Automatic Data Cleaning Tool for OpenML

Sander Schuitemaker
Eindhoven University of Technology / Eindhoven, 5612 AZ, The Netherlands
Tilburg University / Tilburg, 5037 AB, The Netherlands
s.m.schuitemaker@student.tue.nl
Supervisor: Joaquin Vanschoren

[ ]

Abstract

This paper describes the process of the creation of a data cleaning tool for OpenML. The tool consists of an error detection and an error correction section, in which 8 different, common data errors are addressed. Each data error has its own detection and correction algorithms implemented in the tool. Additionally, the tool actively engages the user in the tool following the Human In The Loop (HITL) principles. Any false error detections, undetected errors, or wrong error correction recommendations can be corrected by the user. Summary statistics, colored badges, colored errors, and textual information are used to make the tool as intuitive and user-friendly as possible. The performance of the tool was tested in a downstream model experiment. 20 datasets were gathered from OpenML, 10 containing a classification task and 10 containing a regression task. The task was performed on both the original dataset as well as on the dataset cleaned by the tool and the performances of both dataset versions were compared. The results showed that the 'cleaned' dataset versions in general obtained a higher performance.

1 Introduction

Machine learning (ML) and Artificial Intelligence (AI) have been rapidly growing in recent years, with many new sectors integrating the concept into their work (e.g. healthcare (Miotto et al., 2018), biology (Webb et al., 2018), finance (Ozbayoglu et al., 2020)). For years, the ML community focused mainly on designing robust-to-noise-algorithms that worked with dirty data as input (Li et al., 2021), instead of addressing the origin of the issue: the data itself. Recently, this has changed, with researchers shifting attention from advancing model design to enhancing data quality resulting in the concept of data-centric AI (DCAI) (Zha et al., 2023). Jakubik et al. (2024) defined DCAI as *the paradigm emphasizing that the systematic design and engineering of data are essential for building effective and efficient AI-based systems*. One process to ensure high data quality is data cleaning. Data cleaning is defined by Brownlee (2020) as *the process of identifying and correcting mistakes or errors in the data*. Examples of such data errors are missing values, duplicate instances, outlier values, and instances, etc.. This is a crucial step in the Machine Learning pipeline, as previous research has shown that dirty (not cleaned) data leads to unreliable results (e.g. Kim et al., 2003, Whang and Lee, 2020). The consequences of unreliable results can be detrimental, especially if one does not realize that the results are untrustworthy. Several studies have shown that poor data quality leads to unfair decisions with negative societal impacts (e.g. Stoyanovich et al., 2020, Yang et al., 2020, Bender et al., 2021). It is thus crucial to remove incomplete, inaccurate, inconsistent, and irrelevant data, but this often requires some basic data science knowledge. This paper aims to (partially) automate this process of data cleaning by creating a tool that guides a user through the errors in his/her dataset and helps to correct them.

1.1 Research setting

(semi-)Automated data cleaning tools already exist (e.g. Li et al., 2021, Mahdavi et al., 2019, Krishnan et al., 2016, Neutatz et al., 2019). However, this paper creates a tool in a specific context: that of OpenML[1]. OpenML is a Machine Learning platform that shares the greatest results from ML researchers all over the world on their platform. One of the key components of their platform is the extensive database of 5.000+ datasets that are openly available for anyone to download and use in their projects. The uploading of a

[1] https://www.openml.org/

dataset to OpenML is currently done through an API, which can make it difficult for people without data science knowledge to contribute a dataset to OpenML. Therefore, a colleague of mine is working on an uploading interface, which uses techniques to improve the meta-data quality of a dataset. However, one part that remains untouched is the quality of the data within the uploaded dataset itself. Especially if a dataset is uploaded by someone without (much) data science knowledge, can we expect them to correct all data errors in the dataset by themselves, before uploading it to OpenML? That is the problem this research paper hopes to address. This paper will describe the process of designing a data cleaning tool, which guides users who uploaded a dataset to OpenML through the different data errors in their dataset and provides recommendations on how to correct/repair those errors. The ultimate goal is to improve the quality of the datasets on OpenML, which can improve the quality of further research done using OpenML's datasets.

1.2 Research question

The research question this paper aims to address is the following: *How can an 1) automated data cleaning process be created and integrated into the platform of OpenML, in the form of a 2) user interface, 3) to enhance the overall quality of datasets on OpenML?*

The research question consists of 3 crucial parts, which will be discussed in this paper. These parts address the following 3 broad questions: what, how, and why. What this paper describes is the creation and integration of an 1) automated data cleaning process into the platform of OpenML. How this will be accomplished is in the form of a 2) user interface. And why this subject is discussed is 3) to enhance the overall quality of datasets on OpenML.

1) Automated data cleaning process refers to the data error detection and data error correction steps. In the data error detection step, several different error detection techniques will be used to detect 8 different error types, which will be discussed in Section 2.2. Afterwards, the data error correction step will provide recommendations on how to correct each error type based on several different error correction techniques (also discussed in Section 2.2). Although the vast majority of this process will be automated, it will not be fully automated. The Human-In-The-Loop (HITL) principles will be applied to check the detections and corrections of the tool. For the detection step, this means that the user is able to correct false positives (an error is detected by the tool as such, but in reality, it is no error) and add false negatives (an error is not detected as such by the tool, but in reality, it is an error). For the correction step, this means that the tool provides recommendations of correction techniques for each error, but the user can change the correction technique if he/she wants to.

2) User interface refers to an interface where the user will be guided through the different errors detected and the different corrections recommended by the tool. This interface will be created in Plotly's Dash[2] and incorporated into the personalized dataset page of a user's dataset on OpenML. Several techniques will be incorporated into the user interface to make the process as optimal and user-friendly as possible for the user.

3) To enhance the overall quality of datasets on OpenML refers to the ultimate goal of improving the reliability, usability, and trustworthiness of datasets hosted on OpenML. The data cleaning process will ensure that there are less erroneous datasets on the platform. This will ensure that users can more confidently utilize OpenML's datasets in their experiments and analyses. Quality of datasets is quite a difficult concept to objectively measure. How we have addressed this will be discussed in Section 2.4.

2 Methodology

2.1 Set-Up

Before applying any of the error detection and/or correction techniques, the dataset, in whatever format it is, will be converted to a Pandas[3] DataFrame. Pandas DataFrames are tables with attributes as columns and indices as rows. They are easily adjustable with many different functionalities, which is why they are the basis for the tool.

[2] https://dash.plotly.com/
[3] https://pandas.pydata.org/

2.2 Error detection and correction

Several data errors are detected and corrected by the tool. We will now for every error discuss 1) what it is (with examples), 2) why it should be addressed, 3) which technique was used to detect it and why, and 4) which technique was recommended to correct it and why. We will discuss the errors in order of importance, which will also be the order in which the tool will detect and correct them. Crucial to note for the correction section is that we comply with the HITL principles. This means that all error corrections described below are just recommendations to the user. The user always has the power to deny a correction or choose a different correction.

2.2.1 Missing values

The tool considers two types of missing values: (explicit) missing values and disguised missing values. Both will be discussed.

(Explicit) Missing values

Explicit missing values refers to the classical ways in which missing values are represented in datasets, including NaN, NA, None, and NULL among others. There are also other ways of representing missing values, for example using a question mark (?), a dash (-), a zero (0), or the aforementioned NaN, NA, None, and NULL but then in string format (i.e. "NaN", "NA", "None" and "NULL"). However, it is not necessarily always the case that these characters represent missing values (e.g. in a column called "end-of-sequence tokens", a "?" probably will not represent a missing value, or "NA" might refer to "North-America" in a column called "continent"). How such cases are handled, will be discussed in the next section on disguised missing values.

There are 2 main reasons why missing values need to be dealt with. The first reason is that many ML models cannot handle missing data (i.e. it returns an error) (Lakshminarayan et al., 1999). The second reason is because missing data introduces an element of ambiguity, which affects the properties of statistical estimators, results in loss of power, and leads to misleading conclusions (Schmitt et al., 2015, Somasundaram and Nedunchezhian, 2011, Alabadla et al., 2022).

The technique used to detect explicit missing values is quite simple. The function pd.isnull() is used on a column to create a mask, which is "True" if an index in the column is a missing value and "False" if it is not.

Disguised missing values

Disguised missing values (DMV) refer to missing values in a dataset that do not appear as such. For example, for an attribute "phone number" the value +111 111 1111 is not a legitimate value but does also not appear as a missing value. Probably, a user did not want to fill in his/her phone number, so just pressed the 1 key 10 times. This value is thus actually a missing value, but cannot be detected with the technique we described in the previous section: it is a **disguised** missing value. This example demonstrates one of the 5 possible DMV types as defined by Qahtan et al. (2018), namely a string with repeated characters (or characters next to each other on the keyboard). The other 4 types of DMVs are 1) out-of-range values (e.g. -1 for an attribute which only takes positive values), 2) outliers (e.g. 1.000 for attribute age), 3) values with non-conforming data types (e.g. if a string value is missing, replace with a numerical value and vice versa) and 4) inliers (values that are randomly distributed within the range of the data and are used frequently in the dataset, e.g. if a user fills in a legitimate address, but not his/her own address).

The first reason for why missing values should be dealt with described in the previous section does not apply to DMVs; models can run perfectly fine with them incorporated. The second reason still holds though, the element of ambiguity of a DMV leads to unreliable results.

Part of Qahtan et al. (2018)'s technique is used to detect disguised missing values. They use a syntactic pattern discovery algorithm to detect the 1) out-of-range values, 2) strings with repeated characters, and 3) values with non-conforming data types. This algorithm generates syntactic patterns for every attribute that describe most values in that attribute. If a value does not comply with one of those patterns, it is labeled as a DMV. Secondly, they use numerical outlier detection to tackle the outlier DMVs. The challenges here are that DMVs might not be recognized as outliers, because they occur frequently. Furthermore, not all outliers are DMVs. They tackle this by using a probability density function (PDF), which ignores

duplicates when applying the outlierness test for a value. Therefore, it does not matter if DMVs occur frequently. Lastly, they also incorporated an algorithm to detect DMVs as inliers. However, we will not use that algorithm as the performance is not too great (since inliers are more difficult to detect) and the algorithm takes quite long to run. In addition to Qahtan et al. (2018)'s DMV detection algorithms, we will also implement two functions of Deepchecks[4] to find any undetected DMV's. The functions we will use are SpecialCharacters() and MixedNulls(). SpecialCharacters() finds any data values that consist of solely special characters, like a question mark (?). MixedNulls() finds any data values that are Null or NaN values, but might be encoded differently, for example in a string ('Null') or as a zero (0).

Correction

A lot of research has been conducted on the topic of 'missing value correction'. Many papers have proposed new imputation methods that perform supposedly better than existing methods. However, there are two recurring problems with the results of all these papers. Firstly, the papers do not compare their method to all other methods, but rather just to the basic 'baseline' imputation methods, such as mean imputation and K-Nearest Neighbors imputation (KNNI). This makes it hard to compare two new, proposed methods. Secondly, the results of the papers are heavily context-specific. Whether a new imputation method is better than existing methods often depends on several factors. Examples of such factors include the number of datasets on which the methods were compared, the number of features and instances in those datasets, the missingness ratios (MR) in those datasets (e.g. only small ratios or ranges of ratios), the feature types in those datasets (e.g. only categorical, only numerical, or mixed), the mechanism that caused the missing data (MCAR, MAR, MNAR or a combination), how the performance was measured (accuracy / RMSE on ground truth values or performance of downstream model), what downstream model was used (if any) and the domain-context of the datasets (e.g. medical dataset or business dataset). In Table 3 in the Appendix, we compare the results of several different papers with respect to the conditions used in their study.

Table 3 demonstrates that there does not exist one method that always performs the best. Therefore, in this paper, we apply a framework for the correction of missing values that consists of a series of steps to find the best suitable imputation method for a dataset. This framework is displayed in Figure 1.

The first step checks whether the dataset contains 1) only numerical, 2) only categorical, or 3) mixed feature types. If the dataset contains only numerical or only categorical features, then the second step is to calculate the dataset size. For the numerical dataset, if the dataset size is small (<10.000 values), the proposed method will be Multilayer Perceptron imputation (MLP), and if the dataset is large (>10.000 values) the proposed method will be Classification and Regression Tree imputation (CART). This is mainly based on the results of Tsai and Hu (2022), who concluded that MLP is the best method for numerical values, but since MLP takes almost twice as long as CART for the imputation, MLP is only used for small datasets and CART will be used for large datasets. For the categorical dataset, Tsai and Hu (2022) showed that CART is also an excellent method for categorical features. Additionally, many other studies showed that Random Forest imputation (RF) is also a good method for categorical (and mixed) features (e.g. Tsai and Hu, 2022, Jäger et al., 2021, Poulos and Valle, 2018, Penone et al., 2014). RF is essentially the same as CART, but instead of one decision tree, multiple trees are constructed and then aggregated. This makes RF models less prone to overfitting and more robust to outliers than CART models, but this comes at the cost of increased training and prediction time. Because time is quite an important factor in our case, we do not want to use RFs for large datasets. Therefore, we made the decision to propose RF if the dataset size is small and CART if the dataset size is large.

Finally, if the dataset contains numerical and categorical features (mixed), then the second step is to calculate the missingness ratio (MR). If the MR < 10%, the KNN method will be proposed. This is because almost all studies mentioned before in Table 3 tested the KNN method and in almost all those studies was it ranked among the best methods. It is one of the most popular MV imputation methods of all time and there is a reason for that. Therefore, we had to include this method and since Tsai and Hu (2022) showed that it performed well for small MRs on mixed datasets, we propose it for mixed datasets with an MR < 10%. For mixed datasets with an MR > 10%, the same procedure as for categorical datasets

[4]https://docs.deepchecks.com/stable/tabular/auto_checks/data_integrity/index.html

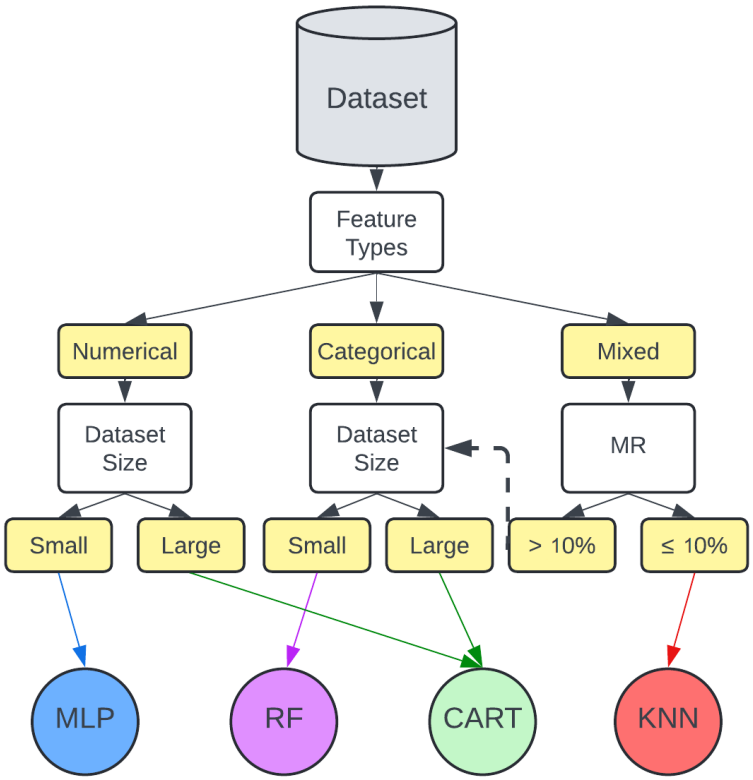


Figure 1: Framework for determining missing value imputation method

is used, namely to use RF for small datasets and CART for large datasets.

As mentioned before, the methods proposed are just recommendations. The user is able to change the imputation method in the dashboard. In addition to the methods described in the previous section (MLP, CART, RF, and KNN), the user can select from the "standard" imputation methods Mean, Median, and Mode (note that Mean and Median imputation are not possible for categorical attributes), and the options "Remove" and "Keep". "Remove" removes the instances containing the missing values and "Keep" simply retains the missing values in the dataset. The methods discussed in the previous section will be (if approved by the user) applied to all columns of the dataset (that contain missing values). However, if the user wants to select a different imputation method, he/she can select a method per attribute/column. Thus, for every column a dropdown menu is available with the methods: MLP, CART, RF, KNN, Mode, "Remove" and "Keep" (and Mean and Median for numerical attributes). Additionally, the user can choose the option "Remove rows", which removes all instances containing a missing value from the dataset, or the option "Keep all", which retains all missing values in the dataset.

2.2.2 Duplicates

The tool considers two types of duplicates: duplicate instances and duplicate attributes. Both will be discussed.

Duplicate instances

Duplicate instances are instances where the entire instance (excluding the index attribute) exists more than once in the dataset. For example, consider a dataset with attributes name, university, and program. A duplicate instance would then be (for example) "Sander, TU Eindhoven, Data Science" as multiple Sanders could study Data Science at TU Eindhoven.

There are multiple reasons why duplicate instances should be dealt with. The first danger of having duplicate instances is overfitting. Because instances occur more than once, the model will fit too closely

to the training data, which can hinder its performance on unseen instances. Furthermore, it is important for dimensionality reduction, which leads to better time performances of models (Abbas et al., 2023).

The technique used to detect duplicate instances is the pd.duplicated() function. This returns (by default) all duplicate instances except the first instance. This is preferred, because one does not want to remove all instances that have a duplicate, but rather only the duplicates, so every unique instance in the dataset is kept. Furthermore, since duplicate instances are the same, it should also not matter which instance is kept.

Duplicate attributes

Duplicate attributes are attributes for which every value in an attribute (in the same order) is already present in a different attribute. For example, consider the case of a dataset with columns "capital" and "country", which has been sliced to only the cases where country is equal to Luxembourg or Monaco. This slice means that automatically all values of the capital column will also only be Luxembourg and Monaco. Furthermore, the columns capital and country are the same, although they represent different real-world entities.

It is important to remove duplicate attributes for dimensionality reduction. This ensures that models can be ran faster because they have to take into account less attributes.

The technique to detect duplicate attributes is almost the same as for the duplicate instances, except that before applying the pd.duplicated() function, the dataset is first transposed. This basically entails that the attributes become the indices and the indices become the attributes. This is done because the pd.duplicated() function works over rows instead of over columns.

Correction

The technique used to correct duplicates (both instances and attributes) is to retain the first occurring duplicate, in a duplicate group, in the dataset, and remove the other duplicates, belonging to that same duplicate group, from the dataset. This means that for any n number of duplicates in a duplicate group X, the first duplicate x_1 is retained in the dataset and the others $(x_2, ..., x_n)$ are removed. The user is able to choose a different correction technique choosing from "Remove all" (which removes all duplicates in all duplicate groups from the dataset), "Keep all" (which retains all duplicates in all duplicate groups in the dataset) and "Select keep/remove". The latter option allows the user to select per duplicate group, which duplicate instances/attributes should be retained in the dataset and which should be removed from the dataset.

2.2.3 Outliers

Outliers are samples of data that are exceptionally far from the mainstream of the data (Brownlee, 2020). Outliers can occur as single values in an attribute/column or as whole instances that do not fit the distribution of the dataset. The tool will handle both types of outliers.

Outlier values

Outlier values are thus values that do not fit the distribution of the column. For example, in a column named "age" where all values range between 0 and 100, a value of 200 would be an outlier. In this example, the outlier is probably a data error, which can occur due to multiple reasons, for example because of an input error or a data corruption. It is however also possible that the outlier is not a data error, but an actual real-world observation (Brownlee, 2020). For example, in a dataset with all football players of the "Eredivisie", a value of 32 for the "goals scored" column, although being an outlier if the other players in the dataset had between 0 and 20 goals, does not have to be a data error. This is important to consider for how we will detect outliers and how we will repair them. Although it is also possible to detect outliers in categorical attributes (Hellerstein, 2013), we will only consider outliers in numerical columns. This is because our DMV detection algorithms also already take into account outliers (of both numerical and categorical) values.

Outliers can have quite a negative impact on ML models. When training a model on data containing outliers, they could significantly reduce the accuracy of a model or even result in a biased model leading to an inaccurate classification (Perez and Tah, 2020). Therefore, it is important to consider correcting these values.

Several techniques exist for identifying outliers in a column of a dataset. As mentioned before, an outlier is not necessarily a data error. Another important remark is that we already performed (some)

outlier detection in the previous step of detecting DMVs. Therefore, we will use a lenient outlier detection method (as opposed to more strict ones). We will use the Interquartile Range (IQR) method ($Outliers \notin [q_{0.25} - k * IQR; q_{0.75} + k * IQR]$), because this method is based on percentiles, which is more robust than the Standard Deviation (SD) method, based on the mean and standard deviation (Hellerstein, 2013). However, the IQR method is also considered a more aggressive method (i.e. finding more outliers) than the SD method for similar scalars ($1.5IQR \approx 3SD$) (Li et al., 2021). Therefore, instead of the common value for k of 1.5, we will use value of 2 and 3, which will be less aggressive than 1.5 and more robust than the SD method. The outliers found using $k = 3$ are classified as "far" outliers, and the outliers found using $k = 2$ are classified as "close" outliers. We use this method to give the user an intuition what outliers are actually errors, and what errors are just unlikely, but real values. These two outlier value types can also separately be corrected, as will be explained below in the "Correction" section.

Outlier instances

Outlier instances are complete samples/rows in a dataset that do not fit the distribution. This essentially requires a combination of outliers on multiple attributes. It is again not necessarily the case that any outlier instance is a data error, but we can be more certain that these are errors compared to single value outliers since these require more than one outlier in one instance, which is rare.

The same reasons, why outlier values in a column are errors, apply for outlier instances but then magnified.

The detection technique used to detect outlier instances is Deepchecks' OutlierSampleDetection() function. This function uses the LoOp algorithm (Kriegel et al., 2009), which provides an outlier score between [0,1] for every instance. If this score is above a certain threshold, e.g. 0.8, the instance is classified as an outlier instance. By default, we have set this threshold to 0.8.

Correction

For the correction of outlier values in columns, the proposed correction method is to impute the far outlier values and retain the close outliers in the dataset. Li et al. (2021) showed in their study that cleaning outliers (in columns) using imputation could improve the performance of downstream models, but the impact was insignificant. They used the imputation techniques Mean, Median, Mode, and HoloClean (Rekatsinas et al., 2017) imputation. Thus, none of the imputation methods that we proposed for missing value imputation were tested for outliers in this study. Furthermore, they did not make a division between close and far outliers, but rather just imputed all outliers. Since, the correction technique is also just a recommendation, and the user can change it if he/she wants to, by default, the far outlier values will be imputed using the flowchart in Figure 1. The close outliers will be retained in the dataset, as they are less likely to be errors. Similar to the methodology for the missing value correction, also for this section the user is able to change the correction strategy per outlier type if he/she wants to. The available methods are the same as for the missing value correction step: MLP, CART, RF, KNN, Mode, "Remove", "Keep" and, Mean and Median (since we only detect outliers in numerical columns). The user can also, in a similar fashion, choose imputation per column, where all these options will also be available but then per column. Other options include "Remove rows", and "Keep all", similar to the missing values.

The correction technique for the outlier instances is to remove these instances from the dataset. Imputation for a complete instance does not make sense. Other options for the user include "Keep all", and "Select keep/remove" (both similar as before).

2.2.4 Cryptic attribute names

'Cryptic attribute names' refers to attribute names in datasets that are "cryptic". The most common type of cryptic attribute names are abbreviations. For example, the attribute name "Customer Name" is abbreviated to "C_Name". Other types of cryptic attribute names include names in other languages than English (E.g. Klant Naam (Dutch) instead of Customer Name) and misspellings (longtitude instead of longitude).

Cryptic attribute names should be changed to correct English words when uploading a dataset to OpenML, because one might not understand your cryptic attribute names. Furthermore, many cryptic attribute names are context-dependent. For example, C_Name in a customer dataset might relate to the name of a customer, but in a dataset of cars, it can relate to the name of a car brand. Another example

is the attribute name "rec_id", commonly related to the id of a record, but in the context of a cookbook dataset, it might relate to the id of a recipe.

The technique used to detect is based on Zhang et al. (2023)'s NameGuess algorithm. They studied the subject of cryptic attribute names and tried to create better attribute names using (regular) language models and large language models, which will be further discussed in the Section 2.3. Part of their algorithm is their CrypticIdentifier() function, which uses NLP to identify whether an attribute name is cryptic. We improved this function slightly by extending the vocabulary with an external English word vocabulary[5], since it had trouble classifying plurals and short English words like age or sex (which are quite common attribute names) as non-cryptic.

Correction

The recommended correction technique for the cryptic attribute names is to convert the attribute names to the suggestions of GPT-3.5. The detected cryptic attribute names are queried to OpenAI[6]'s GPT-3.5 model with the task of decrypting the attribute names. The model then returns for every cryptic attribute name an improved, non-cryptic name. We conducted an experiment to find the optimal query structure, which is described in Section 2.3. Again, the user has the ability to change the correction technique. He/she can choose to change the suggested names of GPT-3.5 and apply those, or choose "Keep original names" to retain the cryptic attribute names in the dataset, or choose "Select keep/change" to select per attribute which name he/she wants in the dataset.

2.2.5 Single value attributes

Single value attributes are attributes in a dataset where the column contains one unique value for every row. This can occur for example if a dataset is sliced on some attribute. For example, consider a dataset containing demographic information about The Netherlands, with a city column. If the dataset is sliced to only the city of Eindhoven, the resulting dataset will have only one unique value for the city column (i.e. Eindhoven).

Single value attributes do not have any predictive power for a model. Most of the time, if one would provide such an attribute as one of the inputs for a model, the model will just not use it. But just to be sure and to reduce dimensionality and improve training time, it is recommended to remove such attributes.

The technique used to detect this error is quite simple. For every column in the dataset, the number of unique values is counted using pd.nunique(). The single value attributes are the columns for which the nunique() function is smaller or equal to 1. We also count a column with zero values as a single value column, because then the column would consist of only NaN values and we would have to remove it anyway.

Correction

The technique to correct single value attributes used in the tool is to simply remove those attributes. These attributes do not add predictive power to models, and thus should be removed for dimensionality reduction and to avoid errors. The user is able to choose a different correction technique, choosing from "Keep all" and "Select keep/remove" (similar to before).

2.2.6 Mixed data types

Mixed data types are errors on the attribute level of a dataset, where an attribute/column contains both numeric and string values. An example would be a column named "height", which contains data on the height of people in the dataset. The numeric values are then for example 1.78, 1.92, 1.67 and the string values would be "1.81m", "1.55m" and "2.01m". Both the numeric and string values represent the height in meters, but the string values contain the unit of measurement "m" (meter) at the end of the string, after the value.

Mixed data types will almost always cause a downstream model to fail. If the mixed data column is assigned as a numerical column, the model will give an error for the samples that contain strings, because they do not fit the pattern. Categorical columns are often encoded into one-hot encodings or labels when

[5]https://www.kaggle.com/datasets/bwandowando/479k-english-words
[6]https://openai.com/

used as input for a model. In this case, the mixed data column will lead to inaccurate encodings, which will lead to suboptimal performance of the model and potentially unreliable results.

The technique used to detect mixed data types is using Deepchecks' MixedDataTypes() function. The function simply calculates, for the mixed data columns, the ratios of the number of string values and the numeric values. It takes a dataset as input and returns all column names in a dictionary including for the mixed data columns the percentage of string values, percentage of numeric values, examples of string values, and examples of numeric values.

Correction

Mixed data type attributes will be corrected using the output of the MixedDataTypes() function of Deepchecks. That function returns the ratios of the number of string values and the number of numeric values for all mixed data type columns. We will consider the majority data type (i.e. the type with the higher ratio) as the correct data type for the attribute. All values that are not in line with that data type (i.e. the string values if the ratio of numeric values is larger than the ratio of string values and the numeric values vice versa) will be considered "missing values". These "missing values" will then be (recommended to be) imputed using the framework we have defined before in Figure 1.

The user has the ability to change a lot in this section. First of all, the user can choose whether he/she wants to correct the minority data type in each column, the majority data type, or choose per column whether he/she wants to correct the minority or majority data type. Afterwards, the user can choose how to correct those data types. He/she can choose for imputation, with the possibility to change the imputation model per column (similar to missing values and outlier values). Other options include "Remome rows" and "Keep all" with similar meanings as before.

2.2.7 Incorrect labels

Incorrect labels are cases in a dataset where the target variable does not have the correct class/label corresponding to the instance. Examples often occur in image datasets, when the images are classified by a computer. But it can also happen in company datasets, where just by accident for example the wrong income class was assigned to an employee.

Incorrect labels can be detrimental for downstream models, as they have to train on faulty data, which will lead to worse performance and thus provide unreliable results (Bootkrajang and Kabán, 2014). Furthermore, label errors have been shown to increase the number of instances required to train effective predictive models. Lastly, they also increase model complexity and decrease model interpretability (Salekshahrezaee et al., 2021).

The technique used to detect label errors is Cleanlab[7]. Cleanlab uses confident learning to detect label errors which works impressively well even for high percentages of mislabeled samples (Northcutt et al., 2017). Cleanlab's python package has a function find_label_issues() which requires as input the target feature and the probabilities of any model predicting that target feature based on the other features in a dataset. The model we use to obtain the probabilities is RandomForest (RF) with cross-validation using a stratified K-fold split of $n = 5$. The function will return a table with a column "is_label_issue", which evaluates to True for the label errors and False for the correct labels. Those label errors will be our detected incorrect labels.

A disadvantage of Cleanlab is that it is very time-consuming to run. Therefore, we decided to, by default, only allow this detection tool for small datasets (n < 10.000). The user has the ability to enable the detection tool if he/she wants to, but there will be a clear remark on the time issue. Furthermore, this detection tool will only be deployed if the target feature is categorical, of course.

Correction

The technique to correct incorrect labels is implemented through Cleanlab. As mentioned above, the find_label_issues() function provides us the label errors using the "is_label_issue" column. Additionally, the table consists of the columns given_label and predicted_label, which will be used to change the labels from their erroneous value to the correct value (as predicted by Cleanlab). Again, following the HITL principles, the predicted labels will be recommended as new labels for the detected erroneous labels, but

[7] https://github.com/cleanlab/cleanlab

the user has the ability to choose other correction techniques. The user additionally can choose between the options "Remove rows", and "Keep all" (similar as before).

2.2.8 String mismatches

String mismatches are values in columns with string data where two (or more) values represent the same real-world entity while being encoded differently. For example, a dataset containing a column with names can have the values "Sander" and "sander", which refer to the same name, but are encoded differently (one with a leading uppercase, another fully lowercase). This example would be quite easily fixable, but it becomes more difficult if other variations are present, for example with plurals of words or words including punctuation tokens.

It is important for the performance of downstream models that take categorical features as input, that each category is actually distinct. One can imagine if the categories for an attribute "state" are "Noord-Brabant", "Noord Brabant", "noord-brabant" and "Utrecht" that the model will perform worse than if the categories were only "Noord-Brabant" and "Utrecht".

The technique used to detect these errors is the StringMismatch() function of Deepchecks. Their function transforms every value in a string column of a dataset to its base form (with only alphanumeric characters in lowercase) and then checks if two strings have the same base form. If two (or more) different strings have the same base form, they are detected as string mismatches by our tool.

Correction

String mismatches will also be corrected using Deepchecks' StringMismatch() function. We will recommend to the user to convert all variations (different strings with the same base form) of a string mismatch to their corresponding base form. Additionally, the user can choose from the options "Remove rows", "Keep all", and "Convert to mode". The first two options have been explained before. The latter option converts all variations of a string mismatch to the most frequently occurring variation in that string mismatch.

2.3 Cryptic attribute names experiment

As mentioned before in Section 2.2.4, Zhang et al. (2023) compared different LMs and LLMs on the task of creating better attribute names. As a query, they used an example (e.g. "C_Name" to "Customer Name), then provided the attribute names for which they wanted to generate better names, and finally added some content of those attributes. We will experiment with whether we can improve this procedure by additionally adding the title and/or the description of the dataset to the query.

2.3.1 Approach

For this experiment, we adopted a similar approach as Zhang et al. (2023) did in their paper. First, 20 datasets with 10-30 attributes were collected from OpenML. We randomly sampled 15 datasets with 10-20 attributes and 5 datasets with 20-30 attributes. Datasets with less than 5 non-cryptic attributes were replaced, so we have at least $5 * 20 = 100$ sample attribute names for the experiment. The details of all 20 datasets can be found in Table 5. Then, Zhang et al. (2023)'s CrypticIdentifier() function is used to find the attribute names in a dataset that are **not** cryptic. Then, their CrypticGenerator() function is used to generate cryptic/abbreviated names for those non-cryptic attribute names. This approach is used so we would have a ground truth value (the original non-cryptic attribute name). We insert these generated cryptic attribute names as input in a query to GPT-3.5 with the task of generating better names for those attributes. We then compare the output of GPT-3.5 to the ground truth values of those attribute names. We calculate the performance using the measures Exact Match accuracy and BERTScore F1.

Exact Match (EM). The EM accuracy is calculated as the number of predicted attribute names that are identical to the ground truth attribute name divided by the total number of predicted attribute names. The ground truth and predicted attribute names are first normalized, which means removing articles and punctuation and converting the names to lowercase.

BERTScore F1. The BERTScore F1 calculates a similarity score for each token in the predicted name with that in the ground truth name using pre-trained contextual embeddings.

2.3.2 Experiments

We will first experiment with the number of instances we should provide as content for the query. We will test $n = 0, 1, 3, 5, 10$, where n is the amount of randomly sampled rows. We will evaluate the performance relative to the number of tokens used (since GPT-3.5 costs \$0.003 per 1.000 tokens). We do this to find the optimal n as content for the query, which will be the sweet spot between maximum performance and minimum costs. Using this optimal n, we will perform the second experiment in which we test whether performance improves (relative to the extra tokens used) if we add the title, description, or title and description to the query. This experiment will be evaluated in a similar fashion, where we consider the performance relative to the amount of tokens used.

2.4 Quality improvement

We would like to evaluate whether the overall quality of datasets on OpenML will be improved using our tool. In order to test this, we sample 20 datasets from OpenML from which the details can be found in Table 4 in the Appendix. The tasks related to the datasets consist of classification and regression tasks. We will use 5 different downstream models for the different tasks (Random Forest, Linear/Logistic Regression, K-Nearest Neighbors, Decision Tree, and Gradient Boosting) combined with a 5-fold cross-validation, which is stratified for the classification task. Then, we evaluate the performance of the "dirty" (current OpenML datasets) and "clean" datasets and compare both. The performance metrics used for classification tasks are the Accuracy and the F1 score. The metrics used for regression tasks are the Mean Absolute Error (MAE) and the Root Mean Squared Error (RMSE). We will compare the amount of times a dirty dataset performed the best compared to the amount of times a clean dataset performed the best.

2.5 User testing

The final part of the methodology describes the process of user testing the tool. It is important to get feedback on the tool to identify pitfalls, in order to correct those. Since the tool should be usable for any person uploading a dataset to OpenML (whether they have data science knowledge or not), we select a diverse group of people to test the tool. 10 people will be selected to provide feedback of which 5 have a data science background and 5 have not.

3 Related work

Several research papers have already created data cleaning tools. In this section, we will provide a summary of the most prominent papers on this subject and their findings. Afterwards, we will compare these tools to the OpenML Data Cleaner to argue why this tool is superior.

3.1 CleanML (Li et al., 2021)

CleanML investigates 5 different, common data errors (missing values, duplicates, outliers, inconsistencies, and mislabels) and how effective different cleaning methods are in correcting those errors. They used a combination of simple, well-known cleaning methods, such as mean/mode imputation, but also more advanced techniques like HoloClean (Rekatsinas et al., 2017) and ZeroER (Wu et al., 2020). They evaluated using hypothesis testing whether the impact of certain cleaning methods is significant or not. The results showed that for some cleaning methods for certain errors, the model accuracy did improve significantly, but for others the improvement was negligible or the model performed even worse.

3.2 ActiveClean (Krishnan et al., 2016)

ActiveClean is an interactive data cleaning tool, which iteratively updates the model it uses to correct errors as the data is being cleaned. Krishnan et al. (2016) highlight that modern data analysis pipelines frequently encounter issues with missing, incorrect, or inconsistent data, which can severely impact the performance of ML models. ActiveClean addresses these issues by using the user to clean the dataset in small batches, while continuously updating the model. For this cleaning process, they have implemented a visual interface, which makes ActiveClean accessible for use in various applications such as video classification or topic modeling.

In the tool, they use importance weighting and dirty data detection, which prioritizes the cleaning of certain data values, which are more likely to have a significant impact on the model's accuracy. In this way, they can create a very accurate model with only limited input data. This approach is particularly effective when data errors are not uniformly distributed, allowing the tool to dynamically adapt to the specific needs of the dataset being analyzed.

3.3 Raha (Mahdavi et al., 2019)

Raha does not focus on cleaning data errors in datasets itself, but rather on error detection and correction in relational databases. Raha automatically detects and corrects several relational database errors, like rule violations and inconsistencies. The tool uses statistical methods to detect and correct errors as well as ML techniques. The main advantage of Raha is its ability to use user-defined constraints and rules in the cleaning process. Raha can achieve high precision scores in both error detection and correction due to implementing this domain-specific knowledge in the tool.

3.4 ED2 (Neutatz et al., 2019)

ED2 is pretty similar to ActiveClean as it also uses active learning for error detection. In a similar fashion, ED2 selects the most important data values which the user needs to label. These are values that the model is the least certain about, and after the user labels them, the model uses them to learn and improve performance. Their tool achieves better detection accuracy with fewer user-provided labels compared to complex models that rely on data augmentation.

3.5 Comparison to OpenML Data Cleaner

Finally, we will discuss the results of the aforementioned papers compared to the results of this paper and the OpenML Data Cleaner tool.

3.5.1 Addressed errors

The first advantage the OpenML Data Cleaner has in comparison to the aforementioned tools is the amount of errors it addresses. It is hard to compare to ActiveClean and ED2 as they do not have specific algorithms for each error, but they rather train a model that detects whether any value in the dataset is erroneous or not using queries to the user. But, in general, this will not (or not always) address whole attribute or whole instance issues, such as outlier instances, duplicate instances, duplicate attributes and single value attributes. In comparison to CleanML, our tool does not address inconsistencies directly, although it is quite similar to string mismatches and disguised missing values. On the other hand, their tool does not address cryptic attribute names, single value attributes and mixed data type attributes. Raha is also hard to compare, but they also lack some errors.

3.5.2 Knowledge required

The second advantage of the OpenML Data Cleaner in comparison to the aforementioned tools is that the user is not required to have extensive knowledge on data errors and about the dataset itself. In ActiveClean and ED2, the user gets queried values from the dataset and has to validate whether they are actual true values or errors. This requires the user to have good knowledge on both data errors as well as the dataset itself. The OpenML Data Cleaner, however, provides the user with the opportunity to validate and add errors him-/herself, but does not require them to do so. If they do not have enough required knowledge, they can simply apply the default recommendations of the tool.

4 Tool deployment

4.1 Introduction

The tool is completely implemented in Dash. Dash makes it easy to build extensive dashboards in Python language. The tool will be structured into an error detection section and an error correction section.

In the error detection section, the detection tools for each error type described in Section 2.2 will be implemented. Each error type will get its own subsection in the tool. The order of the sections is based on the importance of each error type. More well-known and detrimental error types are addressed first,

less important errors are addressed later. In each subsection, the user is shown information about the specific error type and the actual errors detected by the tool for that specific error type. The user is then (depending on the error type) able to deselect instances/attributes/values that he/she does not consider errors (false positives). Furthermore, the user can add instances/attributes/values that the tool did not detect but are actual errors (false negatives).

After all steps in the error detection section have been completed, and the user has submitted the detected errors, the error correction section will be available. There, all the error correction tools (also described in Section 2.2) will be implemented. Again, each error type gets its own subsection for correction, in the same order as the error detection section. Each subsection in the error correction section consists of two parts: recommendations and corrections. In the recommendations part, the tool displays the recommended correction technique (as described in Section 2.2). Using this information, the user can then select the correction technique him-/herself. After submitting the correction technique, the corrections part becomes available. In this part, the dataset is shown with the selected correction technique applied to it. The dataset is directly updated with the corrections, so for every next error type subsection, the updated dataset is corrected instead of the original dataset. This makes it possible that certain errors (of the last error types) are already corrected before the user has corrected them directly in the corresponding error type section. For example, if instance 22 in the dataset is a duplicate instance and contains an outlier value in column "A", the instance might already be removed in the "Duplicate instances" section, thus the outlier value does not exist anymore in the "Outlier values" section.

After applying correction techniques to all error types, the user is shown his/her final cleaned dataset and he/she can "update" the dataset on his/her personal OpenML dataset page with the cleaned dataset (and download the cleaned dataset).

4.2 Error detection

4.2.1 Overview

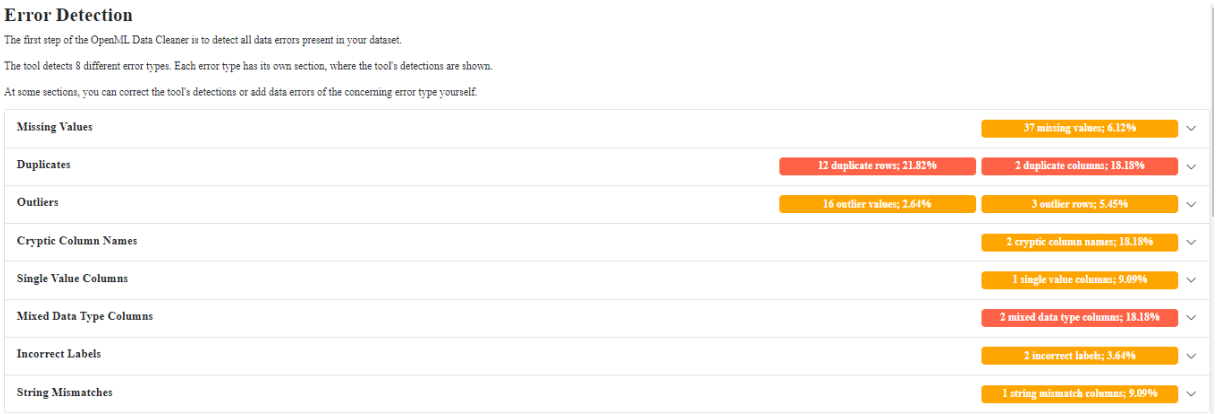


Figure 2: Overview of the error detection section of the tool

When the user accesses the data cleaning tool for a specific dataset, the tool first detects all errors using several algorithms for each specific error type. The landing page of the tool can be seen in Figure 2. The 8 different error types all have a separate section. The header of each section contains the name of the error type and a "badge" (or sometimes 2 badges). This badge displays the number of errors for that specific error type and the percentage of those errors compared to the whole dataset. For example, the error type "Missing Values" contains 27 (disguised) missing values, which is 4.46% of all values in the dataset in the example of Figure 2. The colors of the badges display the seriousness of the errors: green means no errors, orange means "a couple" errors and red means many errors. What the difference between "a couple" and "many" errors is, differs per error type.

4.2.2 Missing values

(Disguised) Missing Values

Missing values are values in your dataset that do not contain a value. This is often represented as NaN (not a number), None, NA (not applicable) or just by an empty cell.

Disguised missing values are also missing values but these values are not easily distinguishable from regular correct values. For example, the value 12345678 in a column named "phone number", is probably not a legitimate value. As one can understand it can be difficult to detect such disguised missing values, therefore we ask you kindly to correct the incorrect detections of our algorithm.

Disguised missing value | Regular missing value

index	age	workclass	fnlwgt	education	gender	native-country	income	high_label_correlation	redundant SV column	duplicate column	mixed_data_types
0	96	Private	121321	?	Male	United-States	<=50K	average salary	this_is_the_same	96	3
1	36	Private	73023	HS-grad	Male	$$$$$	<=50K	average salary	this_is_the_same	36	MD2
4	44	VARiation!	282192	?	Male	United-States	<=50K	CHANGED VALUE	this_is_the_same	44	3
5	44	Private	176063	Bachelors	Male	$$$$$	<=50K	average salary	this_is_the_same	44	2
7	88	Self-emp-not-inc	187097	Prof-school	Male	$$$$$	<=50K	average salary	this_is_the_same	88	MD3
8	67	Private	197816	Some-college	nan	United-States	<=50K	average salary	this_is_the_same	67	1
10	26	Self-emp-not-inc	31143	HS-grad	Male	@#$@	<=50K	average salary	this_is_the_same	26	MD3
12	49	Self-emp-not-inc	208872	HS-grad	nan	United-States	<=50K	average salary	this_is_the_same	49	2
19	38	Private	238721	Assoc-acdm	nan	United-States	<=50K	average salary	this_is_the_same	38	1
21	47	Variation@	282830	Assoc-acdm	Male	United-States	>50K	high salary	this_is_the_same	47	1
22	42	Self-emp-not-inc	103759	HS-grad	nan	###	<=50K	average salary	this_is_the_same	42	MD1
23	26	vaRiAtIoN	33604	HS-grad	nan	United-States	<=50K	average salary	this_is_the_same	26	1
24	29	Private	59792	?	Male	Taiwan	<=50K	average salary	this_is_the_same	29	1
30	37	Self-emp-inc	126675	?	nan	United-States	<=50K	average salary	this_is_the_same	37	MD1
31	32	Variation@	133530	Masters	Female	United-States	<=50K	average salary	this_is_the_same	32	MD2
32	61	Self-emp-not-inc	181033	HS-grad	Male	@#$@	<=50K	average salary	this_is_the_same	61	3
34	47	Variation@	282830	Assoc-acdm	Male	United-States	>50K	high salary	this_is_the_same	47	1
37	47	Private	24723	10th	Female	$$$$$	<=50K	average salary	this_is_the_same	47	MD3
38	143	variation	162108	0	nan	United-States	<=50K	average salary	this_is_the_same	143	1
39	36	Private	250238	1st-4th	Female	###	<=50K	average salary	this_is_the_same	36	1

Figure 3: Missing values detected by the tool

In Figure 3, you can see the first part of the missing values section. In this first part, the user can see the missing values (red) and disguised missing values (orange) in the dataset. Again, the colors display the seriousness: red values are definitely missing values, orange values not necessarily; they are potentially disguised missing values.

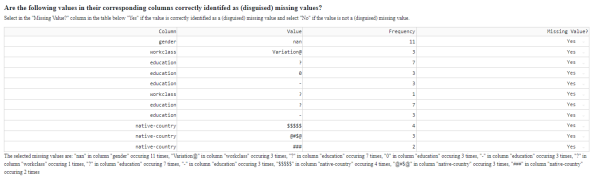


Figure 4: Section where the user can deselect (disguised) missing values wrongly detected by the tool

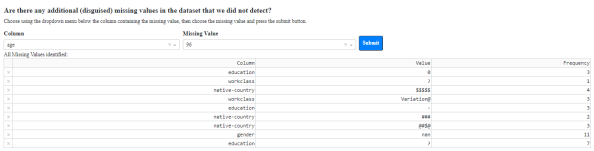


Figure 5: Section where the user can add undetected (disguised) missing values

The next step of the missing values section is to let the user validate the missing values detected by the tool and then allow the user to add (disguised) missing values that were not detected by the tool. This step can be seen in Figures 4 and 5. Following the HITL principles, we want to make use of the user's expertise of the dataset, which is why these sections are included.

4.2.3 Duplicates

Duplicate instances

Duplicate Rows

These are the rows/rows in your dataset that contain identical values across all columns (except for the index column). In the table below, we have displayed these errors by outlining pairs of duplicate rows with the same color.

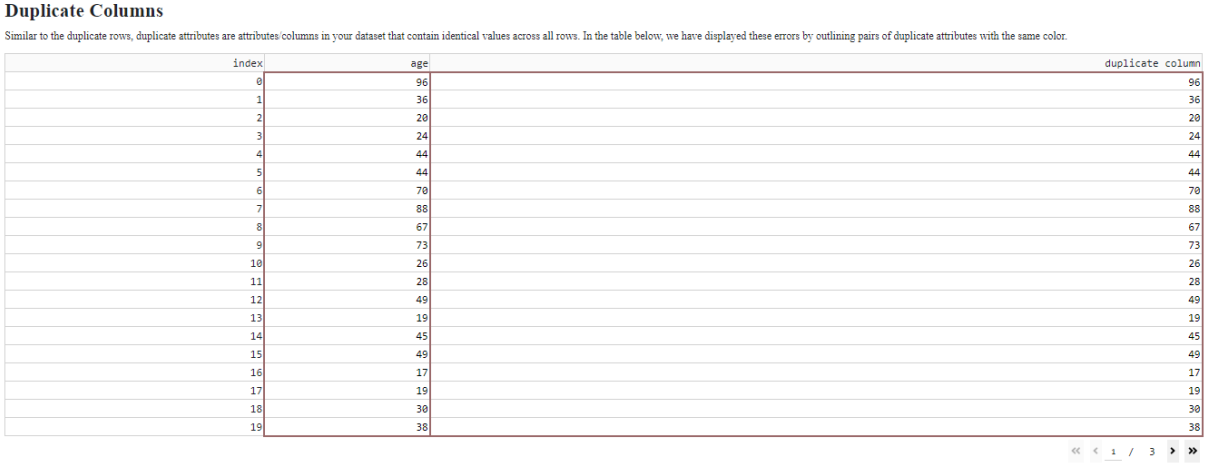
index	age	workclass	fnlwgt	education	gender	native-country	income	high_label_correlation	redundant SV column	duplicate column	mixed_data_types
13	19	Private	100009	Some-college	Female	United-States	<=50K	average salary	this_is_the_same	19	2
14	45	Private	55720	Bachelors	Male	United-States	>50K	high salary	this_is_the_same	45	3
20	45	Private	55720	Bachelors	Male	United-States	>50K	high salary	this_is_the_same	45	3
21	47	Variation@	282830	Assoc-acdm	Male	United-States	>50K	high salary	this_is_the_same	47	1
23	26	vaRiAtIoN	33604	HS-grad	nan	United-States	<=50K	average salary	this_is_the_same	26	1
24	29	Private	59792	?	Male	Taiwan	<=50K	average salary	this_is_the_same	29	1
25	19	Private	100009	Some-college	Female	United-States	<=50K	average salary	this_is_the_same	19	2
34	47	Variation@	282830	Assoc-acdm	Male	United-States	>50K	high salary	this_is_the_same	47	1
35	41	Local-gov	242586	Some-college	Male	United-States	<=50K	average salary	this_is_the_same	41	MD1
43	41	Local-gov	242586	Some-college	Male	United-States	<=50K	average salary	this_is_the_same	41	MD1
46	26	vaRiAtIoN	33604	HS-grad	nan	United-States	<=50K	average salary	this_is_the_same	26	1
47	29	Private	59792	?	Male	Taiwan	<=50K	average salary	this_is_the_same	29	1

Figure 6: Duplicate instances detected by the tool

Duplicate instances in the dataset are made visible using the same colored borders for duplicate groups in

a table, as can be seen in Figure 6. This error type does not allow the user to validate or add errors, since all instances detected are for sure duplicates, and all instances not detected are for sure not duplicates.

Duplicate attributes



index	age	duplicate column
0	96	96
1	36	36
2	20	20
3	24	24
4	44	44
5	44	44
6	70	70
7	88	88
8	67	67
9	73	73
10	26	26
11	28	28
12	49	49
13	19	19
14	45	45
15	49	49
16	17	17
17	19	19
18	30	30
19	38	38

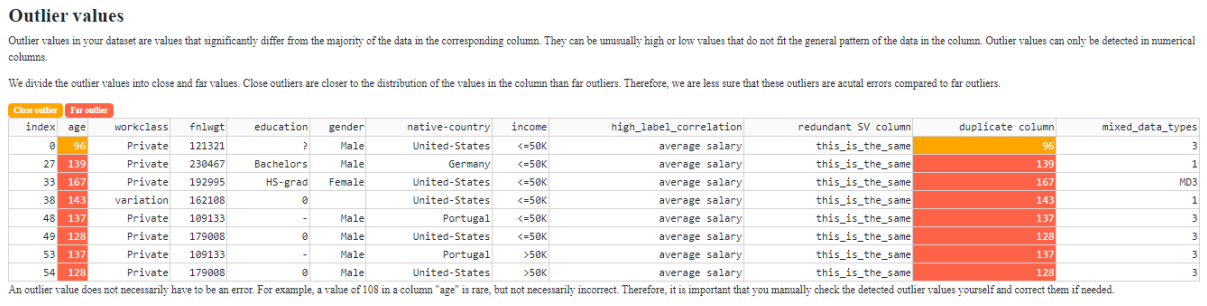
« ‹ 1 / 3 › »

Figure 7: Duplicate attributes detected by the tool

In Figure 7, you can see that also the duplicate attributes in the dataset are displayed by creating same-colored borders for the duplicate groups. Also for the duplicate attributes the user cannot validate or add any attributes for the same reason as for the duplicate instances.

4.2.4 Outliers

Outlier values



index	age	workclass	fnlwgt	education	gender	native-country	income	high_label_correlation	redundant SV column	duplicate column	mixed_data_types
0	96	Private	121321	?	Male	United-States	<=50K	average salary	this_is_the_same	96	3
27	139	Private	230467	Bachelors	Male	Germany	<=50K	average salary	this_is_the_same	139	1
33	167	Private	192995	HS-grad	Female	United-States	<=50K	average salary	this_is_the_same	167	MD3
38	143	variation	162108	0		United-States	<=50K	average salary	this_is_the_same	143	1
48	137	Private	109133	-	Male	Portugal	<=50K	average salary	this_is_the_same	137	3
49	128	Private	179008	0	Male	United-States	<=50K	average salary	this_is_the_same	128	3
53	137	Private	109133	-	Male	Portugal	>50K	average salary	this_is_the_same	137	3
54	128	Private	179008	0	Male	United-States	>50K	average salary	this_is_the_same	128	3

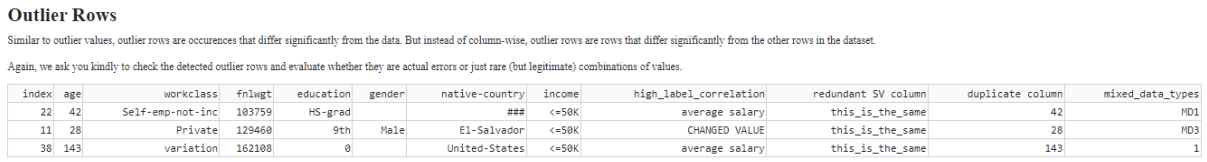
An outlier value does not necessarily have to be an error. For example, a value of 108 in a column "age" is rare, but not necessarily incorrect. Therefore, it is important that you manually check the detected outlier values yourself and correct them if needed.

Figure 8: Outlier values detected by the tool

The outlier values detected by the tool are displayed in a similar way as to the missing values as can be seen in Figure 8. In this case, red values are "far" outliers and orange values are "close" outliers. This is based on the "k" parameter in the Inter-Quartile Range (IQR) method to detect outlier values, as described in Section 2.2. The close outliers are more than $k = 2$ IQRs away from the 25th and 75th quartile, while the far outliers are $k = 3$ IQRs away. Thus, the red values are more likely to be erroneous outliers.

For the outlier values, there is a similar section as to the missing values (Figure 5), where the user can validate the detected outlier values and can add undetected outlier values him-/herself.

Outlier instances



index	age	workclass	fnlwgt	education	gender	native-country	income	high_label_correlation	redundant SV column	duplicate column	mixed_data_types
22	42	Self-emp-not-inc	103759	HS-grad		###	<=50K	average salary	this_is_the_same	42	MD1
11	28	Private	129460	9th	Male	El-Salvador	<=50K	CHANGED VALUE	this_is_the_same	28	MD3
38	143	variation	162108	0		United-States	<=50K	average salary	this_is_the_same	143	1

Figure 9: Outlier instances detected by the tool

The outlier instances are displayed in a table, as you can see in Figure 9. The instances are ordered on the probability of being an outlier instance. In Figure 9, instance 22 is most likely to be an outlier instance, then instance 11, then instance 38. In this example, we have set the probability threshold to 0.5 to show the working of the tool, but as described in Section 2.2, by default this threshold is set to 0.8.

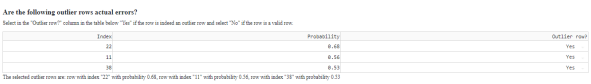


Figure 10: Section where the user can validate the detected outlier instances

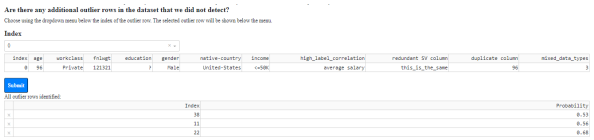


Figure 11: Section where the user can add undetected outlier instances

Similar to the missing values and outlier values sections, also the outlier instances section has a validate- and add-section. It looks a little bit different as to the missing values, as can be seen in Figures 10 and 11, which is why we included it. In the validation section, the user gets to see the probabilities of each instance being an outlier, to help inform his/her choice to deselect the instance or not. For the add-section, the user can select an index from a dropdown menu. Then, the instance is shown and the user can submit it as an outlier instance.

4.2.5 Cryptic attribute names

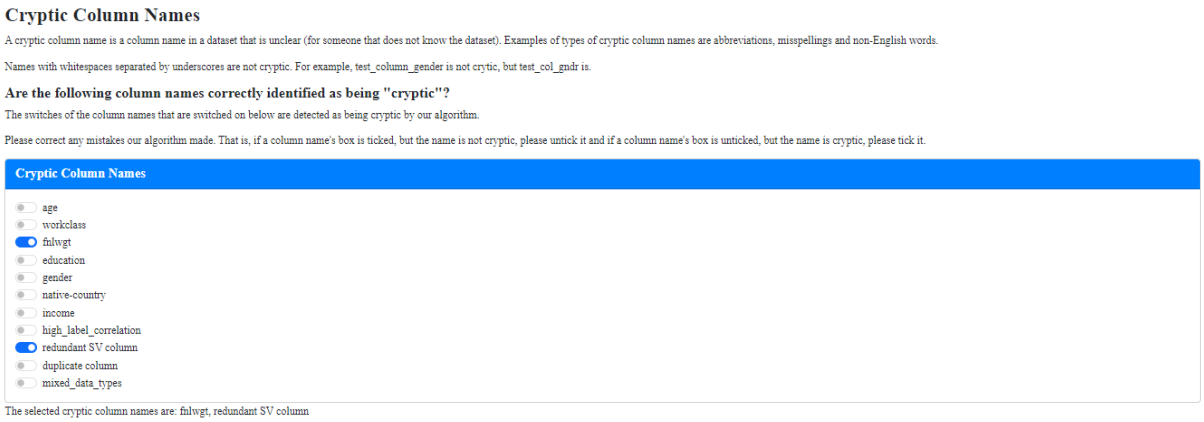


Figure 12: Cryptic attribute names detected by the tool

In Figure 12, the "Cryptic attribute names" section is shown. The column names of the dataset are displayed in a checklist, where the boxes of the cryptic column names (as detected by the algorithm) are ticked. The user can easily tick or untick boxes and in this way validate the detections of the tool and add undetected cryptic columns.

4.2.6 Single value attributes

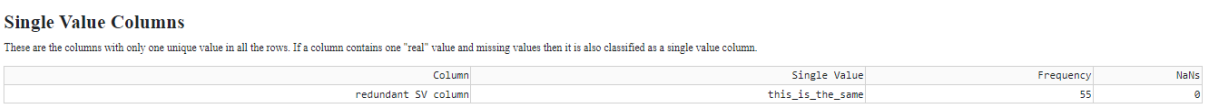


Figure 13: Single value attributes detected by the tool

The detected single value columns are displayed in a table with the column name, the single value, the frequency of the single value (can differ due to NaNs), and the number of NaNs in the column as can be seen in Figure 13. Similar to the duplicate instances and attributes, there is nothing to validate or add for the user.

4.2.7 Mixed data type attributes

Mixed Data Type Columns

These are columns that contain both numeric values as well as strings (or text) values. For example, a column named "age" containing numbers (e.g. 21, 43, 76) and strings (e.g. twenty-one, forty-three, seventy-six).

Column	Strings (%)	Numbers (%)	String Examples	Number Examples
education	0.9454545454545454	0.05454545454545454	{'Masters', 'HS-grad', 'Bachelors'}	{'0'}
mixed_data_types	0.2727272727272727	0.7272727272727273	{'MD2', 'MD3', 'MD1'}	{'3', '1', '2'}

Figure 14: Mixed data type attributes detected by the tool

In Figure 14 can be seen that the mixed data type attributes are displayed in a table that consists of the column name, the percentage of strings and numbers in that column, and examples of those strings and numbers. Again, there is nothing to validate or add for the user.

4.2.8 Incorrect labels

Incorrect Labels

Incorrect labels are rows that got assigned the wrong label/value of the target column. For example, a dataset with target column "country" and in the dataset there is a column named "capital". If the value in the target column for an row with capital "Amsterdam" is "Germany", then we have an incorrect label (since it should be "The Netherlands").

Previous label | Predicted label

index	age	workclass	fnlwgt	education	gender	native-country	high_label_correlation	redundant SV column	duplicate column	mixed_data_types	income (previous)	income (predicted)
45	26	Private	102476	Bachelors	Male	United-States	average salary	this_is_the_same	26	3	<=50K	>50K
50	26	Private	102476	Bachelors	Male	United-States	average salary	this_is_the_same	26	3	>50K	<=50K

Figure 15: Incorrect labels detected by the tool

The incorrect labels are shown in a table of the dataset where the target column is split into the previous target (as present in the dataset) and the predicted target (as predicted by Cleanlab). The previous labels are displayed in red and the predicted labels are displayed in green as can be seen in Figure 15.

Figure 16: Section where the user can validate the detected incorrect labels

Figure 17: Section where the user can add undetected incorrect labels

Again, there is the possibility for the user to validate and add incorrect labels, as can be seen in Figures 16 and 17. In the validation section, the user can in the 'Predicted label' column also select a different correct label, if the tool did correctly identify the incorrect label but did not predict the correct label. In the add-section, the user can select an index of an instance in which an incorrect label occurs and then select the correct label from the dropdown menu.

4.2.9 String mismatches

String Mismatches

String mismatches are cases where two different text/string values in a column refer to the same entity. For example, in a column named country "Belgium" and "belgium" refer to the same country, but the values are not identical (one starts with a capital and the other not).

In the tables below, we display the string mismatches we found in your dataset. The base of these string mismatches can be found in the header above the table.

String mismatch in column "workclass" with base variant "variation"

variant	count	percent
VARiation!	1	0.01818181818181818
variation	2	0.03636363636363636
vaRiAtIoN	3	0.05454545454545454
Variation@	3	0.05454545454545454
VARIATION	1	0.01818181818181818

Figure 18: String mismatches detected by the tool

In Figure 18, the first part of the string mismatches section is shown. Each detected string mismatch consists of a column name, base variant, and multiple variation values. Each column name-base variant combination gets its own table in which the variation values are shown, how often they occur, and how much percent of the column they claim.

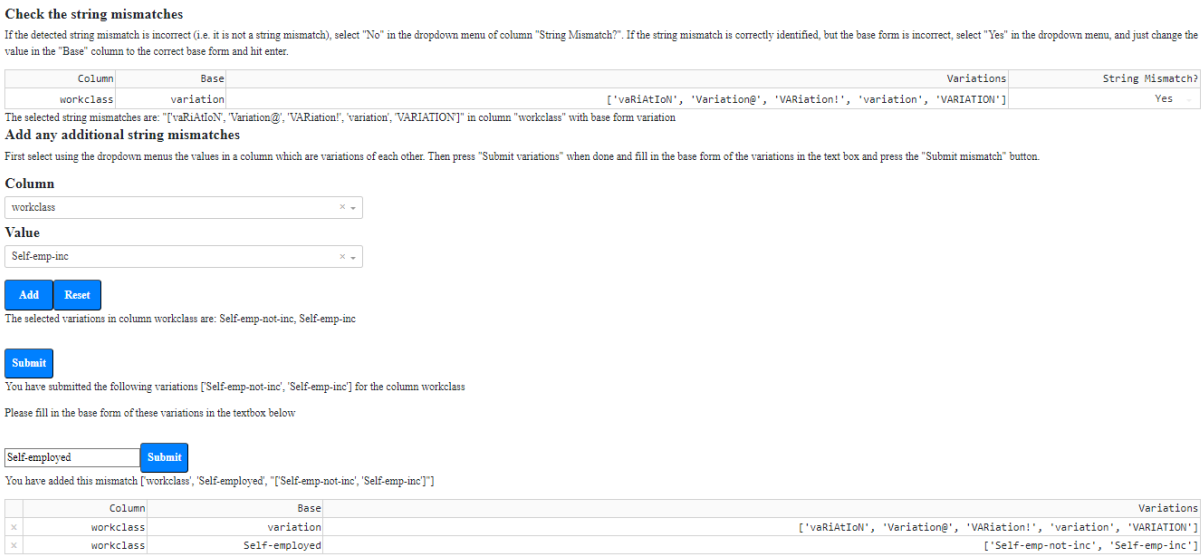


Figure 19: Section in which detected string mismatches are validated and additional string mismatches can be added by the user

In the second part, the user can validate the detected string mismatches by the tool and add additional string mismatches (Figure 19). The user can add value variations to a string mismatch using dropdown menus and then add the base variant of those variations using the input section.

It is not possible for a value variation to be part of multiple string mismatches, as one is then unable to determine what base should be assigned to that value. Therefore, the section consists of a pop-up subsection, which is enabled whenever the user tries to add a value variation that is already present in a different string mismatch. This subsection is displayed in Figure 20. The user has two options: choose a new value variation to add to the current string mismatch or delete the value variation from the already existing string mismatch and add it to the current.

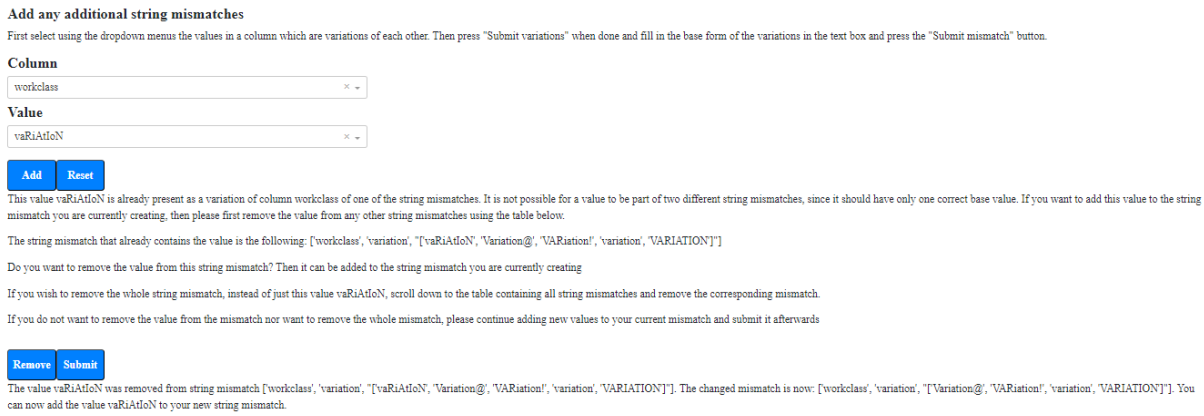


Figure 20: Subsection which opens up whenever the user tries to add a value variation that is already present in a different string mismatch

4.2.10 Submit errors

Finally, to conclude the error detection step of the tool, the user has to submit the errors by pressing the "Submit" button and then pressing "OK" in the confirmation window that pops up. After pressing "OK" the error correction section will be generated using the submitted errors.

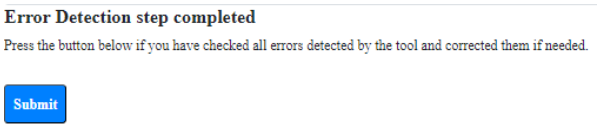


Figure 21: Final section of the error detection step where the user submits the detected errors

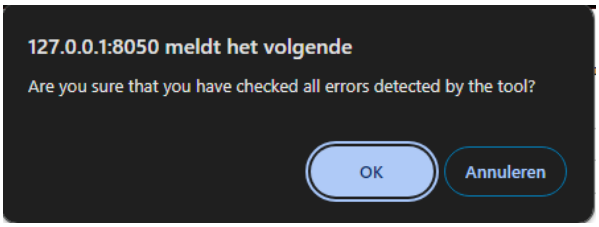


Figure 22: Confirmation window to confirm that the user wants to submit the detected errors

4.3 Error correction

4.3.1 Overview

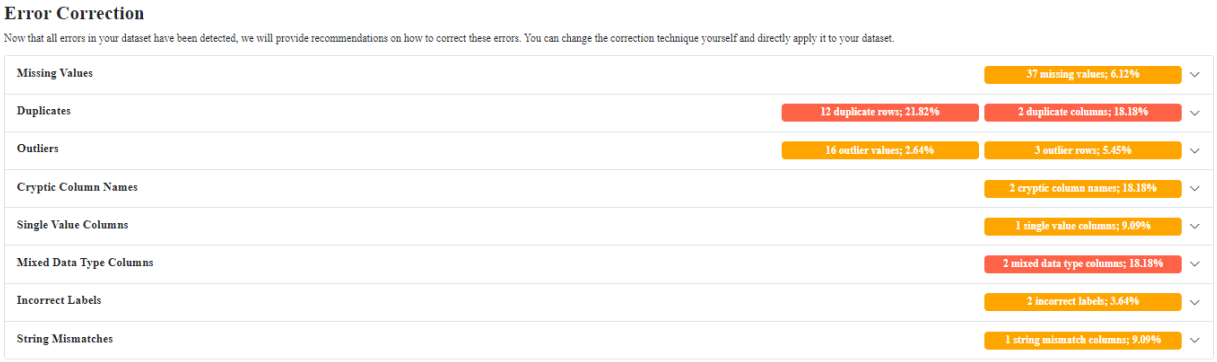


Figure 23: Overview of the error correction section of the tool

After the user has submitted the detected errors, a new section in the tool appears: the error correction step. In this section, the user can correct the detected errors in the dataset. For each error type, again a badge is placed in the header. The values are now updated to the number of submitted (i.e. the numbers change depending on the validations and additions of the user). Each error type section consists of two parts: recommendations and corrections. First, the recommended correction for a specific error type is displayed to the user, after which he/she can select the correction technique. Then, the correction technique is applied on the dataset and in the 'corrections' part, the updated dataset is shown.

4.3.2 Missing values

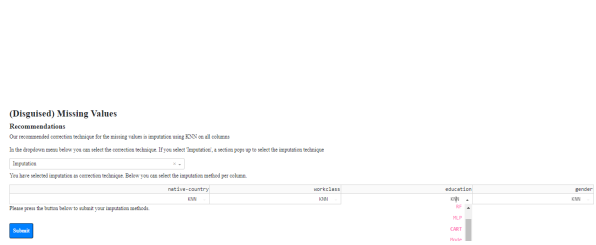


Figure 24: Recommendations section for the missing values

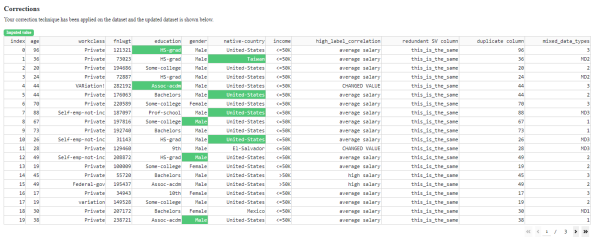


Figure 25: Corrections section for the missing values

In the 'Recommendations' part of the missing values section, first the recommended correction technique (imputation) is put forward. Then, the user can select from the dropdown menu any of the choices mentioned in Section 2.2.1: Imputation, "Remove rows", or "Keep all". In Figure 24, Imputation was chosen, after which a table appears where the user can select per column the imputation technique he/she wants to apply. The possible imputation techniques have also been mentioned in Section 2.2.1.

After submitting the correction technique in the 'Recommendations' part, the 'Corrections' part becomes visible (Figure 25). In this section, the dataset is displayed with the correction technique applied.

Imputed values are displayed in green cells, removed instances are displayed using red lines, and retained values are displayed in blue cells.

4.3.3 Duplicates

Duplicate instances

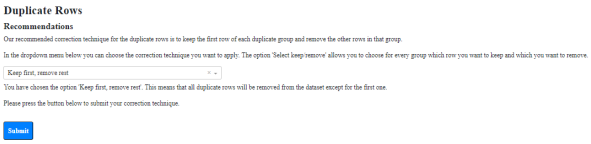


Figure 26: Recommendations section for the duplicate instances

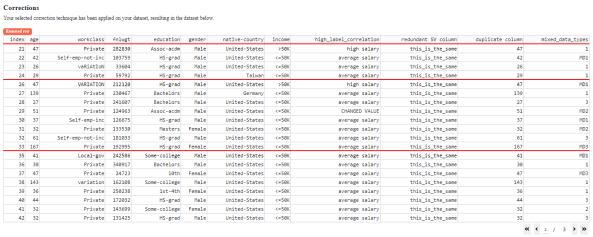


Figure 27: Corrections section for the duplicate instances

In the Recommendations part of the duplicate instances section, the user is recommended to keep the first instance of each duplicate group, and remove the other instances in that group from the dataset. The other options the user can choose from are "Remove all", "Keep all", and "Select keep/remove". In Figure 26, we chose for the recommended correction technique.

After submitting the correction technique, the Corrections part displays the updated dataset, where the removed instances are shown using red lines as can be seen in Figure 27.

Duplicate attributes

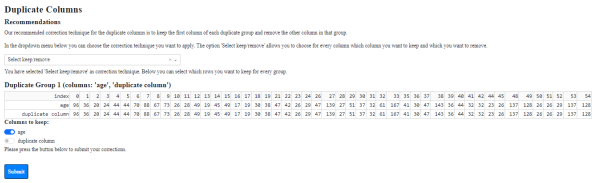


Figure 28: Recommendations section for the duplicate attributes

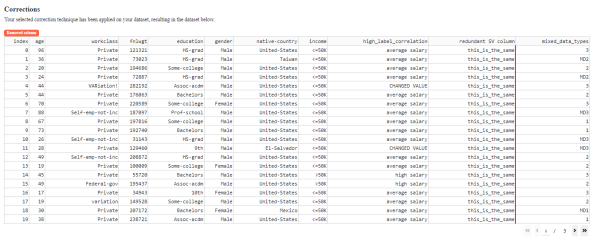


Figure 29: Corrections section for the duplicate attributes

The recommended correction technique is to retain the first attribute from each duplicate group, and remove the other attributes in that duplicate group from the dataset. The other correction options for the user are again "Remove all", "Keep all", and "Select keep/remove". In Figure 28, we chose for "Select keep/remove" to display how this section looks like. In this section, the user is able to select for each duplicate group, the attributes that he/she wants to keep in the dataset. The attributes that are not selected to be kept, will be removed from the dataset.

After submitting, Figure 29 displays the updated dataset, where the removed attributes are marked with a red line. Note that the dataset does not contain any (disguised) missing values anymore, since those were corrected in a previous step.

4.3.4 Outliers

Outlier values

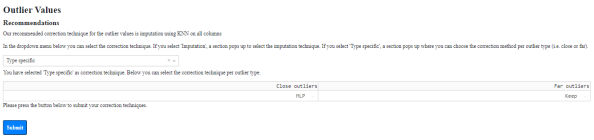


Figure 30: Recommendations section for the outlier values

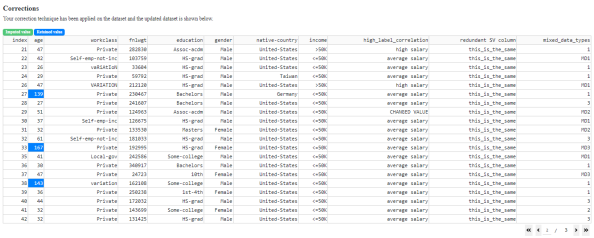


Figure 31: Corrections section for the outlier values

The recommended correction technique is imputation of the outlier values. The user is able to select a different correction technique including "Remove rows", "Keep all", and "Type specific". The latter is selected in Figure 30. It allows the user to select a specific imputation technique per outlier type (i.e. close and far outliers). In the example, the close outliers were imputed using MLP and the far outliers were retained in the dataset.

The Corrections section now shows blue cells for the retained far outliers and green cells for the imputed close outliers, as can be seen in Figure 31.

Outlier instances

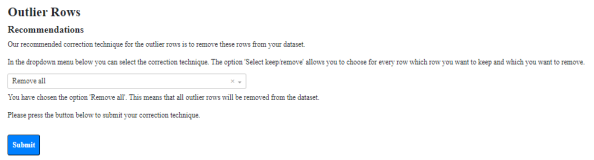


Figure 32: Recommendations section for the outlier instances

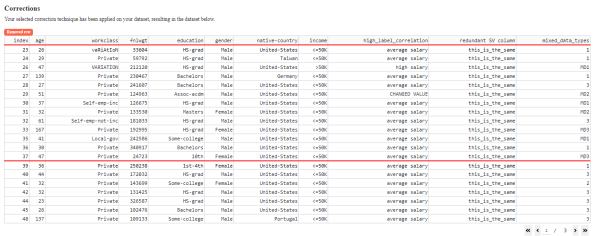


Figure 33: Corrections section for the outlier instances

The Recommendations part for the outlier instances suggests to the user to remove all outlier instances from the dataset. The user is able to select a different correction technique, choosing from "Keep all" and "Select keep/remove". In Figure 32, the recommended technique was chosen.

The updated dataset is shown in the Corrections part after submitting the correction technique. As can be seen in Figure 33, the removed instances are displayed with a red line.

4.3.5 Cryptic attribute names

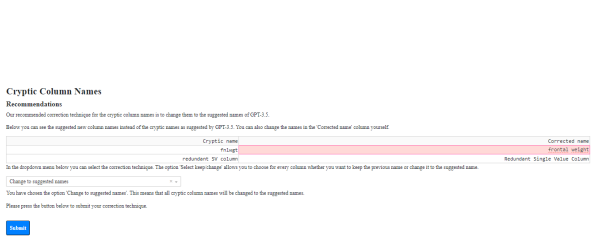


Figure 34: Recommendations section for the cryptic attribute names

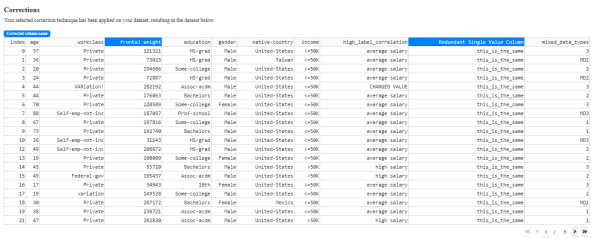


Figure 35: Corrections section for the cryptic attribute names

The Recommendations part for the cryptic attribute names looks a little different than for the other error types. The recommended correction technique is to convert the original, "cryptic" attribute names to the names suggested by OpenAI's GPT-3.5. It is important that the user first knows what the suggestions are before applying them to the dataset. In the table, visible in Figure 34, the user is also able to change the suggestions made by GPT-3.5 to a name that suits better. Then, when choosing "Change to suggested

names", the name put in by the user is used instead of the original suggestion of GPT-3.5. Other options for the user include "Keep original names" and "Select keep/change", which are similar as to previous sections.

In the Corrections part, the cryptic column names have been changed according to the correction technique chosen. As can be seen in Figure 35, the name suggested by the user ("frontal weight") is used instead of the suggestion of GPT-3.5 for the "fnlwgt" column. The changed attribute names are displayed in a blue cell.

4.3.6 Single value attributes

Figure 36: Recommendations section for the single value attributes

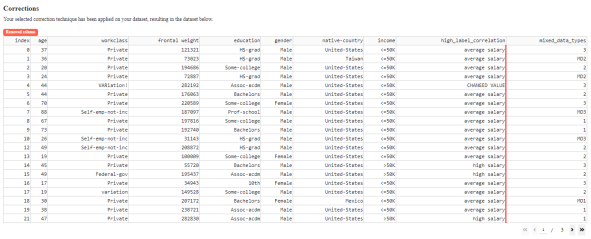


Figure 37: Corrections section for the single value attributes

The recommended correction technique for the single value attributes is to remove those attributes from the dataset. Other options for the user are "Keep all" and "Select keep/remove", both similar as in previous sections.

The corrected dataset is displayed, where in this case the single value columns have been removed from the dataset. The removed columns are shown using a red line, as can be seen in Figure 37.

4.3.7 Mixed data type attributes

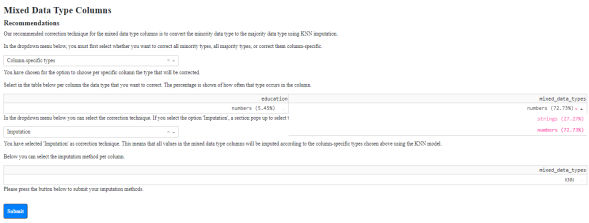


Figure 38: Recommendations section for the mixed data type attributes

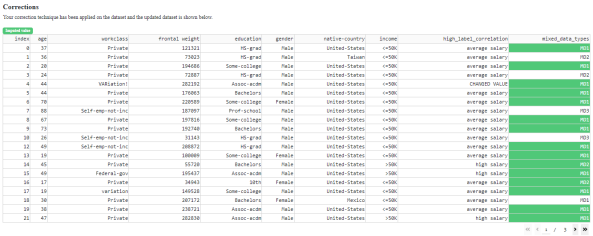


Figure 39: Corrections section for the mixed data type attributes

The Recommendations part for the mixed data type attributes is quite extensive. Before choosing the correction technique, the user has to select what data type(s) he/she wants to correct. The recommendation is to correct the minority data types of all mixed data type attributes to the majority data type. But, the user has the ability to choose to correct the majority data type instead or choose per column what type to correct (see Figure 38). For the latter option, the user is shown a table with the mixed data type columns and he/she can select per column what type should be corrected (numbers or strings). After choosing the data types that should be corrected, the correction technique itself can be selected. The recommendation is to use imputation, but the user can also select "Remove rows" or "Keep all".

After submitting the to be addressed data types per column and the correction technique(s), the Corrections part shows the updated dataset. In Figure 39, all numbers were imputed with string values for the "mixed_data_types column". The "education" column did not get addressed, because the mixed data types were already addressed during the missing values step for that column.

4.3.8 Incorrect labels

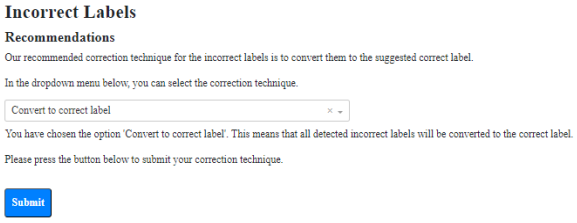


Figure 40: Recommendations section for the incorrect labels

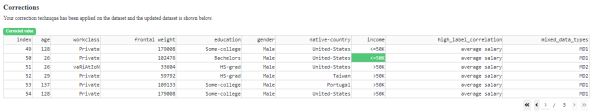


Figure 41: Corrections section for the incorrect labels

In the Recommendations part for the incorrect labels, the user is recommended to convert the incorrect labels to the correct (validated/user-added) labels (see Figure 40). Additionally, the user can choose for the options "Remove rows" and "Keep all".

After submitting the correction technique, the dataset is updated with the corrections applied. The changed values are shown in green cells, as can be seen in Figure 41.

4.3.9 String mismatches

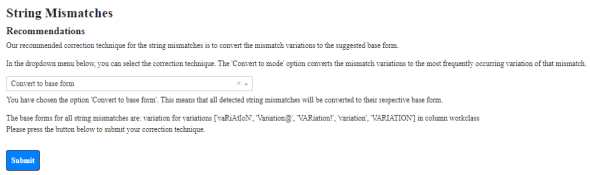


Figure 42: Recommendations section for the string mismatches

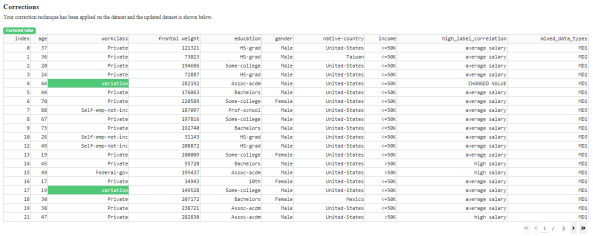
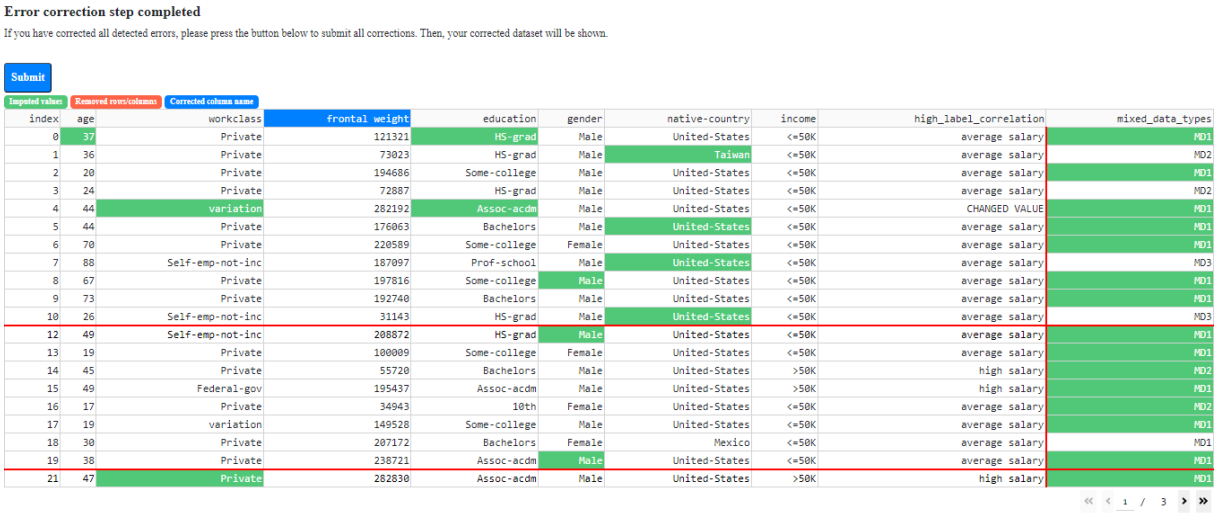


Figure 43: Corrections section for the string mismatches

Finally, the string mismatches are recommended to be converted to their base form. This is also the technique applied in Figure 42. Additionally, the user can choose for one of the other options: "Remove rows", "Keep all", and "Convert to mode". The latter option converts all variations in a string mismatch to the most frequently occurring variation in that string mismatch (i.e. the mode).

In the Corrections part, the final dataset is shown, since this was the last error type addressed. In Figure 43, it can be seen that the string mismatches have been converted to their base form.

4.3.10 Correction completed



index	age	workclass	frontal weight	education	gender	native-country	income	high_label_correlation	mixed_data_types
0	37	Private	121321	HS-grad	Male	United-States	<=50K	average salary	MD1
1	36	Private	73023	HS-grad	Male	Taiwan	<=50K	average salary	MD2
2	20	Private	194686	Some-college	Male	United-States	<=50K	average salary	MD1
3	24	Private	72887	HS-grad	Male	United-States	<=50K	average salary	MD2
4	44	variation	282192	Assoc-acdm	Male	United-States	<=50K	CHANGED VALUE	MD1
5	44	Private	176063	Bachelors	Male	United-States	<=50K	average salary	MD1
6	70	Private	220589	Some-college	Female	United-States	<=50K	average salary	MD1
7	88	Self-emp-not-inc	187097	Prof-school	Male	United-States	<=50K	average salary	MD3
8	67	Private	197816	Some-college	Male	United-States	<=50K	average salary	MD1
9	73	Private	192740	Bachelors	Male	United-States	<=50K	average salary	MD1
10	26	Self-emp-not-inc	31143	HS-grad	Male	United-States	<=50K	average salary	MD3
12	49	Self-emp-not-inc	208872	HS-grad	Male	United-States	<=50K	average salary	MD1
13	19	Private	100009	Some-college	Female	United-States	<=50K	average salary	MD1
14	45	Private	55720	Bachelors	Male	United-States	>50K	high salary	MD2
15	49	Federal-gov	195437	Assoc-acdm	Male	United-States	>50K	high salary	MD1
16	17	Private	34943	10th	Female	United-States	<=50K	average salary	MD2
17	19	variation	149528	Some-college	Male	United-States	<=50K	average salary	MD1
18	30	Private	207172	Bachelors	Female	Mexico	<=50K	average salary	MD1
19	38	Private	238721	Assoc-acdm	Male	United-States	<=50K	average salary	MD1
21	47	Private	282830	Assoc-acdm	Male	United-States	>50K	high salary	MD1

Figure 44: Overview of the cleaned dataset after all corrections have been performed

After all detected errors have been corrected and the corrections have been submitted, the user is shown its cleaned dataset. In this cleaned dataset representation, all imputed/changed values are marked in green cells, all removed rows and columns are marked with red lines, and all changed column names are marked in blue (see Figure 44). This helps the user easily evaluate whether the changes made are correct.

4.3.11 Update and download

Figure 45: Final section of the tool where the user can update the dataset on OpenML and download the cleaned dataset

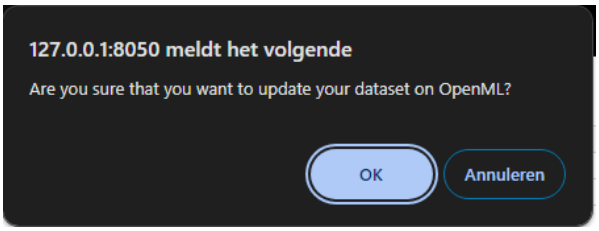


Figure 46: Confirmation window to confirm that the user wants to update the dataset on OpenML

In the final section of the tool, the user can do two things: download its cleaned dataset and update the "dirty" dataset on OpenML with the cleaned version. If the 'Download' button is pressed, the cleaned dataset (visible in Figure 44) is converted to a .csv file and downloaded. If the 'Update' button is pressed, and the user confirms its request in the confirmation dialogue pop up (Figure 46), the original dataset on OpenML is replaced by the cleaned version.

5 Results

5.1 Column renaming

The Bert F1 score and the Exact Match (EM) score were calculated for each of the five different queries (with 0 example instances, 1 instance, 3, 5, or 10). This score was then standardized through division of the number of tokens used for that specific query type. The results in Table 1 show that the best performance in terms of the F1 score is for the query with 3 example instances with a score of 0.966. This type also performs the best in terms of EM, together with the queries with 5 and 10 example instances: all obtained an EM score of 0.799. The performance of the query type using 0 example instances performs on both metrics the worst, which is logical considering GPT-3.5 receives less information.

By taking into account the number of tokens used for each query type (on average), we are able to determine which query type is the best in this scenario. Obviously, the query type with the most tokens used is the one with 10 example instances and the one with the least tokens used is the query type with zero example instances. After standardizing each query type's F1 score and EM score to the number of tokens used, we find that no query type can improve the baseline of zero example instances significantly; on both metrics the zero example instances query type has the best standardized score. After that query, the next best query type is the query using one example instance.

Table 1: Comparison of performance of query types that use the 0, 1, 3, 5, or 10 example instances to obtain the best column names

Query Type	F1	EM	Tokens	Norm. F1	Norm. EM
0 examples	0.946	0.718	**89.65**	**0.01055**	**0.00801**
1 example	0.956	0.765	146.15	0.00654	0.00523
3 examples	**0.966**	**0.799**	223.35	0.00432	0.00358
5 examples	0.964	**0.799**	299.45	0.00322	0.00267
10 examples	0.963	**0.799**	490	0.00196	0.00163

For the second experiment, we thus used as a baseline a query with zero example instances added. Then, we tested whether adding a title, description or both would improve the standardized F1 score and EM score. The results in Table 2 show that the best performance in terms of both F1 and EM is obtained using the query type with the title and description of the dataset added. This query type obtained an F1 score of 0.965 and an EM score of 0.812. The second best query type, based on the F1 score, is the one using only the title, and based on the EM score, the one using only the description. The baseline has the worst performance on both F1 and EM, but its performance is not much worse. After standardizing the scores, the baseline again comes out on top. The standardized performances between the baseline and the query type with title are very close, but the small performance increase on F1 and EM does not outweigh the increase in the extra tokens used. The query types with the description included do perform very well, but they also use close to 10 times as much tokens as the baseline.

Table 2: Comparison of performance of query types that use the title, description, both, or neither to obtain the best column names

Query Type	F1	EM	Tokens	Norm. F1	Norm. EM
Baseline	0.948	0.739	**89.65**	**0.01057**	**0.00825**
Title	0.962	0.769	97.8	0.00984	0.00787
Description	0.957	0.778	878.1	0.00109	0.00088
Title + Description	**0.965**	**0.812**	884.25	0.00109	0.00092

5.2 Tool performance

In this subsection, the performance of the tool in improving the data quality of datasets on OpenML will be assessed. We sampled 20 datasets and evaluate the performance of the 'original'/'dirty' and 'clean' versions of each dataset on a downstream model task. 10 datasets had a classification task (the target feature was categorical) and 10 datasets had a regression task (the target feature was numerical). 5 different downstream models (RF, LR, KNN, DT, GB) are used and the best performance for the dirty dataset is compared to the best performance for the clean dataset. The best performances for both the dirty and clean datasets are marked in bold font and the best performance of those two is outlined with a bold box.

Figure 47 shows the results for the 10 datasets with a classification task. The performance is evaluated using the F1 score and the accuracy score. For 3 out of 10 datasets, the original dataset performs the best,

for 6 out of 10 datasets, the clean dataset performs the best and for dataset ID 3, both dataset versions perform equally on both the F1 score and accuracy.

	Original												Clean											
	RF		LR		KNN		DT		GB		Average		RF		LR		KNN		DT		GB		Average	
Dataset ID	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc	F1	Acc
2	0,89	0,90	0,69	0,76	0,80	0,82	0,87	0,88	**0,90**	**0,91**	0,83	0,85	**1,00**	**1,00**	0,75	0,82	0,89	0,90	**1,00**	**1,00**	**1,00**	**1,00**	0,93	0,94
3	0,94	0,94	0,93	0,93	0,78	0,78	**0,98**	**0,98**	0,94	0,94	0,91	0,91	0,95	0,95	0,93	0,93	0,78	0,78	**0,98**	**0,98**	0,95	0,95	0,92	0,92
5	**0,65**	**0,72**	0,61	0,62	0,51	0,62	0,56	0,56	**0,65**	0,69	0,60	0,64	0,69	**0,75**	0,68	0,70	0,51	0,62	0,68	0,68	**0,72**	0,73	0,66	0,70
31	**0,74**	**0,76**	0,69	0,71	0,62	0,65	0,69	0,68	**0,74**	0,75	0,70	0,71	**0,84**	**0,84**	0,73	0,75	0,64	0,67	0,75	0,75	0,83	0,83	0,76	0,77
38	**0,91**	**0,94**	**0,91**	**0,94**	**0,91**	0,93	**0,91**	**0,94**	**0,91**	**0,94**	0,91	0,94	**0,98**	0,98	0,96	0,96	0,91	0,93	**0,98**	0,98	**0,98**	**0,99**	0,96	0,97
44	**0,93**	**0,93**	0,90	0,90	0,77	0,77	0,88	0,88	**0,93**	**0,93**	0,88	0,88	**0,90**	**0,90**	0,85	0,85	0,74	0,74	0,84	0,84	0,89	0,89	0,84	0,84
458	0,98	0,98	**0,99**	**0,99**	0,98	0,98	0,82	0,83	0,93	0,92	0,94	0,94	**0,94**	**0,94**	0,91	0,91	0,93	**0,94**	0,89	0,90	0,93	0,93	0,92	0,92
1050	**0,88**	**0,90**	0,87	**0,90**	0,84	0,88	0,86	0,85	**0,88**	**0,90**	0,87	0,89	0,91	**0,93**	0,90	0,92	0,89	0,91	0,91	0,91	**0,92**	**0,93**	0,91	0,92
1053	**0,76**	0,79	0,73	0,76	0,74	0,76	0,72	0,71	0,75	**0,80**	0,74	0,76	**0,86**	**0,87**	0,81	0,82	0,82	0,84	0,83	0,82	**0,86**	**0,87**	0,84	0,84
1464	0,64	0,67	**0,71**	**0,77**	0,64	0,67	0,60	0,61	0,67	0,71	0,65	0,69	0,60	0,62	**0,70**	**0,73**	0,58	0,60	0,60	0,61	0,66	0,69	0,63	0,65

Figure 47: Performance results of the datasets on a classification task comparing the original dataset to the cleaned version. The bold numbers are the best performances for that dataset (original or clean) on that performance metric. The bold boxes are the best performances of both datasets (original and clean).

Figure 48 shows the results for the 10 datasets with a regression task. The performance is evaluated using the MAE and the RMSE. Based on MAE, the original dataset performs the best for 2 out of 10 datasets and the cleaned dataset for 5 out of 10 datasets. The performance was equal for datasets 189, 491, and 673. Based on RMSE, the original dataset performs the best for 2 out of 10 datasets and the cleaned dataset performs the best for 4 out of 10 datasets. Furthermore, there were 4 draws in performance for dataset ID 189, 210, 639, and 673.

	Original												Clean											
	RF		LR		KNN		DT		GB		Average		RF		LR		KNN		DT		GB		Average	
Dataset ID	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
8	2,88	3,41	**2,80**	**3,36**	3,02	3,65	3,69	4,69	2,88	3,45	3,05	3,71	2,87	3,40	**2,73**	**3,28**	3,10	3,71	3,52	4,35	2,90	3,46	3,02	3,64
189	0,11	0,14	0,16	0,20	**0,09**	**0,12**	0,16	0,21	0,14	0,18	0,13	0,17	0,11	0,14	0,16	0,20	**0,09**	**0,12**	0,16	0,21	0,14	0,18	0,13	0,17
204	42,81	54,76	**40,24**	**52,50**	43,32	55,64	56,38	71,98	45,00	57,41	45,55	58,46	40,26	50,33	**37,93**	**48,12**	41,37	51,80	56,68	69,69	42,89	53,33	43,83	54,65
210	0,31	0,49	**0,26**	**0,39**	0,35	0,54	0,37	0,59	0,30	0,47	0,32	0,50	**0,25**	**0,39**	0,27	0,42	0,31	0,47	0,32	0,49	0,28	0,40	0,29	0,43
491	0,66	0,90	**0,60**	**0,77**	0,63	0,86	0,72	1,13	0,69	0,98	0,66	0,93	0,66	0,93	**0,60**	**0,79**	0,65	0,90	0,71	1,14	0,72	1,02	0,67	0,96
560	0,49	1,48	0,55	**1,19**	5,00	6,00	0,78	2,23	**0,48**	1,36	1,46	2,45	**0,49**	1,46	0,59	**1,23**	4,98	5,99	0,76	2,30	**0,49**	1,35	1,46	2,47
566	**202,80**	685,51	244,50	**624,64**	256,01	748,67	263,52	1015,13	214,32	727,84	236,23	760,36	**12,96**	**17,34**	13,78	18,90	15,45	20,80	19,43	27,40	14,78	20,63	15,28	21,01
639	0,67	0,85	0,97	1,16	0,72	0,92	0,88	1,21	**0,59**	**0,79**	0,77	0,99	0,67	0,86	0,97	1,16	0,72	0,92	0,88	1,16	**0,60**	**0,79**	0,77	0,98
673	0,42	0,52	**0,41**	**0,51**	0,42	0,53	0,55	0,69	0,43	0,54	0,45	0,56	**0,41**	0,52	**0,41**	**0,51**	0,42	0,53	0,57	0,71	0,42	0,54	0,45	0,56
41700	281,51	486,13	1274,65	1467,99	1403,92	1690,89	353,51	776,57	**262,55**	**467,15**	715,23	977,75	275,64	472,80	1259,52	1523,58	1345,59	1709,48	388,57	805,77	**260,27**	**456,03**	705,92	993,53

Figure 48: Performance results of the datasets on a regression task comparing the original dataset to the cleaned version. The bold numbers are the best performances for that dataset (original or clean) on that performance metric. The bold boxes are the best performances of both datasets (original and clean).

6 Discussion

6.1 Tool deployment

6.1.1 Appearance

Each error section of the tool is carefully constructed to make the experience for the user as good as possible. The first method to accomplish this is by using colored badges for every section to make the user understand the seriousness of that specific error type in the dataset. Red badges represent a serious problem, orange badges are a minor problem and green badges represent no problem. In these badges, values are placed that provide the user with a summary of the presence of that error type in the dataset. The amount of errors is displayed and the percentage of those errors over the whole dataset.

The second method to make the experience as pleasant as possible for the user is by providing a lot of information about the error type in a specific error section. Not all users of the tool will have sufficient data science knowledge to understand what all errors mean. That is why each section has a short explanation to ensure that the user understands what the problem is and why it should be addressed.

The third and final method to create a good experience for the user is by using colors in (almost) all error detection and correction sections. For example, in the 'Missing values' error detection section, the

'regular' missing values (NaN, None, NA, etc.) are displayed in red and the 'disguised' missing values are displayed in orange. This instantly helps the user see what the different missing values in the dataset are and evaluate how important they are. Colors are also used in the final subsection of the tool (Figure 44) to display all changes to the original dataset. All deletions are displayed in red, all changed values in green and all changed column names in blue.

6.1.2 HITL

Throughout the whole implementation of the tool the Human In The Loop (HITL) principles were adhered to. Although we have implemented high-quality algorithms to detect and correct all errors, the tool is not flawless. In Section 4.2, where the different sections of the error detection step of the tool were shown, one could already see a few mistakes. For example, in the 'Missing values' section, the value 'Variation@' in the column 'workclass' was detected as a disguised missing value. This synthetic data value was added to the dataset to be detected during the 'String mismatch' step; it is not a missing value. This example displays the necessity of the HITL principles. The user is needed to check all error detections to correct any mistakes made by the tool. Furthermore, the tool may also miss errors that should have been detected. Thus, the second reason why we need HITL is for the user to add any additional errors, which were not detected by the tool.

Also during the error correction step, the HITL principles were used. The tool recommends corrections for each detected error, but these recommendations are not fixed. For example, the cryptic column names are recommended to be changed to GPT-3.5's suggestions, but the user can also change the column names him-/herself or just keep the previous 'cryptic' column names. For all imputation corrections, the imputation method recommended is based on Figure 1, but the user is able to change the method per column (or choose one method for the whole dataset). The third reason to use the HITL principles is thus to correct the correction recommendations of the tool.

6.2 Column renaming

The results of the column renaming experiment showed that the best query type is one using zero example instances and no title or description. First, we experimented with different amounts of example instances. We tested whether 0, 1, 3, 5, or 10 example instance(s) led to the best performance. The experiment showed that although the query with 3 example instances had the highest F1 score, after standardizing all F1 scores with the number of tokens used in the query, the baseline with zero example instances performed the best, as can be seen in Table 1. The improvement in the F1 score did thus not outweigh the downgrade in the number of tokens used. This is mainly because the baseline (of zero example instances) already performed really well, with very limited information. There is just not a lot of room for improvement for the other query types and since they use a lot more tokens, their standardized performance can hardly ever be better than the baseline.

In the second experiment, we tested whether we could improve the performance even more by adding the title of the dataset, the description of the dataset, or both to the query. Adding these variables indeed improved performance, as can be seen in Table 2, but (obviously) also saw an increase in the number of tokens used. For the query types containing the description, the downgrade in the number of tokens used was way stronger than the improvement in the F1 score. For the title-only type, however, it was very close. This can be explained by the fact that the titles on OpenML cannot be longer than 128 characters. These titles thus do not use many tokens, yet can provide quite valuable information to GPT-3.5. However, the performance increase was not enough to outperform the baseline on standardized scores. This can be due to the fact that there are also many uninformative titles as can be seen in Table 5, such as 'cmc', 'ar1', and 'xd6'. Thus, to conclude, the normalized F1 and EM scores are still the best for the baseline query type, using no extra information such as the title and/or description. The final query that will be used in the tool for any dataset will thus look as follows:

As abbreviations of column names from a table, the column names c_name | pCd | dt stand for Customer Name | Product Code | Date. From a dataset, the abbreviated column names {cryptic column names} stand for

6.3 Tool performance

The results of the experiment testing whether the tool improves the quality of datasets on OpenML shows that this, in general, is the case. As discussed in Section 5.2, the cleaned version performed the best in 6 out of 10 datasets for the classification task, and the dirty version only 3 out of 10 datasets. For the regression task, the cleaned version (depending on the performance metric) performed the best 5 or 4 out of 10 datasets based on MAE and RMSE respectively, and the dirty version 2 out of 10 datasets based on both metrics. Additionally, there were 3 draws in performance based on MAE and 4 draws based on RMSE. The cleaned datasets, in general, thus perform better, but the results are not staggering.

However, some explanations can be given for this. First of all, the tool cleaned the dataset purely based on its recommendations; there was no human in the loop. Potential false detections, missed detections or inefficient corrections have thus not been corrected by the user. Therefore, the performance is expected to be worse than when a user intervenes in the detections and corrections of the tool. Secondly, the original datasets were often not exactly the original datasets. For the use of almost all downstream models, no "NaN" values could be present in the dataset. Therefore, before running any of the models on a dataset, we first had to drop the "NaN" values, which could have potentially already improved performance. Thirdly, although the tool detects and corrects the most important errors in datasets, there are some errors it does not address which might influence the result. For example, features that are highly correlated with the target feature. Finally, a dataset with a higher 'data quality' does not necessarily have to lead to a better performance on a downstream model. For example, a model based on a dataset with duplicate instances removed can perform worse than a model including those duplicate instances. Duplicate instances can reinforce patterns in the data, making certain relationships more prominent, which might help the model learn these patterns better. The performance of the dirty dataset might then be better, yet its data quality is worse.

6.4 User testing

As shortly discussed in Section 2.5, feedback from third persons was also used to improve the tool. Each of the 10 persons was asked to find a dataset (could be a personal one or one from OpenML), and use the OpenML Data Cleaner to clean the dataset. They were tasked with writing down any bugs that they obtained during the cleaning process, what parts of the tool could be improved, and how. Using this feedback, some bugs that were not discovered before were discovered, for example what happens for certain error type sections if that error type is not present in the dataset. Additionally, much feedback was received on how to improve the user-friendliness of the tool, by using more colors and by providing more information on errors.

6.5 Future research

This study can be extended by future research in numerous ways. First of all, the tool can be extended with additional data errors. The 8 errors addressed by the tool are the most common data errors, but there are many more that could also be implemented. For example, feature-label correlation, class imbalance, and biased data. Secondly, the performance of the tool can be improved. The results in Section 5.2 showed that the cleaned dataset performs better than the original dataset, but the results were not very impressive. This indicates that the error detections and corrections made by the tool are not optimal and the algorithms used for those steps could thus be improved. Thirdly, the tool can be improved in terms of scalability. In the current tool, some steps can take a long time to run for larger datasets, especially the 'incorrect labels' section. Future research can research how the same performance can be maintained (or improved) while the time efficiency is improved. Fourthly, Active Learning, as described in Krishnan et al. (2016) and Neutatz et al. (2019), can be implemented in the tool. Active Learning queries for error types in the most uncertain cases to the user to learn from his/her input. Implementing this could significantly improve the performance of the tool. Finally, the tool's interface could be improved to make the tool even more intuitive and user-friendly. Some sections, for example the 'string mismatch' error detection section, is not very visibly pleasing. No colors are implemented in that section and a lot of text is shown. Such sections could use a more user-friendly design.

7 Conclusion

In this paper, the process of creating a (semi-)automatic data cleaning tool for OpenML was described. The tool consists of an error detection and an error correction section in which multiple common data errors are addressed. The errors addressed are, in order: missing values, duplicates (instances and attributes), outliers (values and instances), cryptic attribute names, single value attributes, mixed data type attributes, incorrect labels, and string mismatches. For each error type, the tool uses specific algorithms to both detect and correct the errors.

The tool uses many elements to make the experience as ideal as possible for the user. Summary statistics, colored badges, colored errors, and textual information were used throughout the tool to make the user understand each error and the seriousness of that error in the dataset thoroughly. Additionally, HITL principles were used throughout the tool for 3 reasons: to correct false error detections, to add undetected errors, and to correct error correction recommendations. That is why the tool is not fully automatic, but semi-automatic; the user is needed to optimize the performance of the tool.

The performance of the tool was evaluated on 20 OpenML datasets using downstream model tasks (10 classification and 10 regression tasks). The original dataset was compared to the cleaned version by calculating the performance on 5 different downstream models (RF, LR, KNN, DT, and GB) using 5-fold cross-validation. The results showed that, in general, the performance improved by using the cleaned versions, but the results were not very convincing.

References

Amro Abbas, Kushal Tirumala, Daniel Simig, S. Ganguli, and Ari S. Morcos. 2023. Semdedup: Data-efficient learning at web-scale through semantic deduplication. *ArXiv*, abs/2303.09540.

Mustafa Alabadla, Fatimah Sidi, Iskandar Ishak, Hamidah Ibrahim, Lilly Suriani Affendey, Zafienas Che Ani, Marzanah A Jabar, Umar Ali Bukar, Navin Kumar Devaraj, Ahmad Sobri Muda, et al. 2022. Systematic review of using machine learning in imputing missing values. *IEEE Access*, 10:44483–44502.

Gustavo EAPA Batista and Maria Carolina Monard. 2003. An analysis of four missing data treatment methods for supervised learning. *Applied artificial intelligence*, 17(5-6):519–533.

Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big?. In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, pages 610–623.

Dimitris Bertsimas, Colin Pawlowski, and Ying Daisy Zhuo. 2018. From predictive methods to missing data imputation: an optimization approach. *Journal of Machine Learning Research*, 18(196):1–39.

Jakramate Bootkrajang and A. Kabán. 2014. Learning kernel logistic regression in the presence of class label noise. *Pattern Recognit.*, 47:3641–3655.

Jason Brownlee. 2020. Data cleaning, feature selection, and data transforms in python. *Machine Learning Mastery*.

Soroosh Ghorbani and Michel C Desmarais. 2017. Performance comparison of recent imputation methods for classification tasks over binary data. *Applied Artificial Intelligence*, 31(1):1–22.

Joseph M Hellerstein. 2013. Quantitative data cleaning for large databases.

Anil Jadhav, Dhanya Pramod, and Krishnan Ramanathan. 2019. Comparison of performance of data imputation methods for numeric dataset. *Applied Artificial Intelligence*, 33(10):913–933.

Sebastian Jäger, Arndt Allhorn, and Felix Bießmann. 2021. A benchmark for data imputation methods. *Frontiers in big Data*, 4:693674.

Johannes Jakubik, Michael Vössing, Niklas Kühl, Jannis Walk, and Gerhard Satzger. 2024. Data-centric artificial intelligence.

Won Kim, Byoung-Ju Choi, Eui-Kyeong Hong, Soo-Kyung Kim, and Doheon Lee. 2003. A taxonomy of dirty data. *Data mining and knowledge discovery*, 7:81–99.

Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. 2009. Loop: local outlier probabilities. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1649–1652.

Sanjay Krishnan, Michael J Franklin, Ken Goldberg, Jiannan Wang, and Eugene Wu. 2016. Activeclean: An interactive data cleaning framework for modern machine learning. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2117–2120.

Kamakshi Lakshminarayan, Steven A Harp, and Tariq Samad. 1999. Imputation of missing data in industrial databases. *Applied intelligence*, 11(3):259–275.

Peng Li, Xi Rao, Jennifer Blase, Yue Zhang, Xu Chu, and Ce Zhang. 2021. Cleanml: A study for evaluating the impact of data cleaning on ml classification tasks. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 13–24. IEEE.

Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A configuration-free error detection system. In *Proceedings of the 2019 International Conference on Management of Data*, pages 865–882.

Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. 2018. Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics*, 19(6):1236–1246.

Felix Neutatz, Mohammad Mahdavi, and Ziawasch Abedjan. 2019. Ed2: A case for active learning in error detection. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 2249–2252.

Curtis G Northcutt, Tailin Wu, and Isaac L Chuang. 2017. Learning with confident examples: Rank pruning for robust classification with noisy labels. *arXiv preprint arXiv:1705.01936*.

Ahmet Murat Ozbayoglu, Mehmet Ugur Gudelek, and Omer Berat Sezer. 2020. Deep learning for financial applications: A survey. *Applied soft computing*, 93:106384.

Caterina Penone, Ana D Davidson, Kevin T Shoemaker, Moreno Di Marco, Carlo Rondinini, Thomas M Brooks, Bruce E Young, Catherine H Graham, and Gabriel C Costa. 2014. Imputation of missing data in life-history trait datasets: which approach performs the best? *Methods in Ecology and Evolution*, 5(9):961–970.

H. Perez and J. Tah. 2020. Improving the accuracy of convolutional neural networks by identifying and removing outlier images in datasets using t-sne. *Mathematics*.

Jason Poulos and Rafael Valle. 2018. Missing data imputation for supervised learning. *Applied Artificial Intelligence*, 32(2):186–196.

Abdulhakim A Qahtan, Ahmed Elmagarmid, Raul Castro Fernandez, Mourad Ouzzani, and Nan Tang. 2018. Fahes: A robust disguised missing values detector. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2100–2109.

Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *arXiv preprint arXiv:1702.00820*.

Zahra Salekshahrezaee, Joffrey L. Leevy, and T. Khoshgoftaar. 2021. A reconstruction error-based framework for label noise detection. *Journal of Big Data*, 8:1–16.

Peter Schmitt, Jonas Mandel, and Mickael Guedj. 2015. A comparison of six methods for missing data imputation. *Journal of biometrics & biostatistics*, 6(1):1.

RS Somasundaram and R Nedunchezhian. 2011. Evaluation of three simple imputation methods for enhancing preprocessing of data with missing values. *International Journal of Computer Applications*, 21(10):14–19.

Julia Stoyanovich, Bill Howe, and HV Jagadish. 2020. Responsible data management. *Proceedings of the VLDB Endowment*, 13(12).

Yige Sun, Jing Li, Yifan Xu, Tingting Zhang, and Xiaofeng Wang. 2023. Deep learning versus conventional methods for missing data imputation: A review and comparative study. *Expert Systems with Applications*, page 120201.

Olga Troyanskaya, Michael Cantor, Gavin Sherlock, Pat Brown, Trevor Hastie, Robert Tibshirani, David Botstein, and Russ B Altman. 2001. Missing value estimation methods for dna microarrays. *Bioinformatics*, 17(6):520–525.

Chih-Fong Tsai and Ya-Han Hu. 2022. Empirical comparison of supervised learning techniques for missing value imputation. *Knowledge and Information Systems*, 64(4):1047–1075.

Sarah Webb et al. 2018. Deep learning for biology. *Nature*, 554(7693):555–557.

Steven Euijong Whang and Jae-Gil Lee. 2020. Data collection and quality challenges for deep learning. *Proceedings of the VLDB Endowment*, 13(12):3429–3432.

Katarzyna Woźnica and Przemysław Biecek. 2020. Does imputation matter? benchmark for predictive models. *arXiv preprint arXiv:2007.02837*.

Renzhi Wu, Sanya Chaba, Saurabh Sawlani, Xu Chu, and Saravanan Thirumuruganathan. 2020. Zeroer: Entity resolution using zero labeled examples. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1149–1164.

Ke Yang, Biao Huang, Julia Stoyanovich, and Sebastian Schelter. 2020. Fairness-aware instrumentation of preprocessing˜ pipelines for machine learning. In *Workshop on Human-In-the-Loop Data Analytics (HILDA'20)*.

Daochen Zha, Zaid Pervaiz Bhat, Kwei-Herng Lai, Fan Yang, Zhimeng Jiang, Shaochen Zhong, and Xia Hu. 2023. Data-centric artificial intelligence: A survey. *arXiv preprint arXiv:2303.10158*.

Hongbao Zhang, Pengtao Xie, and Eric Xing. 2018. Missing value imputation based on deep generative models. *arXiv preprint arXiv:1808.01684*.

Jiani Zhang, Zhengyuan Shen, Balasubramaniam Srinivasan, Shen Wang, Huzefa Rangwala, and George Karypis. 2023. Nameguess: Column name expansion for tabular data. *arXiv preprint arXiv:2310.13196*.

A Example Appendix

Table 3: Comparison of different research papers on the topic of MV imputation

Research Paper	Compared Methods	Results Summary	Nr. of Datasets	Datasets Sizes	Missing Ratio (MR)	Feature Types
Schmitt et al. (2015)	Mean KNN FKM SVD bPCA MICE	- bPCA and FKM perform well for all - MICE and KNN good for small - SVD good for large - FKM takes long time	4	small (2) large (2)	5% 15% 25% 35% 45%	Num (4)
Li et al. (2021)	Delete HoloClean Mean_mode Median_mode Mode_mode Mean_dummy Median_dummy Mode_dummy	- All similar performance	6	large (5) ? (1)	∼1% (1) ∼2% (3) ∼8% (1) ? (1)	Num (1) Cat (1) Mix (3) ? (1)
Jäger et al. (2021)	Mean_mode KNN RF DL (Discriminative) VAE GAIN	- Overall, RF best, KNN and DL good - For high MR (and MNAR): Mean_mode - DL takes long time to run	69	large (69)	1% 10% 30% 50%	Num (14) Cat (5) Mix (50)

Tsai and Hu (2022)	MLP SVM CART KNN NB	- MLP best for mixed and numerical datasets, but takes twice as long as CART (2nd best) - KNN good for mixed datasets with small MR - KNN poor performance on numerical datasets CART best performance for categorical	33	small (17) large (16)	10% 20% 30% 40% 50%	Num (15) Cat (10) Mix (8)
Poulos and Valle (2018)	Mode Random KNN Log. R RF SVM	- KNN best in most situations	2	large (2)	0% 10% 20% 30% 40%	Cat (1) Mix (1)
Jadhav et al. (2019)	Random Mean Median KNN PMM Bayesian Lin. R Lin. R non-Bayesian	- KNN performed best for each dataset-MR combination (except 1) - PMM also okay - Ranking is consistent across datasets and MRs	5	small (5)	10% 20% 30% 40% 50%	Num (5)
Woźnica and Biecek (2020)	Random Mean SVD RF KNN Hot Deck MICE (PMM & LR)	- Could not find single best method - Depends on performance measure and downstream model	13	small (6) large (7)	1%-33%	Cat (1) Mix (12)
Zhang et al. (2018)	Iterative EM Zero Mean Median MICE RF SVD KNN PCA AE DAE RAE	- Their method (iterative EM) outperformed other methods - DAE, Mean and RAE also good - Consistent results across MR: SVD, Zero and MICE always worst, KNN and RF always medium	13	small (9) large (4)	25% 50% 75%	Num (6) Cat (3) Mix (4)

Bertsimas et al. (2018)	Optimal Impute Mean KNN Iterative KNN Bayesian PCA PMM MICE	- Their method (Optimal Impute) outperformed other methods - Their method can have long running times for big datasets	84	small (?) large (?)	10% 20% 30% 40% 50%	Num (?) Cat (?) Mix (?)
Penone et al. (2014)	KNN PMM RF Phylopars	- PMM, RF and Phylopars equivalent, KNN worse	1	small (1)	10% 20% ... 70% 80%	Mix (1)
Troyanskaya et al. (2001)	SVD KNN Mean	- KNN performed better than SVD, which performed better than Mean	3	small (?) large (?)	1% 5% 10% 15% 20%	Num (3)
Batista and Monard (2003)	KNN C4.5 CN2	- KNN best, also for large MRs - KNN did not perform well for dataset with highly correlated features	4	small (3) large (1)	10% 20% ... 50% 60%	Mix (4)
Ghorbani and Desmarais (2017)	RF MIEM Sequential Hot-Deck MILR	- Generally, MIEM the best - MILR only for low MRs - Differs per downstream model what imputation method leads to the (best) improvements	14	small (2) large (12)	5% 10% 20% 30% 40% 50%	Cat (14)
Sun et al. (2023)	GAIN (onehot) GAIN (embed) VAE (onehot) VAE (embed) MICE RF	- For small dataset size: conventional methods (RF and MICE) better than DL	10 (7 real, 3 synthetic)	small (4) large (6)	10% 30% 50%	Num (3) Cat (2) Mix (5)

Dataset ID	Dataset Name	Nr of Features	Nr of Instances	Task
2	anneal	39	898	classification
3	kr-vs-kp	37	3196	classification
5	arrhythmia	280	452	classification
8	liver-disorders	6	345	regression
31	credit-g	21	1000	classification
38	sick	30	3772	classification
44	spambase	58	4601	classification
189	kin8nm	9	8192	regression
204	cholesterol	14	303	regression
210	cloud	6	108	regression
458	analcatdata_authorship	71	841	classification
491	analcatdata_negotiations	6	92	regression
560	bodyfat	15	252	regression
566	meta	22	528	regression
639	fri_c3_100_25	26	100	regression
673	chscase_census2	8	400	regression
1050	pc3	38	1563	classification
1053	jm1	22	10885	classification
1464	blood-transfusion-service-center	5	748	classification
41700	CPMP-2015-regression	27	2108	regression

Table 4: Dataset information for downstream performance evaluation

Dataset ID	Dataset Name	Nr of Columns	Nr of Non-Cryptic	Nr of Cryptic
4	labor	17	17	0
9	autos	26	18	8
10	lymph	19	6	13
13	breast-cancer	10	6	4
23	cmc	10	9	1
24	mushroom	23	21	2
36	segment	20	13	7
50	tic-tac-toe	10	10	0
54	vehicle	19	15	4
55	hepatitis	20	14	6
171	primary-tumor	18	9	9
185	baseball	17	10	7
187	wine	14	6	8
1059	ar1	30	14	16
1100	PopularKids	11	9	2
40691	wine-quality-red	12	11	1
40693	xd6	10	10	0
40701	churn	21	17	4
40708	allrep	30	10	20
41760	FOREX_eurhuf-day-High	12	9	3

Table 5: Dataset information for the cryptic column names experiment