RMSE, *etc.*, of our method with each method that was used as its part.

The goal, and consequently the outcome, of the work is the implemented method that showed quite promising results in case of multiple imputation (when there is more than one independent variable). In such case, the designed hybrid method showed better results than other existing techniques it was compared with. This method also has its limitations. Namely, there are special requirements for the input of the program, such as only numeric data and only .csv files are accepted. The other thing is the time complexity of the method, it is bigger than using one of the existing simple methods, but even though the algorithm has all those if-statements and different checks, it is faster than multiple imputation methods, since it saves time when using a simple method instead of training a model. Talking in numbers, it is a matter of less than 0.1-0.5 seconds for a 30000-records dataset with 7000 missing values. The other weakness, based on the performance measures, is simple imputation (when there is one independent variable). In that case, the method didn't show better results, but, as we outlined in the section 3.4, it can be caused by the data itself.

The implemented designed method was created with a view of its real future usage. Therefore, we hope that in case of multiple imputation, it can really be useful for data analysts. At the same time, the part that is responsible for simple imputation cases can be a potential future improvement of the designed method.

Resumé

1 Úvod

V súčasnosti majú dáta nevyčíslitelnú hodnotu. Informácie, spolu so schopnosťou ich analyzovať a získavať z nich určité výsledky sú silným nástrojom z rôznych dôvodov. Príkladom je poskytovanie trhu žiadaný produkt alebo službu, či vytváranie dopytu pomocou reklám pre cieľových spotrebiteľov. Existujú ale aj iné oblasti, kde analýza údajov zohráva významnú rolu, ako napríklad veda a medicína.

Z vyššie uvedených dôvodou existujúci dopyt na údaje nie je žiadnym prekvapením. Tieto záznamy by ale mali byť vyhovujúce na vytvorenie spoľahlivého predikčného modelu, konkrétne by mali byť presné a úplné. V opačnom prípade to môže mať škodlivý vplyv na výsledky modelu. Žiaľ, údaje bývajú dokonalé veľmi zriedka, a preto každí dátoví analytik čelí výzve nájsť spôsob, ako zmierniť vplyv takýchto dát na budúci odhad.

Z tohto dôvodu existujú rôzne techniky, ako sa vysporiadať s týmito neúplnými dátami. Príkladom môže byť odstránenie určitej časti dát, ktorá nie je správna, a na trénovanie sa použije iba časť, ktorá je vhodná. Toto môže byť často nevyhovujúci prístup, nakoľko všetky údaje sú dôležité. Bežným spôsobom preto býva dodefinovanie chýbajúcich hodnôt, napríklad nahradením priemernou alebo najbežnejšou hodnotou, no častejšie je potrebné vytvoriť ďalší model na ich predikciu. Existuje mnoho takýchto techník, no všetky majú svoje výhody a nevýhody a je vhodné ich používať iba v špecifických prípadoch.

Cieľom tejto práce je teda nájsť kombináciu týchto metód - takzvanú hybridnú metódu imputácie - s cieľom znížiť vplyv nevýhod a naplno využiť výhody každej z nich.

2 Analýza problému

Žiaľ, nie všetky údaje bývajú úplné, čoho príčinou môže byť softvérová, hardvérová alebo ľudská chyba. Zvyčajne sú tieto hodnoty reprezentované pomocou ?, NaN, Na, nulou alebo prázdnou bunkou, čo záleží najmä od programovacieho jazyka [1]. Keďže chýbajúce údaje môžu mať rôzny vplyv na výsledky výskumu, uvádzame tri rôzne typy chýbajúcich hodnôt (viď obrázok 1):

- Missing at Random (MCAR) – popisuje prípad, kedy hodnota chýba nezávisle na hodnotách, ktoré sa mali zbierať, ako aj na sérii iných pozorovaní [2].
- Missing at Random (MAR) – používa sa na definovanie prípadu, keď ostatné premenné súvisia s chýbajúcou hodnotou, ale medzitým nezávisia od jej hypotetickej hodnoty [4].
- Missing Not at Random (MNAR) – všetky údaje, ktoré sa nepovažujú za MCAR ani MAR, patria do tohto typu [2].

Výber metódy na spracovanie chýbajúcich hodnôt však zvyčajne závisí nielen od typu chýbajúcich údajov (MCAR, MNAR alebo MAR), ale aj od iných charakteristík pozorovaní. Niektoré z techník sa zameriavajú napríklad na numerické údaje (ktoré môžu byť spojité alebo diskrétne), iné sú vhodné len pre kategorické typy. Tieto metódy sú zároveň rozdelené do samostatných skupín – tie, ktoré nahrádzajú chýbajúcu hodnotu, nazývané metódy imputácie hodnoty, a tie, ktoré vymazávajú celú vzorku v prípade absencie jednej alebo viacerých hodnôt.
Metóda odstránenia údajov je najjednoduchší spôsob, ako riešiť chýbajúce hodnoty. Princípom je vymazanie celého prediktora a/alebo celej vzorky, ktorá obsahuje aspoň jednu chýbajúcu hodnotu. Existujú rôzne prístupy, ako je *listwise* a *pairwise deletion*, *dropping varibles* (viď sekciu 2.2.1).

Ďalším spôsobom spracovania údajov je použitie metód pre imputácie hodnoty. Tieto metódy analyzujú štatistiku a/alebo vzťahy medzi prediktormi a pozorovaniami, aby bolo možné predpovedať chýbajúcu hodnotu.
Základné imputačné metódy sú vyplnenie medzier priemerom, modusom alebo mediánom. Ďalšou metódou imputácie hodnoty je použitie určitého typu regresie. Najbežnejšie sú lineárne a polynomické regresie, ktoré je možné upraviť v prípade, keď existuje viac ako jedna nezávislá premenná. Existuje aj logistická regresia, ktorá sa často používa na predpovedanie diskrétnych hodnôt. Pozdĺžne údaje vyžadujú osobitné zaobchádzanie, pretože môžu mať v časovom priebehu osobitnú tendenciu. V takom prípade sa používa lineárna interpolácia, Last Observation Carried Forward a Next Observation Carried Backward.
Rovnako ako pri mnohých problémoch v oblasti Strojového Učenia, použitie metód imputácie hodnoty vyvoláva otázku, či sa ich oplatí použiť a či fungujú lepšie ako jednoduché vymazanie zo zoznamu. Na tento účel sa používajú rôzne metriky. Pre modely s číselnými údajmi sú najčastejšie MSE, RSE, resp. RMSE, RRSE, atď. (viac v sekcie 2.4).

3 Opis riešenia

Dáta

Pri implementácii sme používali rôzne datasety, no najviac sme využívali dataset od Slovenského hydrometeorologického ústavu, keďže bol použiteľný na účely viacnásobného imputovania (na základe viac ako jedného prediktora), konkrétne v tomto prípade boli nezávislé premenné čas, zemepisná dĺžka a šírka.
Aby sme mohli pracovať s datasetom počasia, vytvorili sme niekoľko skriptov v programovacom jazyku Python na jeho predbežné spracovanie (všetky skripty sú v `src/com/company/data/`). Konkrétne `combine.py`. a `combine_multiple.py` pre jednoduché, respektíve viacnásobné dodefinovania hodnôt. Taktiež sme použili jeden skript na simuláciu chýbajúcich hodnôt - `add_missing_values.py`.

Pre navrhovanú hybridnú metódu existuje niekoľko hlavných predpokladov: údaje by mali byť iba číselné, inak bude `NumberFormatException`, a datasety pochádzajú z `.csv` súbory.

Vstup

Okrem datasetov má vstup programu dve ďalšie formy - konzolový vstup alebo `config.properties` súbor. So vstupom z konzoly by mal používateľ napísať odpovede na zobrazené otázky. Ale keďže takýto vstup nie je vždy použiteľný (v prípade mnohonásobného spustenia programu), existuje súbor, do ktorého je možné parametre uložiť (viac o `config.properties` v Prílohe C).

Hybridná metóda

Hlavný priebeh údajov je znázornený pomocou vývojového diagramu na obrázku 9. Po prvotnom naskenovaní vstupu a vytvorení objektu datasetu program vypočíta štatistické miery pre každý stĺpec, ktorý sa má predpovedať. Následne sa spustí hlavný cyklus, konkrétne prechádzanie datasetom, záznam po zázname. Pre každú chýbajúcu hodnotu vytvoríme tréningový dataset a na základe štatistík vyberieme vhodnú metódu imputácie. Po pripočítaní všetkých možných hodnôt v datasete program pokračuje k časti ukladania údajov do súboru a v prípade poskytnutia kompletného datasetu vyhodnotí výsledky.
Pre jednoduché a viacnásobné prípady imputácie je príprava tréningového súboru údajov odlišná. V prípade jednoduchého sa vyberú určité záznamy (ktoré nemajú v stĺpcoch prediktor a chýbajúce hodnoty) okolo aktuálneho. Pri viacnásobných prípadoch imputácie vytvárame dataset z najbližších záznamov, ktorým odporuje euklidovská vzdialenosť. Nakoniec na základe rôznych charakteristík, ktoré sú znázornené na obrázku 11 a obrázku 12, vyberieme vhodnú metódu pre jednoduché a viacnásobné prípady imputácie. Ak zvolená metóda nedokáže predpovedať platnú hodnotu,použijeme jednu z ďalších metód dodefinovania hodnôt,

ako je na obrázku 10b.

Výstup

Výstup, okrem datasetu s pripočítanými hodnotami sú aj miery výkonu hybridnej metódy, ktoré program vytlačí a uloží do `result.txt` súboru, ak bol poskytnutý úplný dataset na porovnanie. V konzole sa taktiež zobrazí štatistika predpokladaného stĺpca/-ov.

Výsledky

Táto časť opisuje výsledky dosiahnuté počas tejto práce. Jedna z vecí, ktorú je nutné spomenúť je, že navrhnutá hybridná metóda je schopná predpovedať chýbajúce hodnoty, aj keď neexistujú žiadne prediktory (alebo ak chýbajú).Pri testovaní sme museli túto funkciu deaktivovať, pretože nie všetky metódy, s ktorými sme našu metódu porovnávali, to dokážu. Na testovanie sme pripravili tri datasety, ktorých testovacie údaje sú v Prílohe C. Porovnávali sme výsledky našej hybridnej metódy s výsledkami každej metódy, ktorá bola použitá ako jej súčasť. Príprava trénovacieho datasetu pre každú chýbajúcu hodnotu bola v oboch prípadoch rovnaká (okrem prípadov nahradenia chýbajúcich hodnôt priemerom alebo mediánom). V tejto časti ukážeme len niektoré výsledky, všetky sú v Prílohe B.
Prvý dataset, ktorý sme porovnávali, je zo Slovenského hydrometeorologického ústavu. Hlavným trendom v takmer všetkých predpovedaných stĺpcoch bolo, že ukazovatele výkonnosti hybridnej metódy boli o niečo lepšie ako výsledky iných metód. Napríklad pre teplotu atmosféry je možné vidieť vyhodnotenie na obrázku 13. Všetky ostatné stĺpce vykazovali rovnaký trend, okrem stĺpca ktorý obsahoval smer vetra. Tu patrili výsledky našej metódy k tým stredným. To je spôsobené tým, že ide o vektor, ktorý sa mení v intervale 0 až 360, a ani iné metódy ho taktiež nedokázali predpovedať (najmenšia MAPE, čo je posledná chyba v diagramoch, bola viac ako 200%).

Druhým datasetom bol dataset Combined Cycle Power Plant. Vo všetkých prípadoch hybridná metóda vykazovala stredné výsledky, ako je možné vidieť na obrázku 15. Mohlo to byť spôsobené údajmi, keďže medzi „víťazmi“ boli také metódy ako nahradenie priemerom alebo mediánom. Výsledky lineárnej regresie a polynomiálnej krivky (druhého aj tretieho stupňa) mali ešte horšie ukazovatele výkonnosti ako hybridná metóda.
Posledným datasetom bol dataset Yellowstone Parku. Výsledky testovania sú zobrazené na obrázku 16. Na rozdiel od predchádzajúcich diagramov sme odstránili MSE, pretože jej hodnota pre imputáciu priemeru je príliš veľká a v dôsledku toho by boli ostatné chyby neviditeľné, konkrétne pre hybridnú metódu MSE je 200,08, pre priemer - 24998,53 a pre zostavovanie polynomickej krivky - 201,73. Ako je vidieť, polynomiálna krivka (pomocou zemepisnej šírky v treťom stupni) a naša hybridná metóda sú takmer rovnaké, ale pri bližšom pohľade na hodnoty chýb je zrejmé, že došlo k malému zlepšeniu predpovedí.

4 Záver

Počas tejto práce sme navrhli hybridnú metódu, ktorej cieľom bolo nahrádzanie chýbajúcich hodnôt v datasetoch. Zamerali sme sa na svet strojového učenia, najmä na časť predspracovania údajov.
V prvej kapitole sme opísali rôzne druhy chýbajúcich hodnôt a dôvod ich existencie. Pozreli sme sa aj na existujúce metódy spracovania chýbajúcich údajov, akými sú rôzne typy vymazávania (zoznamové, párové, atď.) a metódy imputácie (niekoľko regresií, nahradenie mediánom alebo priemerom, atď.). V tejto časti sme spolu s prípadmi použitia každej z nich načrtli aj ich výhody a nevýhody, ktoré nás neskôr smerovali pri implementácii hybridnej metódy. Preskúmali sme aj štatisticky opis, ktorý nám pomohol vytvoriť spoľahlivý rozhodovací mechanizmus. Okrem toho sme prešli metrikami, ktoré nám pomohli s hodnotením výkonu

navrhnutej hybridnej metódy. Takisto sme v tejto kapitole popísali aj nástroje a rámce, ako aj niektoré príklady iných relevantných prác.

V druhej kapitole sme popísali naše riešenie, začínajúc hrubým prehľadom a potom sa pomaly ponorili hlbšie do detailov implementácie. Spomenuli sme aj niekoľko technických detailov. V rámci implementácie sme museli vyladiť parametre (nazývané thresholds). Nakoniec sme otestovali náš návrh na troch rôznych datasetoch a ukázali sme dosiahnuté výsledky porovnaním ukazovateľov výkonnosti (MAPE, RMSE atď.) našej metódy s každou metódou, ktorá bola použitá ako jej časť.

Cieľom a následne aj výsledkom práce je teda implementovaná metóda, ktorá ukázala sľubné výsledky pri viacnásobnej imputácii (pri viac ako jednej nezávislej premennej). V tomto prípade navrhovaná hybridná metóda vykazovala lepšie výsledky ako iné existujúce techniky, s ktorými bola porovnávaná. Táto metóda ale má svoje obmedzenia. Konkrétne časovú náročnosť metódy, ktorá je väčšia ako pri použití jednej z existujúcich jednoduchých metód. Ale aj keď má algoritmus všetky if-statements a rôzne overenia a počítania, je rýchlejší ako viacnásobné imputačné metódy, nakoľko šetrí čas pri použití jednoduchej metódy namiesto trénovania modelu. Ďalšou slabinou, podľa ukazovateľov výkonnosti, je jednoduchá imputácia (keď existuje jedna nezávislá premenná). V takom prípade metóda neukázala lepšie výsledky, ale ako sme uviedli v sekcii 3.4, toto môže byť spôsobené samotnými údajmi.

Implementovaná navrhnutá metóda bola vytvorená s ohľadom na jej reálne budúce využitie. Preto dúfame, že môže byť užitočná pre dáta analytikov v prípade, že chýbajúce hodnoty sú v datasete s viac ako jednou nezávislou premennou. Zároveň časť, ktorá je zodpovedná za jednoduché prípady imputácie, môže byť potenciálnym budúcim vylepšením navrhnutej metódy.

References

1. ALAM, Mahbubul. Dealing with missing data in data science projects. *Towards Data Science*. 2020. Available also from: https://towardsdatascience.com/dealing-with-missing-data-in-data-science-projects-e8ac7a4efdff.
2. KANG, Hyun. The prevention and handling of the missing data. *Korean journal of anesthesiology*. 2013, vol. 64, no. 5, pp. 402.
3. KUHN, Max; JOHNSON, Kjell. *Feature engineering and selection: A practical approach for predictive models*. CRC Press, 2019. Available also from: https://bookdown.org/max/FES/imputation-methods.html.
4. SWALIN, Alvira. How to handle missing data. *Towards Data Science*. 2018, vol. 18, pp. 01–19. Available also from: https://towardsdatascience.com/how-to-handle-missing-data-8646b18db0d4.
5. SINGHAL, Shashank. Defining, Analysing, and Implementing Imputation Techniques. *Analytics Vidhya*. 2021. Available also from: https://www.analyticsvidhya.com/blog/2021/06/defining-analysing-and-implementing-imputation-techniques/.
6. ZHANG, Zhongheng. Missing data imputation: focusing on single imputation. *Annals of translational medicine*. 2016, vol. 4, no. 1.
7. AGRAWAL, Raghav. All you need to know about Polynomial Regression. *Analytics Vidhya*. 2021. Available also from: https://www.analyticsvidhya.com/blog/2021/07/all-you-need-to-know-about-polynomial-regression/.
8. SINHA, Priyanka. Multivariate polynomial regression in data mining: methodology, problems and solutions. *International Journal of Scientific and Engineering Research*. 2013, vol. 4, no. 12, pp. 962–965.

9. EDGAR, Thomas W.; MANZ, David O. Chapter 4 - Exploratory Study. In: *Research Methods for Cyber Security*. Syngress, 2017, pp. 95–130. ISBN 978-0-12-805349-2. Available from DOI: https://doi.org/10.1016/B978-0-12-805349-2.00004-2.
10. NICK, Todd G. Descriptive statistics. *Topics in biostatistics*. 2007, pp. 33–52.
11. WITTEN, Ian H; FRANK, Eibe; HALL, Mark A; PAL, CJ; DATA, MINING. Practical machine learning tools and techniques. In: *DATA MINING*. 2005, vol. 2, p. 4.
12. DEVAL, Swati. *Mean Squared Error – Explained | What is Mean Square Error?* 2020. Available also from: https://www.mygreatlearning.com/blog/mean-square-error-explained/. Great Learning.
13. HIREGOUDAR, Shravankumar. *Ways to Evaluate Regression Models*. 2020. Available also from: https://towardsdatascience.com/ways-to-evaluate-regression-models-77a3ff45ba70.
14. DE MYTTENAERE, Arnaud; GOLDEN, Boris; LE GRAND, Bénédicte; ROSSI, Fabrice. Mean absolute percentage error for regression models. *Neurocomputing*. 2016, vol. 192, pp. 38–48.
15. PROTASIEWICZ, Jakub. Python AI: Why Is Python So Good for Machine Learning? *Netguru*. 2018. Available also from: https://www.netguru.com/blog/python-machine-learning.
16. RIDGERS, Malcom. Can Java Be Used for Machine Learning and Data Science? 2020. Available also from: https://www.kdnuggets.com/2020/04/java-used-machine-learning-data-science.html.
17. CAMPIT, Scott. 3 Reasons Why You Should Choose MATLAB As The Programming Language To Learn Data Science. *Towards Data Science*. 2020. Available also from: https://towardsdatascience.com/why-i-chose-matlab-for-learning-data-science-4f5e4650dce9.
18. ELIZABETH, Jane. Top 5 machine learning libraries for Java. 2017. Available also from: https://jaxenter.com/top-5-machine-learning-libraries-java-132091.html.

19. DUCATELLE, Frederick. SOFTWARE FOR THE DATA MINING COURSE. Available also from: http://www.inf.ed.ac.uk/teaching/courses/dme/html/software2.html.
20. VADAPALLI, Pavan. Scikit-learn in Python: Features, Prerequisites, Pros & Cons. 2020. Available also from: https://www.upgrad.com/blog/scikit-learn-in-python/.
21. *Apache Commons Math*. 2021. Available also from: https://commons.apache.org/proper/commons-math/.
22. RAFF, Edward. JSAT: Java Statistical Analysis Tool, a Library for Machine Learning. *Journal of Machine Learning Research*. 2017, vol. 18, no. 23, pp. 1–5. Available also from: http://jmlr.org/papers/v18/16-131.html.
23. PATRAWALA, Fatema. 6 most commonly used Java Machine learning libraries. *Packt*. 2018. Available also from: https://hub.packtpub.com/most-commonly-used-java-machine-learning-libraries/.
24. ABEEL, Thomas; VAN DE PEER, Yves; SAEYS, Yvan. Java-ml: A machine learning library. *Journal of Machine Learning Research*. 2009, vol. 10, pp. 931–934.
25. *JAMA : A Java Matrix Package*. 2012. Available also from: https://math.nist.gov/javanumerics/jama/.
26. NADIMI-SHAHRAKI, Mohammad H; MOHAMMADI, Saeed; ZAMANI, Hoda; GANDOMI, Mostafa; GANDOMI, Amir H. A Hybrid Imputation Method for Multi-Pattern Missing Data: A Case Study on Type II Diabetes Diagnosis. *Electronics*. 2021, vol. 10, no. 24, pp. 3167.
27. PEKER, Nuran; KUBAT, Cemalettin. A Hybrid Modified Deep Learning Data Imputation Method for Numeric Datasets. *International Journal of Intelligent Systems and Applications in Engineering*. 2021, vol. 9, no. 1, pp. 6–11.
28. PURWAR, Archana; SINGH, Sandeep Kumar. Hybrid prediction model with missing value imputation for medical data. *Expert Systems with Applications*. 2015, vol. 42, no. 13, pp. 5621–5631.

29. COHEN, Patricia; WEST, Stephen G; AIKEN, Leona S. *Applied multiple regression/correlation analysis for the behavioral sciences*. Psychology press, 2014.

Appendix A: Technical Documentation

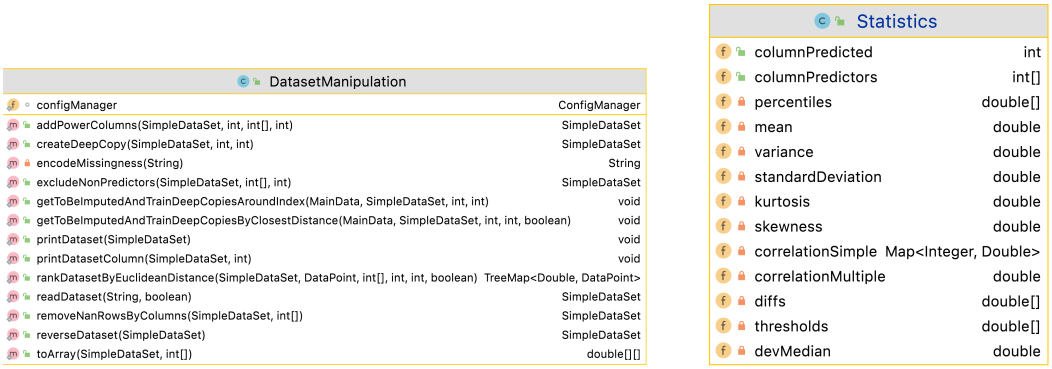
The logic and flow of the designed hybrid method were explained in the Solution Description chapter, therefore, in this part, we are going to describe the implementation from a more technical viewpoint along with the code listings.

Data

The main data that the program works with is the dataset itself. This one is stored in the object of `SimpleDataSet.class`, which relatively simplifies the manipulation of the dataset and its transformation. The other reason for that is that the functionality we use for reading the dataset from a file is coming from the same library as the mentioned class, namely JSAT and is a return type of the read method. As we use methods from different libraries, sometimes we need to transform the dataset from the mentioned type to some other. The implementation for this can be found either in `DatasetManipulation` or, in case the transformation is very method-specific, it is done in place, namely in `processData()` of the corresponding method. Talking about `DatasetManipulation`, it is a class with methods which work with the dataset (see Figure A1a). This class additionally to the reading and writing dataset also encodes missingness (replaces all representations of missing values to one form understandable by the program), removes columns and missing values, *etc.*

Additionally to the dataset, we also store other data along with it. Namely, each dependent value that is being imputed has its description statistic. At the Figure A1b you can see the class that represents the statistic description of a dependent variable. In case, we have both complete and incomplete datasets during the program, we store each imputed and original value along with the index of the record where the missing value occurred in the object of `ImputedValue.class`

which is then used for the evaluation at the end of the program (also done separately for each program).



(a) (b)

Figure A1: Dataset Manipulation (a) and Statistics (b) classes

Main aspects

Input and output were already explained in the main part of the document. A more detailed overview of the `config.properties` file, which was also mentioned there, is in Appendix C: User Input and Output.

The main flow of the algorithm was already explained, here we would like to show the corresponding code (see Listing A1). So, as outlined, we are traversing the dataset. Each record we check for the missing values. Then we additionally check if any of the missing values was not meant to be a predictor. This flag will be passed to the `impute()` method and will result in a decision between mean and median imputation methods. In case we specify a particular column to be imputed, we run imputation only if a value in that column in the current record is missing. When no column is specified, we check all columns and run imputation only for those which were not meant to be predictors. We would like to outline at the beginning that at some places we use *thresholds* which are defined by us based on experiments. These thresholds are stored for each column separately in object

```
//traverse dataset one by one
for (DataPoint dp : datasetMissing.getDataPoints()) {
    // indexes of missing values
    int[] indexes = getIndexesOfNull(dp);
    // if any of the missing values is specified as a predictor
    boolean missingPredictor = getIntersection(indexes,
    ↪  columnPredictors).length > 0;
    // indexes of missing values which are not meant to be predictors
    int[] missingNonPredictors = getDifference(indexes,
    ↪  columnPredictors);

    if (columnPredicted != -1 &&
    ↪  IntStream.of(missingNonPredictors).anyMatch(x -> x ==
    ↪  columnPredicted)) {
        // if column to be imputed is specified and is among missing
        ↪  values
        impute(dp, columnPredicted, missingPredictor);
    } else {
        // if all columns with missing values should be imputed, we
        ↪  impute only non-predictors
        for (int idx : missingNonPredictors) {
            impute(dp, idx, missingPredictor);
        }
    }
}
```

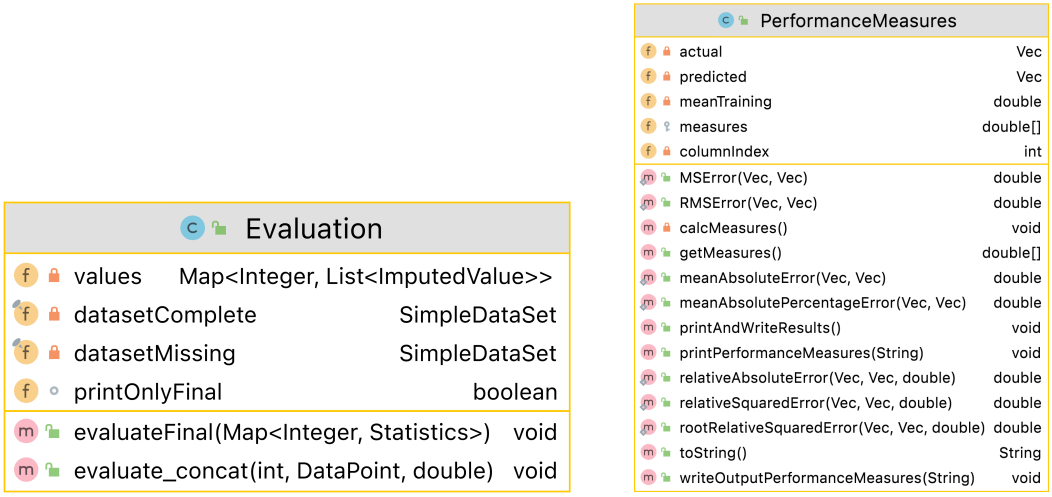
Listing A1: Traversing Dataset

of `Statistics.class` (see Figure A1b) or used in place.

At the end, the evaluation is performed by `Evaluation.class` which gathers the data for final assessment throughout the program. The `values` field of the class contains performance measures calculated by methods in `PerformanceMeasures.class`. Both these classes are shown at Figure A2.

It is also noteworthy that each imputation methods implement a common interface called `ImputationMethod` as it is shown at Fig. A3. There you can see that each method has its implementation the following methods:

- `preprocessData()` - preprocesses data which is understandable for the



(a) (b)

Figure A2: Evaluation (a) and Performance Measures (b) classes

model. Sometimes some conversion or cleaning is needed.

- `fit()` - trains a model/prepares a predictor.
- `predict()` - predicts value and returns it
- `print()` - prints in standard output the name of the method with its parameters, such as coefficients in the Linear Regression equation.

Figure A3: Imputation Method Interface

Conditions for Simple Imputation

In this section, we would like to explain in more detail the conditions which control which simple imputation method is going to be chosen. The part of the program which makes a decision is shown in Listing A2.

```
// select appropriate imputation method
if (getMeanDevPercent(data) <= stat.getThresholds()[0]) {
    return new MeanImputationMethod(data);
} else if (getMedianDevPercent(data) <= stat.getThresholds()[1]) {
    return new MedianImputationMethod(data);
} else if (data.getColumnPredictors().length == 1) {
    int columnPredictor = data.getColumnPredictors()[0];
    if (isStrictlyIncreasing(data, data.getColumnPredicted()) &&
    ↪  isStrictlyIncreasing(data, columnPredictor)) {
        return new LinearInterpolatorApacheMethod(data, true);
    } else if (isStrictlyDecreasing(data, data.getColumnPredicted()) &&
    ↪  isStrictlyDecreasing(data, columnPredictor)) {
        return new LinearInterpolatorApacheMethod(data, false);
    } else if (getLinearCorr(data) > stat.getThresholds()[2]) {
        return new LinearRegressionJSATMethod(data);
    } else {
        int order = getPolynomialOrderSimple(data,
        ↪  stat.getThresholds()[3]);
        if (order != -1) {
            return new PolynomialCurveFitterApacheMethod(data, order);
        }
    }
    if (isWithinInterval(getMeanDevPercent(data), new double[]{0.32,
    ↪  0.35}) && isWithinInterval(getLinearCorr(data), new
    ↪  double[]{0.09, 0.16})){
        return new UseClosest(data);
    }
}
return new MeanImputationMethod(data);
```

Listing A2: Simple Imputation Conditions

So, the first step is to define whether simple mean and median imputation methods would be appropriate. The reason why these are the first check is that they are

the least computationally and time expensive. Also, they are the only choice if there are no predictors. More detailed into the conditions `getMeanDevPercent()` and `getMedianDevPercent()`: this methods use current train dataset and also the statistics of the predicted column. In both cases, we count the deviation from the mean, respectively median, by the formula inspired by the standard deviation (Formula 9) with the median used in place of the mean in the second case. Then we calculate the proportion between the deviation and the mean/median and compare the results with predefined thresholds, which we mentioned at the beginning of the section.

```
static public double getMeanDevPercent (MainData data) {
    SimpleDataSet dataSet = data.getTrain();
    Vec columnPredicted =
    ↪  dataSet.getDataMatrix().getColumn(data.getColumnPredicted());
    double std = columnPredicted.standardDeviation();
    double mean = abs(columnPredicted.mean())

    return std / mean;
}

static public double getMedianDevPercent (MainData data) {
    Vec column =
    ↪  data.getTrain().getDataMatrix().getColumn(data.getColumnPredicted());
    double median = column.median();
    double dev = getDevMedian(median, column);

    return abs(dev / median);
}
```

Listing A3: Simple Imputation Conditions: Close to Mean and Median

Then follows a check for whether is a predictor (and there should only be one since there are only simple imputation methods in this part). If no, we proceed with the default, namely the mean imputation method. The same will happen if neither of the following methods will be chosen.

The next two checks are for Linear Interpolation. For this method, the requirement is that both predictor and predicted values are strictly increasing. We do

the check both ways - strictly increasing and decreasing. If the second condition will be true, we will reverse the dataset and use the original Linear Interpolation method.

If neither of the previous condition will be valid, we check whether there is any linear relationship between independent and dependent variables. The `getLinearCorr()` method is common for both simple and multiple imputation cases, but inside we count the correlation coefficient differently (see Listing A4): when there is more than one predictor we calculate multiple correlation coefficient following Formula 25., otherwise we calculate Pearson Correlation Coefficient that is represented by Formula 18. As well as in case of mean and median imputation methods, we compare the result with a predefined threshold.

```
static public double getLinearCorr (MainData data) {
    SimpleDataSet dataSet = data.getTrain();
    int[] columnPredictors = data.getColumnPredictors();
    int columnPredicted = data.getColumnPredicted();

    double corr;
    if (columnPredictors.length > 1) {
        //calculate multiple correlation coefficient
        corr = getCorrMultiple(dataSet, columnPredicted,
        ↪  columnPredictors);
    } else {
        //calculate pearson correlation coefficient with one predictor
        corr = correlation(dataSet.getNumericColumn(columnPredicted),
        ↪  dataSet.getNumericColumn(columnPredictors[0]), true);
    }
    return abs(corr);
}
```

Listing A4: Simple Imputation Conditions: Linear Relationship

Then follows a condition for polynomial regression. In this case, not only do we try to check whether there is a polynomial relationship between the variables, but also try to define the most appropriate degree. To define that we calculate the linear relationship between the predicted variable and the predictors with a

degree from 1 to 4. After that, we use the biggest (we are using absolute values) correlation coefficient and compare it with the threshold in the same way as above. The last condition is for using the closest value for imputation. Here we use some allowed intervals for the values returned from `getMeanDevPercent()` and `getLinearCorr()` methods.

Conditions for Multiple Imputation

In this section, we would like to explain in more detail the conditions which control which multiple imputation method is going to be chosen. The part of the program which makes a decision is shown in Listing A5.
Here, we first check for the number of predictors left after exclusion. If there is less than one we check if multiple polynomial regression would be appropriate based on kurtosis and multiple correlation coefficient with all predictors in the second degree (since it is the maximum degree we use for multiple imputation methods). If that check does not pass, we decide on using the value from the closest data record (based on euclidean distance). This check uses other statistical descriptions of the predicted column. Then, if the last is also not appropriate, the default is polynomial curve fitter which uses the predictor left after exclusion. If there is still more than one predictor, we first decide on the linear relationship. The principle is the same as with simple imputation methods (see Listing A4), but in this case, there is more than one predicted and, therefore, we calculate multiple correlation coefficient.
If the check for linear fails, we test if it is polynomial. Here, we also calculate for multiple correlation coefficient and compare it with a threshold.
If it is not even a polynomial regression, we use one of our default methods. The `MultiDimensionalMultipleImputation` is used only if the configuration in `config.properties` allows to use it, otherwise the default is `MeanImputationMethod`.

```
// After composing train dataset some predictor columns contain same
↪  values, such columns' indexes are excluded from predictors.
if (data.getColumnPredictors().length == 1) {
    double polyRel = getPolyCorr(data);

    if (useMultiplePolynomialForSinglePredictor(stat, polyRel)) {
        return new MultiplePolynomialRegressionJSATMethod(data, 2);
    }
    if (useClosest(stat, polyRel)) {
        return new ClosestImputation(data);
    }
    return new PolynomialCurveFitterApacheMethod(data,
    ↪  getPolynomialOrderSimple(data, 0.0));
}

if (getLinearCorr(data) > stat.getThresholds()[2]) {
    // linear regression is a special way of polynomial regression with
    ↪  1st degree
    // (it is default therefore we do not pass it)
    return new MultiplePolynomialRegressionJSATMethod(data);
} else if (getPolyCorr(data) > stat.getThresholds()[3]) {
    return new MultiplePolynomialRegressionJSATMethod(data, 2);
} else if (useMultiDimensionalAsDefault) {
    return new MultiDimensionalMultipleImputation(data, datasetMissing,
    ↪  stat);
}
return new MeanImputationMethod(data);
```

Listing A5: Multiple Imputation Conditions

Imputation Methods Implementation

Some of the imputation methods used by the design hybrid one were implemented during the work on this project, the others are already existing in libraries and were reused. In this section, we would like to describe those implemented by us as well as some interesting re-usage of the existing ones.

We would like to start with the easier example, namely Multiple Polynomial Regression. We haven't found the implementation of this method in the libraries and

```
SimpleDataSet trainingCopy2 =
↪  DatasetManipulation.excludeNonPredictors(trainingCopy,
↪  columnPredictors, columnPredicted);
SimpleDataSet toBePredicted2 =
↪  DatasetManipulation.excludeNonPredictors(toBePredicted,
↪  columnPredictors, columnPredicted);
int[] predictors = Arrays.copyOf(columnPredictors,
↪  columnPredictors.length);
if (degree > 1) {
    for (int i = 0; i < predictors.length; i++) {
        predictors[i] = i + 1;
    }
    predictors = Arrays.copyOf(predictors, predictors.length * degree);
    for (int i = predictors.length / degree; i < predictors.length; i++)
    ↪  {
        predictors[i] = i + 1;
    }
    trainingCopy2 = DatasetManipulation.addPowerColumns(trainingCopy2,
    ↪  degree, predictors, 0);
    toBePredicted2 = DatasetManipulation.addPowerColumns(toBePredicted2,
    ↪  degree, predictors, 0);
}
```

Listing A6: Multiple Polynomial Regression

tools described in the previous chapter and therefore we created our workaround for that, namely, we reused the implementation of Multiple Linear Regression implemented in JSAT. So, first, we removed redundant columns from the train and to-be-predicted sample calling `excludeNonPredictors()`. And then added columns with predictors in the degree, which is passed as a parameter to the method, using `addPowerColumns()` method. Then, a new dataset with power columns is passed to the Multiple Linear Regression method. The code for that is in Listing A6.

The other interesting implementation is `MultiDimensionalImputationMethod`. Its principle was already described in subsection Choosing Multiple Imputation Method and here we will provide the code explanation (Listing A7). So, at first in `fit()` method we define which method is the most appropriate using each predic-

tor separately by calling `simpleImputationMethods.imputeSimple()`, that has already been described along with Listing A2. Then in `predict()` method, we collect the predictions of each method and return their mean value.

```
public void fit () {
    // select best methods for each predictor separately
    SimpleImputationMethods simpleImputationMethods = new
    ↪  SimpleImputationMethods(datasetMissing);

    for (int columnPredictor : data.getColumnPredictors()) {
        MainData mainData = new MainData(new int[]{columnPredictor},
        ↪  columnPredicted, data.getDp(), data.getTrain(),
        ↪  data.getImpute(), data.isMultiple());
        methods.add(simpleImputationMethods.imputeSimple(mainData,
        ↪  stat));
    }
}

public double predict (DataPoint dp) {
    // get all the values predicted by the methods
    return methods.stream()
        .map(method -> {
            method.preprocessData();
            method.fit();
            return method.predict(dp);
        })
        //return average of all predictors
        .reduce(0.0, Double::sum) / methods.size();
}
```

Listing A7: Multi-Dimensional Imputation Method

Appendix B: All Results

In this appendix we would like to show all the results that were and were not shown in the main part of the document. The tables show all the test results, while the diagrams show only hybrid method results and best and worst methods among others. At some diagrams some of the metrics might be excluded due to big numbers which would worsen the quality of visualisation.

Results of Hydrometeorological Dataset

	MSE	RMSE	MAE	RSE,%	RRSE,%	RAE,%
Hybrid	1	1	0,71	1,03	10,16	8,71
Mean	97,03	9,85	8,2	100	100	100
Median	97,22	9,86	8,2	100,19	100,09	100
Multile Linear Regression	7,14	2,67	1,94	7,36	27,12	23,64
Multiple Polynomial Regression	1,41	1,19	0,83	1,45	12,04	10,11
Polynomial Curve Fitter (2 order, column 0)	1,41	1,19	0,83	1,45	12,04	10,11
Polynomial Curve Fitter (2 order, column 1)	7,82	2,8	2,02	8,06	28,39	24,64
Polynomial Curve Fitter (2 order, column 2)	7,82	2,8	2,02	8,06	28,39	24,64
Polynomial Curve Fitter (3 order, column 0)	1,26	1,12	0,79	1,29	11,38	9,65
Polynomial Curve Fitter (3 order, column 1)	7,82	2,8	2,02	8,06	28,39	24,64
Polynomial Curve Fitter (3 order, column 2)	7,82	2,8	2,02	8,06	28,39	24,64

Table B1: Performance measures of Atmosphere Temperature column

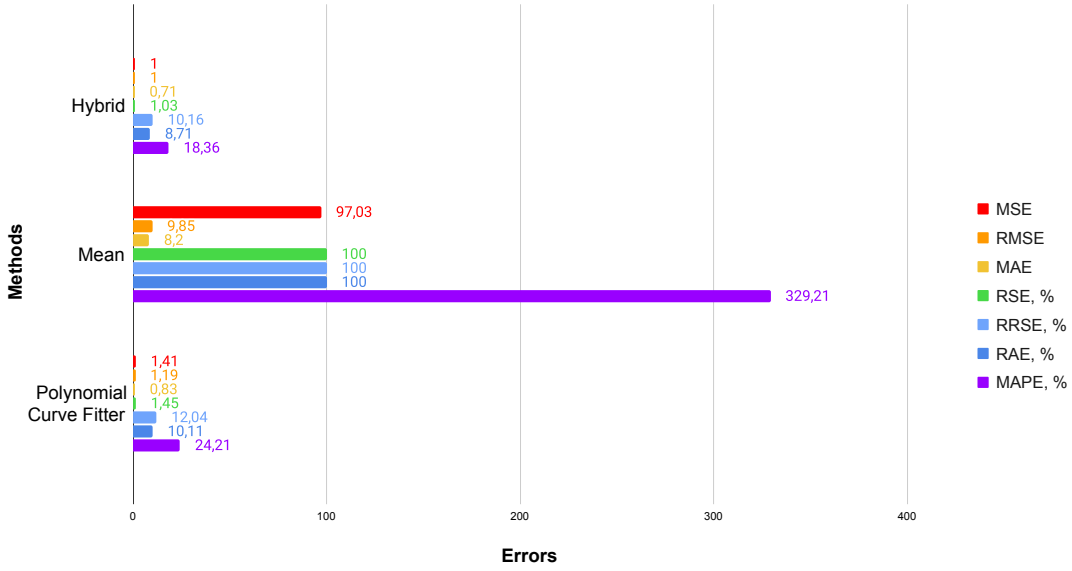


Figure B1: Performance measures of Atmosphere Temperature column

	MSE	RMSE	MAE	RSE,%	RRSE,%	RAE,%
Hybrid	4,08	2,02	0,4	6,69	25,87	6,72
Mean	60,9	7,8	5,92	100	100	100
Median	60,93	7,81	5,93	100,05	100,03	100,02
Multile Linear Regression	1,01	1,01	0,57	1,67	12,91	9,66
Multiple Polynomial Regression	2,56	1,6	0,38	4,2	20,49	6,41
Polynomial Curve Fitter (2 order, column 0)	2,56	1,6	0,38	4,2	20,49	6,41
Polynomial Curve Fitter (2 order, column 1)	1,2	1,09	0,63	1,96	14,01	10,64
Polynomial Curve Fitter (2 order, column 2)	1,2	1,09	0,63	1,96	14,01	10,64
Polynomial Curve Fitter (3 order, column 0)	3,43	1,85	0,38	5,63	23,72	6,43
Polynomial Curve Fitter (3 order, column 1)	1,2	1,09	0,63	1,96	14,01	10,64
Polynomial Curve Fitter (3 order, column 2)	1,2	1,09	0,63	1,96	14,01	10,64

Table B2: Performance measures of Atmosphere Pressure column

	MSE	RMSE	MAE	RSE,%	RRSE,%	RAE,%
Hybrid	31,84	5,64	3,9	8,71	29,51	24,06
Mean	365,64	19,12	16,2	100	100	100,03
Median	387,46	19,68	15,96	105,97	102,94	98,54
Multile Linear Regression	104,27	10,21	7,55	28,52	53,4	46,61
Multiple Polynomial Regression	228,33	15,11	4,59	62,44	79,02	28,34
Polynomial Curve Fitter (2 order, column 0)	35,22	5,93	4,13	9,63	31,04	25,52
Polynomial Curve Fitter (2 order, column 1)	112,1	10,59	7,84	30,66	55,37	48,4
Polynomial Curve Fitter (2 order, column 2)	112,1	10,59	7,84	30,66	55,37	48,4
Polynomial Curve Fitter (3 order, column 0)	33,24	5,77	4,03	9,09	30,15	24,87
Polynomial Curve Fitter (3 order, column 1)	112,1	10,59	7,84	30,66	55,37	48,4
Polynomial Curve Fitter (3 order, column 2)	112,1	10,59	7,84	30,66	55,37	48,4

Table B3: Performance measures of Relative Humidity column

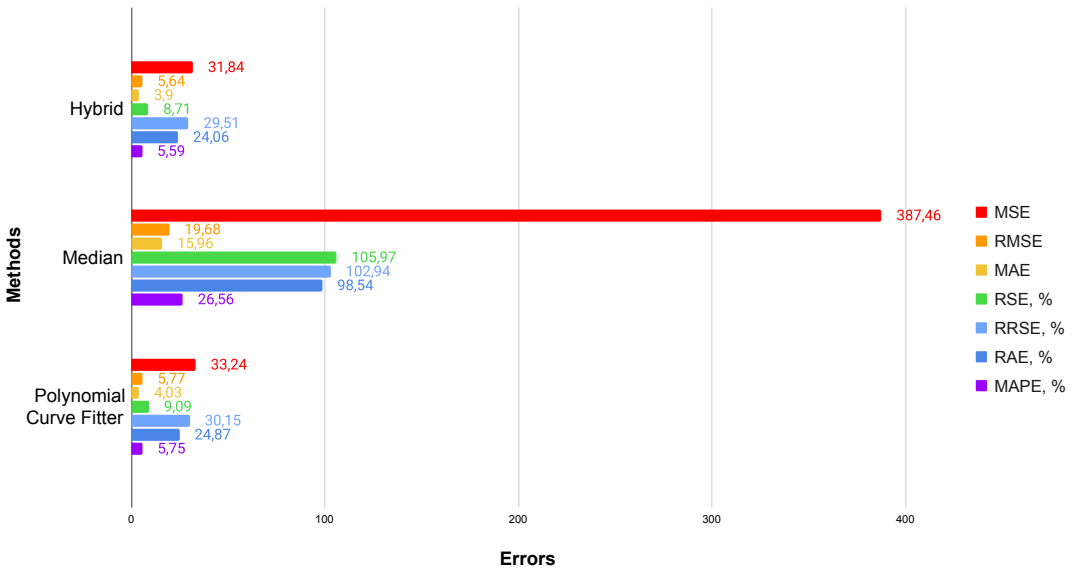


Figure B2: Performance measures of Relative Humidity column

	MSE	RMSE	MAE	RSE,%	RRSE,%	RAE,%
Hybrid	0,78	0,88	0,65	30,16	54,92	53,01
Mean	2,59	1,61	1,22	100,01	100,01	99,97
Median	2,73	1,65	1,19	105,6	102,76	97,25
Multile Linear Regression	0,96	0,98	0,74	37,25	61,03	60,11
Multiple Polynomial Regression	0,78	0,89	0,65	30,3	55,05	53,1
Polynomial Curve Fitter (2 order, column 0)	0,78	0,89	0,65	30,3	55,05	53,1
Polynomial Curve Fitter (2 order, column 1)	1	1	0,74	38,53	62,07	60,74
Polynomial Curve Fitter (2 order, column 2)	1	1	0,74	38,53	62,07	60,74
Polynomial Curve Fitter (3 order, column 0)	0,78	0,89	0,65	30,33	55,07	52,76
Polynomial Curve Fitter (3 order, column 1)	1	1	0,74	38,53	62,07	60,74
Polynomial Curve Fitter (3 order, column 2)	1	1	0,74	38,53	62,07	60,74

Table B4: Performance measures of Wind Speed column

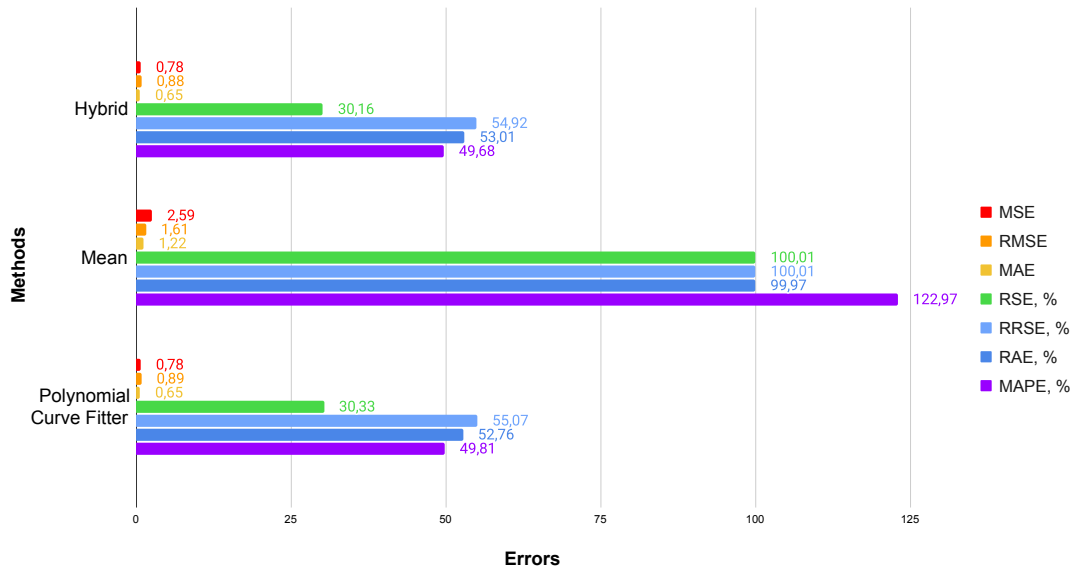


Figure B3: Performance measures of Wind Speed column

	MSE	RMSE	MAE	RSE,%	RRSE,%	RAE,%
Hybrid	6822,22	82,6	53,31	57,36	75,74	54,8
Mean	11893,97	109,06	97,28	100	100	100
Median	11968,42	109,4	97,19	100,63	100,31	99,91
Multile Linear Regression	6417,22	80,11	56,53	53,95	73,45	58,11
Multiple Polynomial Regression	6676,93	81,71	52,91	56,14	74,93	54,39
Polynomial Curve Fitter (2 order, column 0)	6676,93	81,71	52,91	56,14	74,93	54,39
Polynomial Curve Fitter (2 order, column 1)	6426,59	80,17	56,63	54,03	73,51	58,22
Polynomial Curve Fitter (2 order, column 2)	6426,59	80,17	56,63	54,03	73,51	58,22
Polynomial Curve Fitter (3 order, column 0)	6673,5	81,69	52,57	56,11	74,91	54,04
Polynomial Curve Fitter (3 order, column 1)	6426,59	80,17	56,63	54,03	73,51	58,22
Polynomial Curve Fitter (3 order, column 2)	6426,59	80,17	56,63	54,03	73,51	58,22

Table B5: Performance measures of Wind Direction column

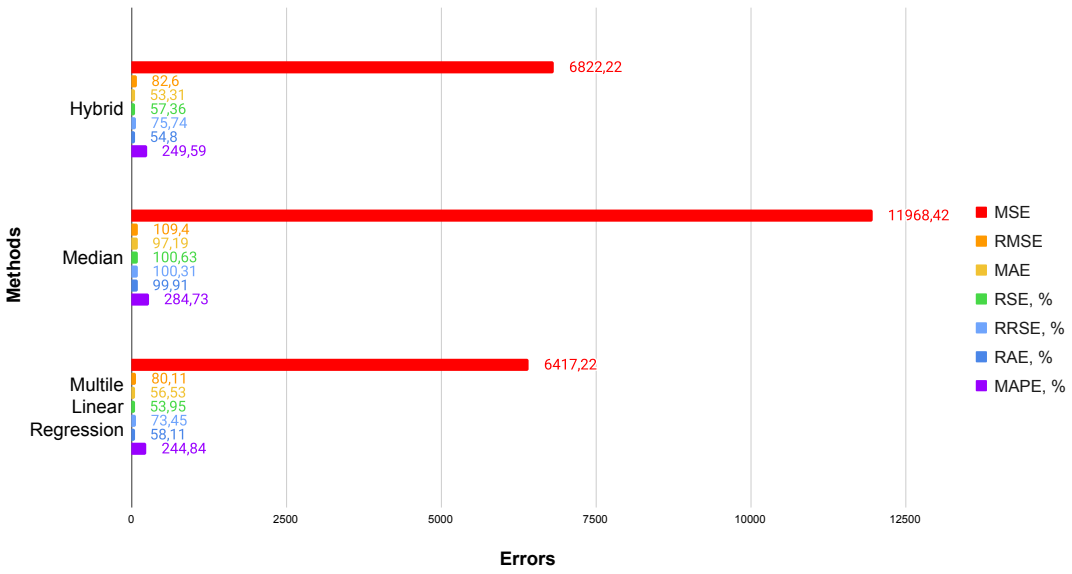


Figure B4: Performance measures of Wind Direction column

Results of CCPP Dataset

	MSE	RMSE	MAE	RSE,%	RRSE,%	RAE,%
Hybrid	62,86	7,93	6,74	110,41	105,08	103,51
Mean	56,95	7,55	6,52	100,02	100,01	100,02
Median	56,91	7,54	6,47	99,96	99,98	99,28
Linear Regression	63,36	7,96	6,75	111,28	105,49	103,59
Polynomial Curve Fitter (2 order)	68,77	8,29	7	120,79	109,9	107,41
Polynomial Curve Fitter (3 order)	69,11	8,31	7,03	121,38	110,17	107,86

Table B6: Performance measures of Temperature column

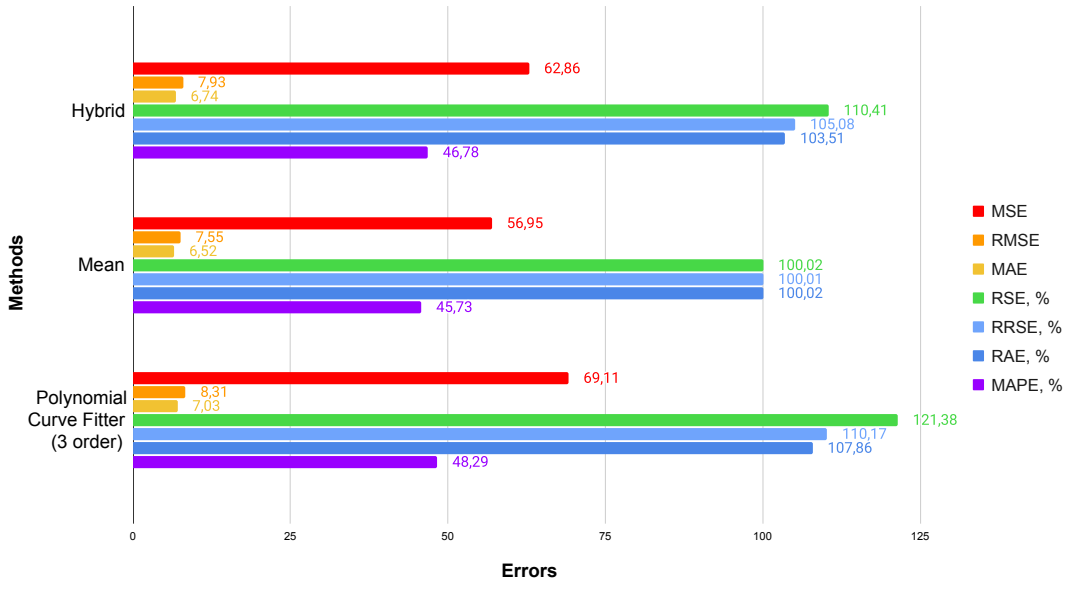


Figure B5: Performance measures of Temperature column

	MSE	RMSE	MAE	RSE,%	RRSE,%	RAE,%
Hybrid	179,71	13,41	11,99	111,1	105,4	102,61
Mean	161,9	12,72	11,7	100,09	100,04	100,06
Median	160,76	12,68	11,52	99,38	99,69	98,56
Linear Regression	179,46	13,4	11,96	110,94	105,33	102,34
Polynomial Curve Fitter (2 order)	196,63	14,02	12,21	121,55	110,25	104,48
Polynomial Curve Fitter (3 order)	197,4	14,05	12,21	122,03	110,47	104,45

Table B7

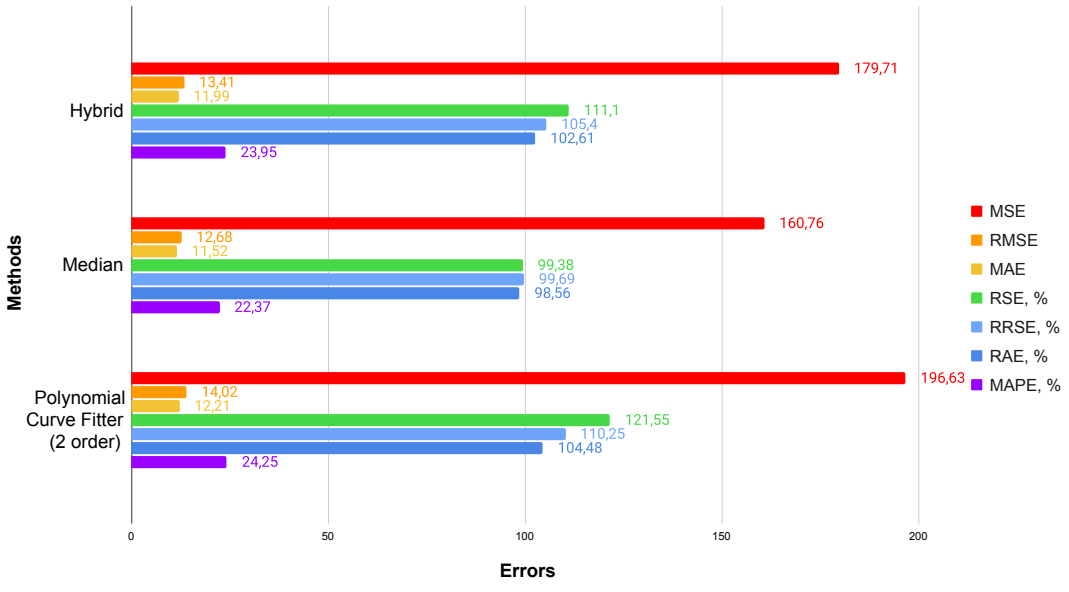


Figure B6: Performance measures of Relative Humidity column

	MSE	RMSE	MAE	RSE,%	RRSE,%	RAE,%
Hybrid	38,28	6,19	4,97	108,61	104,22	103,95
Mean	35,26	5,94	4,78	100,04	100,02	100,03
Median	35,13	5,93	4,77	99,68	99,84	99,7
Linear Regression	39,24	6,26	5,04	111,33	105,51	105,53
Polynomial Curve Fitter (2 order)	41,87	6,47	5,22	118,79	108,99	109,31
Polynomial Curve Fitter (3 order)	41,8	6,47	5,22	118,6	108,91	109,23

Table B8: Performance measures of Ambient Pressure column

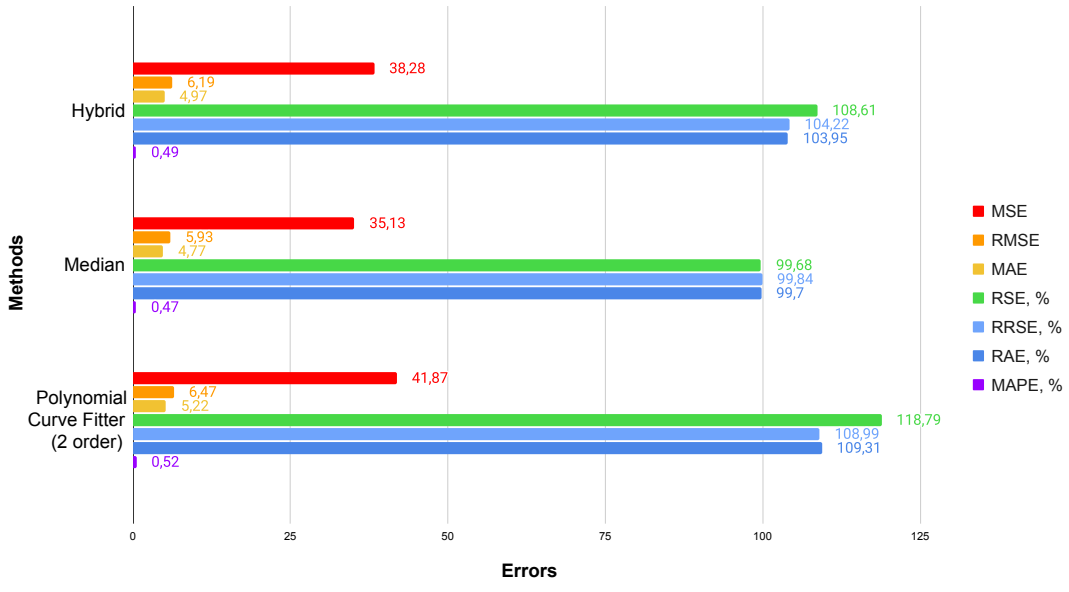


Figure B7: Performance measures of Ambient Pressure column

	MSE	RMSE	MAE	RSE,%	RRSE,%	RAE,%
Hybrid	221,82	14,89	12,03	105,2	102,57	101,86
Mean	211,2	14,53	11,82	100,16	100,08	100,02
Median	220,01	14,83	11,92	104,34	102,15	100,92
Linear Regression	226,43	15,05	12,21	107,38	103,63	103,33
Polynomial Curve Fitter (2 order)	243,03	15,59	12,52	115,26	107,36	105,94
Polynomial Curve Fitter (3 order)	244,01	15,62	12,57	115,72	107,58	106,39

Table B9: Performance measures of Exhaust Vacuum column

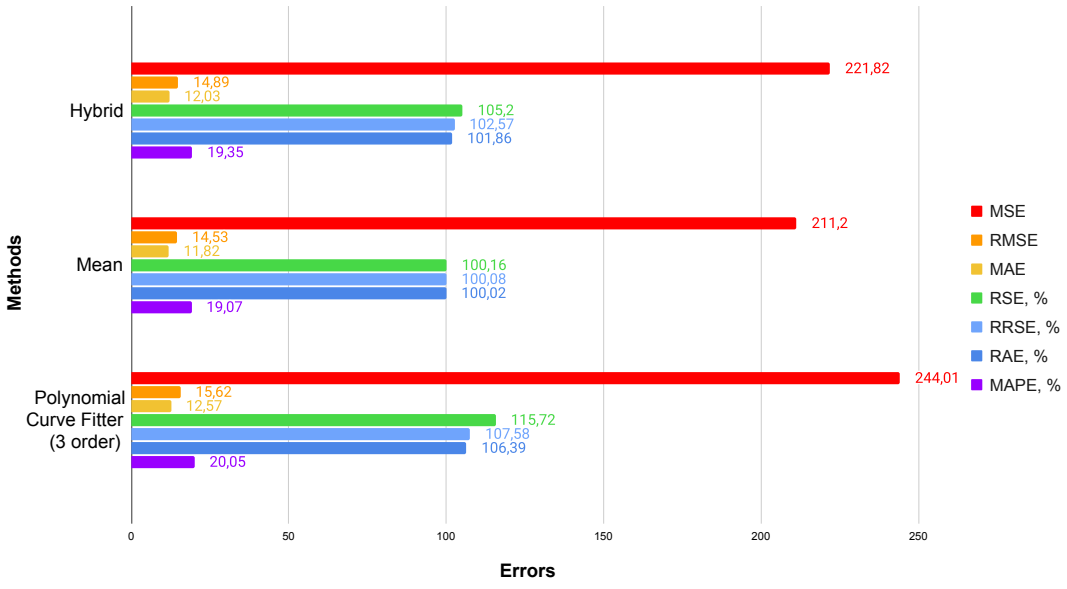


Figure B8: Performance measures of Exhaust Vacuum column

	MSE	RMSE	MAE	RSE,%	RRSE,%	RAE,%
Hybrid	310,71	17,63	15,18	103,2	101,59	100,4
Mean	301,21	17,36	15,13	100,04	100,02	100,06
Median	302,77	17,4	14,82	100,56	100,28	98,01
Linear Regression	317,86	17,83	15,31	105,57	102,75	101,29
Polynomial Curve Fitter (2 order)	348,45	18,67	15,75	115,73	107,58	104,21
Polynomial Curve Fitter (3 order)	353,84	18,81	15,85	117,52	108,41	104,87

Table B10: Performance measures of Net hourly electrical energy output column

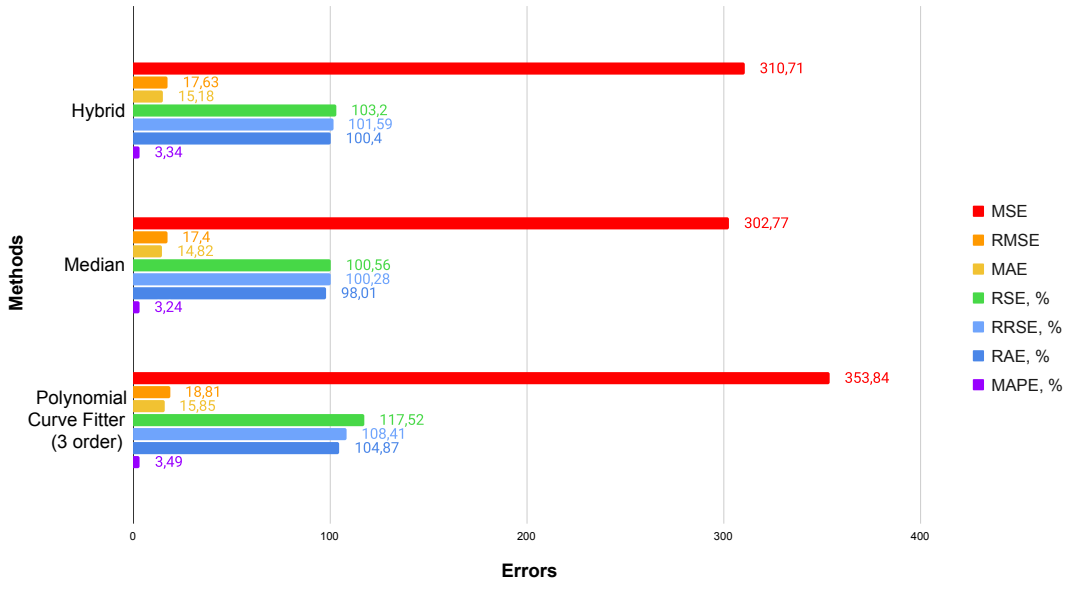


Figure B9: Performance measures of Net hourly electrical energy output column

Results of Yellowstone Park Dataset

	MSE	RMSE	MAE	RSE,%	RRSE,%	RAE,%
Hybrid	200,08	14,14	9,9	0,8	8,95	7,25
Mean	24998,53	158,11	136,54	100	100	99,99
Median	25339,18	159,18	136,37	101,36	100,68	99,87
Multile Linear Regression	461,75	21,49	16,29	1,85	13,59	11,93
Multiple Polynomial Regression	215,82	14,69	10,36	0,86	9,29	7,59
Polynomial Curve Fitter (2 order, column 0)	547,9	23,41	17,69	2,19	14,8	12,96
Polynomial Curve Fitter (2 order, column 1)	215,84	14,69	10,36	0,86	9,29	7,59
Polynomial Curve Fitter (3 order, column 0)	547,9	23,41	17,69	2,19	14,8	12,96
Polynomial Curve Fitter (3 order, column 1)	201,73	14,2	9,92	0,81	8,98	7,26

Table B11: Performance measures of Altitude column

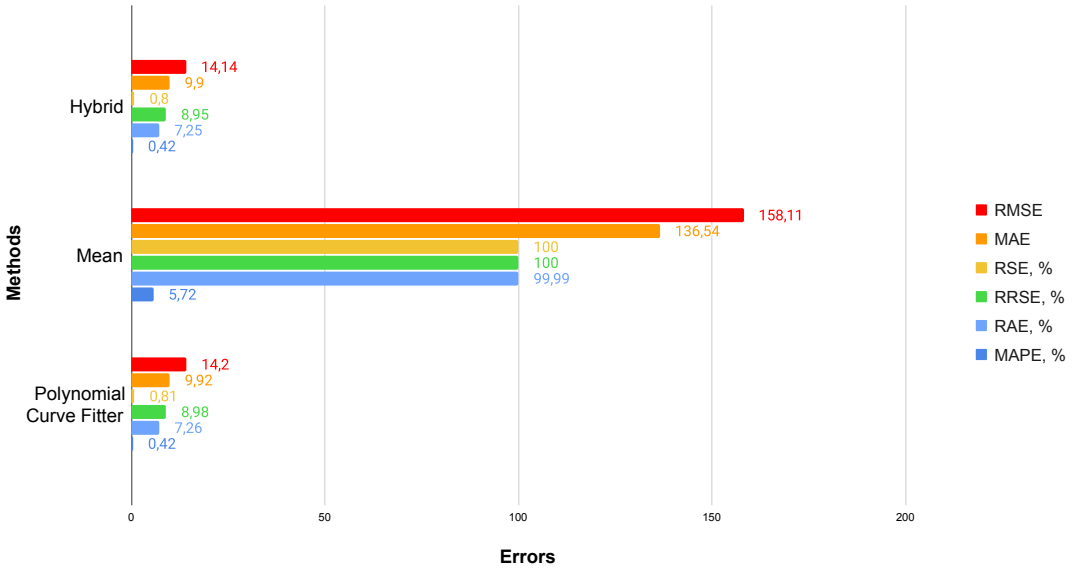


Figure B10: Performance measures of Altitude column

Appendix C: User Guide

Installation

To use the product:

1. Create a project in IDE of your choice (during implementation, IntelliJ IDEA was used).
2. Add files from CD to the `src/com.company/` directory.
3. Add `.jar` files to the classpath of the project.
4. Build the project.

User Input and Output

As it was described in section 3.2, there are two possible ways of input. The main is via `config.properties` file. The values in it are used whenever `input.useConfigValues` is true. The properties (keys) in this file are explained in Table C1. The ones marked with * are always read from the file and in no case are asked from a user in the console. Otherwise, the program waits for input from the user in the console for the values which are not marked with *.

Key	*Description*
input.useConfigValues*	on true, all the program parameters are taken from `config.properties` file and no console input is needed
input.completeDataset	path to the complete dataset, if none specified, there will not be any performance evaluation of the hybrid method in the end
input.incompleteDataset	path to the dataset with missing values to be imputed

input.predictors	indexes of the predictors' columns (starting from 0), separated with space
input.predicted	index of column to be imputed (starting from 0), in case all non-predictors should be imputed use -1
input.printOnlyFinal*	on true, only only performance measures and descriptive statistics of predicted columns are printed out. Otherwise each data (method used, actual and predicted value, MAPE of a single imputation) are also printed out while running the program
input.runTest*	on true, all missing values are imputed using one imputation method returned from com.company.HybridMethod#getTestMethod
input.prepareTraining*	on true, in case input.runTest is true, the training dataset for the tested method is prepared the same way as for hybrid one. Otherwise the dataset is simply divided into training and to be predicted parts - with or without missing value in column that is passed as a parameter respectively.
impute.isZeroMissing*	on true, 0 will be treated as a missing value
impute.weighted*	on true, weighted predicting is used for polynomial and linear multiple imputation methods

impute.useMultiDimensionalAsDefault*	on true, multi dimensional method is used as a default in case of multiple imputation
output.skipSaving*	on true, writing dataset with imputed values into `.csv` file will be skipped
output.filename	path to the |.csv| file which should contain dataset with imputed missing values

Table C1: Structure of config.properties

Simple and Multiple Imputation Test Data

The test data for Combined Cycle Power Plant dataset is in Listing C1.

```
input.incompleteDataset= src/com/company/data/CCPP_missing.csv
input.completeDataset= src/com/company/data/CCPP_complete.csv
input.predictors=0
```

Listing C1: Combined Cycle Power Plant Test Data

The test data for dataset from Slovak Hydrometeorological Institute is Listing C2.

```
input.incompleteDataset=
↪  src/com/company/data/combined_multiple_missing.csv
input.completeDataset=
↪  src/com/company/data/combined_multiple_complete.csv
input.predictors=0 1 2
```

Listing C2: Slovak Hydrometeorological Institute Test Data

The test data for Yellowstone dataset is Listing C3.

```
input.incompleteDataset=
↪  src/com/company/data/YellowStoneElev_missing.csv
input.completeDataset=
↪  src/com/company/data/YellowStoneElev_complete.csv
input.predictors=0 1
```

Listing C3: Yellowstone Test Data

Appendix D: Work Plan

Work plan for WS 2021/22:

20.09 - 01.10	Analyse the problem and types of missing values, create a vision of the work that should be done during work.
04.10 - 15.10	Make a closer look into the variety of imputation methods, outline their pros and cons. Create GitHub repository and project in Overleaf.
18.10 - 29.10	Analyse available tools and libraries to be used for implementation.
01.11 - 12.11	Make a basic structure of the future implementation, prepare datasets using written scripts in Python.
15.11 - 26.11	Make an attempt to impute missing values using a few methods from analysed tools.
29.11 - 10.12	Implement main methods for simple imputation and one for multiple, look into the topic of measuring performance of the implemented hybrid method.
13.12 - 24.12	Finalize the work, check spelling and grammar, correct all small issues and uncertainties, hand in.

Work plan for SS 2021/22:

14.02 - 25.02	Make a deeper look into multiple imputation.
28.02 - 11.03	Prepare dataset for multiple imputation and implement multiple imputation methods.
14.03 - 25.03	Start with the second part of the document - Methodology.
28.03 - 08.04	Results.
11.04 - 22.04	Technical documentation.
25.04 - 06.05	Resume and Appendixes.
09.05 - 16.05	Finalization.

Appendix E: Description of Digital Part

Evidence number of thesis in system: FIIT-5212-98938.

Content of the digital part (ZIP archive):

Directory	*Description*
\data	directory with datasets and scripts which preprocess data for the implemented hybrid method
\imputationMethods	implemented imputation methods and algorithms of their selection
ImputationMethod.java	the interface of each imputation method implemented
SimpleImputationMethods.java	algorithm for simple imputation method selection
MultipleImputationMetods.java	algorithm for multiple imputation method selection
...	imputation methods implementation
\tools	source with .jar files of the libraries used for implementation
\utils	classes with helper methods
\calculations	files with statistical and mathematical calculations
\objects	objects which do not have any functionality but are used to store relatable information side by side
ColorFormat.java	class which contains encodings of colorful print
DatasetManipulation.java	class with functionality that does manipulations with the dataset, e.g. print, write, read, *etc.*
Evaluation.java	class performs the evaluation and stores data needed for that
config.properties	file with stored parameters for program
ConfigManager.java	class that manages parameters read from config.properties
HybridMethod.java	hybrid method itself
Main.java	class where the program starts, responsible for input and output

The name of the submitted archive: BP_OlesiaMelnychyn.zip