# Transparent Data Preprocessing for Machine Learning

Sebastian Strasser
sebastian.strasser@ur.de
University of Regensburg
Regensburg, Germany

Meike Klettke
meike.klettke@ur.de
University of Regensburg
Regensburg, Germany

## ABSTRACT

Data preprocessing is an important task in machine learning which can significantly improve model outcomes. However, evaluating the impact of data preprocessing is often difficult. There is a need for tools which make it transparent to the user on how certain transformations conducted in preprocessing affect the data. Thus, we propose a vision of a transparency system for data preprocessing that provides insights into data preparation pipelines. Our envisioned system consists of a Python library which enables users to log transformations and processed data. Subsequently, the system generates summaries of the data which was processed in the pipeline and so-called change profiles which capture the changes conducted in each processing step. These abstractions offer insight into the transformations and their effects on data. Additionally, the system includes an user interface where users can interactively discover the implemented pipeline and the changes made during preprocessing. This paper presents an initial concept of such a system. It also examines further challenges related to making preprocessing transparent and discusses potential solutions to address these challenges.

## CCS CONCEPTS

• **Information systems** → *Data cleaning*; *Extraction, transformation and loading*; • **General and reference** → **Evaluation**; *Validation.*

## KEYWORDS

data preprocessing, data profiles, change profiles, transparency

**ACM Reference Format:**
Sebastian Strasser and Meike Klettke. 2024. Transparent Data Preprocessing for Machine Learning. In *Workshop on Human-In-the-Loop Data Analytics (HILDA 24), June 14, 2024, Santiago, AA, Chile.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3665939.3665960

## 1 INTRODUCTION

Data preprocessing is a tedious task which often takes a substantial percentage of time spent on data science projects. This is also due to it not being a streamlined process. It can better be explained as an iterative approach where the optimal processing steps and parameters are found based on trial-and-error. The aim of data preprocessing for machine learning is to find a data representation which yields the best result, i.e., the data format where the machine learning model can fit the input to the output in the best possible way. Therefore, data scientists conducting the experiments measure the effectiveness of their data preprocessing by drawing upon machine learning metrics yielded when evaluating the model.

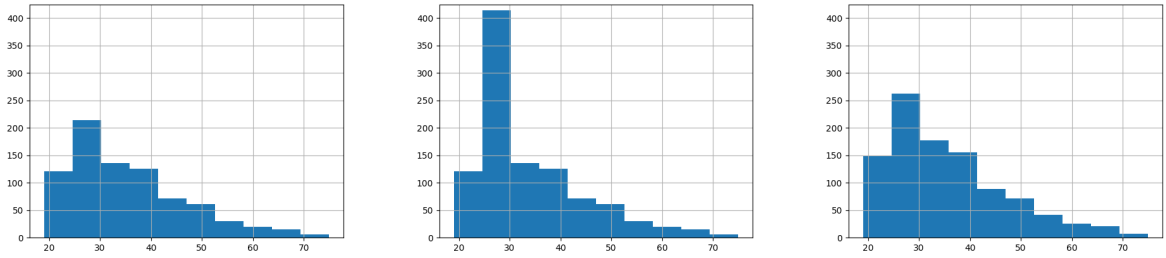
This approach has some drawbacks: as multiple data processing steps are performed on the data, it is not always clear which of them leads to an improvement or detoriation of model performance. Also, data preprocessing is only one factor influencing machine learning metrics, next to model and hyperparameter selection. Another problem is that especially non-experts cannot assess the effects of their preprocessing pipeline accurately. This is also due to the experimental nature of the model training process. Many decisions have to be made in a data science project, e.g., which preprocessing algorithms to use, what kind of machine learning model is suitable for the task, and what parameters are suitable for both preprocessing and machine learning algorithms. This can be overwhelming for unexperienced, but in some cases even for professional data scientists. Significant effort is spent on solving these issues by the *AutoML* community. Not only model-specific tasks like hyperparameter optimization or (in the case of deep learning) neural architecture search, but also data-specific tasks like data preparation and feature engineering are automated more and more [13]. Even though these methods can help to broaden the adoption of machine learning methods, they often have the problem that users do not know if they can *trust* these systems. A study [7] concluded that *transparency*, i.e., displaying information which might have led to automated decisions, is an important factor in order to gain trust.

Another problem which necessitates the evaluation of data preparation is erroneous data preprocessing. An example for this is unintentionally introducing technical bias when instrumenting preprocessing tasks on the data. An example is shown in Figure 1. Here, a data imbalance is caused by imputing missing values of the feature *age* with the most frequent value (=27). This data imbalance in turn can cause a technical bias in the data as it is often the case in machine learning software [4]. Making changes more explicit can help in detecting such problems.

Due to this number of reasons, we argue there is a need for a *transparency system for data preprocessing*. This gives insight into the preprocessing pipeline and enables the detection of potential problems and errors. It also allows for larger trust in automatically or manually built pipelines. A first necessary component is a tracking system which extracts metadata from data preprocessing pipelines. An emphasis here is on comprehensive summaries of the data processed in the pipelines. In a subsequent step, the collected metadata is utilized to create a visual presentation which facilitates an observation and assessment of the resulting pipeline. A further


*HILDA 24 , June 14, 2024, Santiago, AA, Chile*

ACM ISBN 979-8-4007-0693-6/24/06
https://doi.org/10.1145/3665939.3665960



**(a) Histogram of column *age* with a missing rate of 20%**

**(b) Histogram of column *age* with imputed values**

**(c) Histogram of column *age* (Ground Truth)**

**Figure 1: Overrepresentation of people with age = 27 by imputing missing values with the mode**

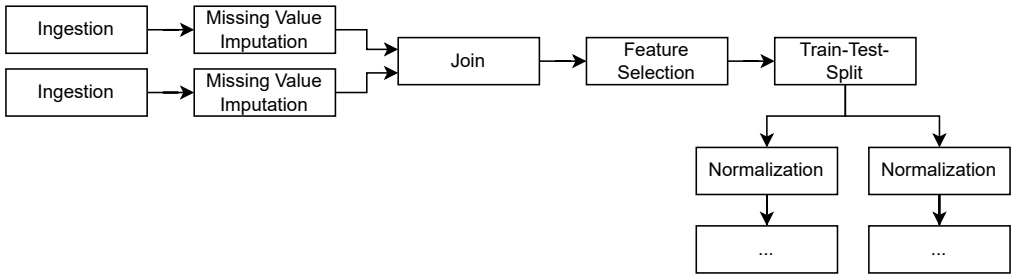



**Figure 2: Exemplary DAG representing a preprocessing pipeline**

idea is to give the possibility of a formal comparison between intermediates, i.e., data which is produced by preprocessing tasks. This enables an even better understanding of the implications of steps taken in preprocessing.

This paper gives an overview of the general approach and first developments. Furthermore, open challenges and possible solutions are exemplified.

## 2 PRELIMINARIES

Data preprocessing usually consists of several *tasks* performed subsequently on the input data, composing a *pipeline*. The most commonly performed operations are intended at cleaning, transforming, and reducing data [10]. An important challenge especially in supervised learning is class imbalance. One possible way of dealing with this challenge is to utilize data preprocessing techniques like oversampling in order to balance the classes [2]. Thus, class imbalance handling can be seen as another class of preprocessing techniques.

As proposed by other authors [12, 22], pipelines for data preprocessing can be represented as directed acylic graphs (DAGs). Figure 2 shows a DAG denoting a typical preprocessing pipeline for a machine learning application where two data sets are processed. The preprocessing DAG $P$ can be defined as a graph with nodes $T$ denoting tasks and edges $F$ which describe the data flow. As can be seen in Figure 2, there are different types of tasks (based on the number of inputs and outputs) in a pipeline:

(i) *Transformation* tasks take one input and yield one output.
(ii) *Integration* tasks take $n$ ($n \geq 2$) inputs and merge them into one output.
(iii) *Splitting* tasks take one input and splits it into $n$ ($n \geq 2$) outputs.

A combination of such tasks results in the preprocessing DAG which thus represents the underlying workflow. We develop an approach to provide the opportunity of evaluating the resulting DAGs. For this, annotations for transformation tasks and the processed data is generated.

## 3 RELATED WORK

In this section, we review related work which inspired the idea of a transparency system for data preprocessing.

*Tracking machine learning experiments.* There is a number of tools which aim at providing transparency into the model training process by tracking metadata about it. *MLflow Tracking* [23] is a prominent example for such a system. It provides an API to log parameters and results in machine learning experiments. Experiments can then be compared in an interactive UI. A similar system is *ModelDB* [21] which automatically tracks hyperparameters, metrics, and model types from different environments (namely spark.ml, scikit-learn, and R environments for machine learning). Pipelines and summaries of experiments are also visualized in a dashboard in order to inspect and compare them. Our transparency system also uses the approach of logging data from the process, collecting information about it, and displaying this information in an UI. However, the information which is collected and presented

in our proposed system mainly concerns the training data and tasks performed in preprocessing, not the machine learning model.

*Evaluating data preprocessing.* Effects of preprocessing are mostly measured by looking at how operations affect the model outcome, i.e., downstream model metrics are used to assess different preprocessing schemes. There are multiple surveys and studies which examine the effects of preprocessing on estimator efficacy in specific domains, e.g., for heart disease classification [3] or rainfall forecast models [8].

However, machine learning metrics are primarily intended for measuring model quality and are also influenced by a lot of factors other than preprocessing. Thus, Gonzalez Zelaya [11] proposes *volatility* as a general metric to quantify the effects of preprocessing. It enables the user to figure out for which data points predictions are most likely to change under different preprocessing schemes. Thus, it is a metric to measure which data is most sensitive to preprocessing tasks. In our work, we are also developing general approaches for evaluating the effects of preprocessing. However, we take a broader look on the data which is processed and evaluate changes by comparing data characteristics.

*Debugging data preprocessing.* Numerous tools enable the inspection and debugging of data in ML training processes (e.g., [5, 12, 17]). The library *mlinspect* [12] provides lightweight data debugging in ML pipelines with the main goal of detecting distribution errors, i.e., technical bias. A similar tool is *dagger* [17] – it gives a more detailed insight into pipelines by recording intermediate pipeline states. It tracks so-called *code-handled data*, i.e., pandas DataFrames or built-in Python types which are processed in Python scripts and builds a dataflow based on the collected data and the logged operations. It then provides a SQL-based DSL to search for provenance information in the pipeline, e.g., what value an attribute has at a specific processing block. Mining provenance information is a feature many of these tools share (e.g., [5, 16]). Another way of inspecting data in ML pipelines is to use data validation tools like *Deequ* [19] or *great expectations*[1] which provide general data statistics and user-defined constraints to validate data.

Our system to be implemented is specialized for ML pipelines and will provide enhanced functionality for these. This especially includes a fine-grained tracking of typical machine learning preprocessing tasks like feature selection and class imbalance handling.

# 4 TRANSPARENCY SYSTEM FOR DATA PREPROCESSING PIPELINES

Our system is intended to complement approaches which aim at making decision processes of machine learning models explainable and therefore transparent (e.g. [14, 18]). In order to ensure transparency over the whole training pipeline, operations conducted on the data before the actual model building have to be explicit. Thus, the goal of our proposed system is to make the whole process engineering the data which is fed into model training *interpretable*, *reproducible*, and *explainable*. This way, data engineers can evaluate the preprocessing pipeline and make informed decisions about altering it. An emphasis of our proposed system lays on explaining changes in data characteristics which were induced by transformation tasks. These preprocessing operations often influence model outcome significantly. Therefore, our emphasis is on *making changes between intermediate datasets explicit*. We are planning on doing this by creating so-called *change profiles*. These can be thought of as a diff files between datasets and utilize summaries of the datasets in the form of *data profiles* created beforehand.

This section gives an overview of the envisioned system and also introduces an example in order to illustrate the potential of a transparency system for machine learning pipelines.

## 4.1 Concept and First Developments

Figure 3 shows the architecture of our proposed *transparency system for machine learning preprocessing* (first developements were already made[2]). It consists of two major components: the first one is responsible for *metadata collection*, the other one is an UI which supports the user in interactively exploring and inspecting the preprocessing pipeline by utilizing the metadata collected before. In between, there is a component which analyzes the collected before and creates abstractions that help users in exploring and evaluating changes conducted in preprocessing. In order to enable a comparison between different pipelines, a database is created which allows the storage of collected metadata. The architecture allows a comprehensive exploration of the preprocessing pipeline while only adding little overhead to the actual process.

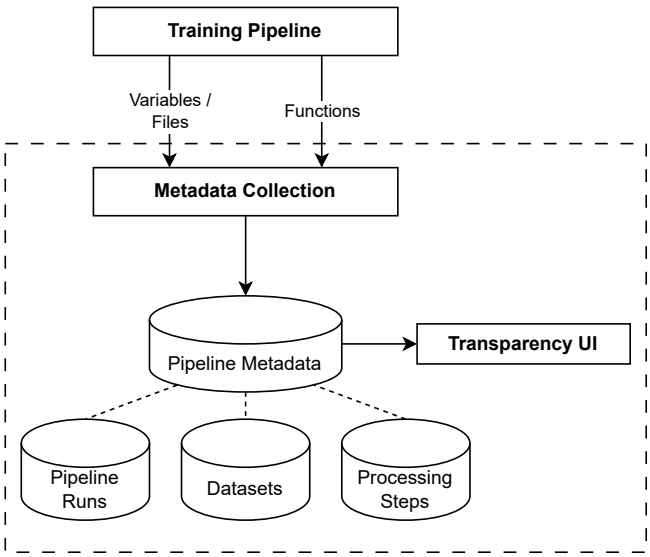



**Figure 3: High-level architecture of the proposed transparency system**

*Collecting and storing pipeline metadata.* Machine learning experiments are often conducted in Python scripts or Jupyter Notebooks where numerous external libraries can be utilized. There are different ways to collect metadata from these programs. One way is to provide an API for the user to log variables and other information

[1] https://greatexpectations.io/

[2] https://github.com/bastistrasser/hawk

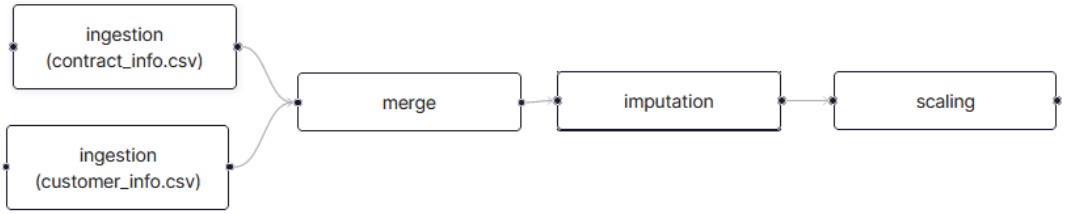



**Figure 4: Preprocessing DAG for a simplified churn prediction pipeline**

which is the basis for the metadata. Another possibility is to run the program and extract variables and function calls (which could translate to the execution of transformation tasks) at runtime. In a first step, we chose the first possibility and implemented a Python library which provides logging functionality for machine learning pipelines. For now, we only provide the possibility of logging pandas DataFrames. The logging library can be easily extended for supporting different ways to collect metadata and also log data of other data formats.

We define three main entities, namely *data sets*, *processing steps* and *pipeline runs*. *Data sets* correspond to all input, intermediate, and output variables/files which are processed in preprocessing. *Processing step* entities store references to their input and output data sets and an annotation which describes the transformation task performed. Lastly, *pipeline runs* represent the structure of the preprocessing DAG by mapping processing steps to their input and output data sets. They also have additional metadata like the start and end time of the pipeline run.

Pipeline runs can then be stored together with its corresponding data sets and processing steps to a MongoDB database in order to view them in the UI. The reason for using a database with flexible schema is to provide the possibility of adaptable data profiles.

*Extracting pipeline information.* In the next step, the metadata collected before is used to generate information about the pipeline. In a first step, a *data profile* is created for every data set. This data profile contains *descriptive statistics* not only of *columns* but also of *inter-column relationships*. We focus on profiling tabular data. When considering tabular data, one could utilize comprehensive work on the profiling of relational data [1]. Various tools were developed specifically to profile data used in machine learning applications [6, 9]. However, an extensive profiling is not always feasible for the application of creating data profiles in preprocessing, as it would be too expensive for many use cases. Thus, a trade-off between getting insightful information about the data and performance has to be made. A first draft of a data profile which considers these requirements is shown in Figure 5. The data profile contains general information and descriptive statistics describing the data set. We track statistics about numeric and categorical features, as well as relationships between columns. The data profile is not meant to be final, but rather serves as a suggestion which can be adjusted based on the use case. Here, we give the possibility to interactively adapt this data profile to the user.

The abstraction of *change profiles* is then used to make changes conducted on the data by transformation tasks explicit. We are planning on using an approach similar to *Data Diff* [20] which provides a summary of the differences between two data sets. The basis for this change profile is our summary of a data set in the form of a data profile.

We are planning to find ways to measure the *significance* of data changes. The answer to this question would be a tremendous aid for data scientists as it points only to transformations which are of interest for them. Thus, it could act as a tool to pre-select data changes for the user. The challenge, obviously, is how to infer significance. Firstly, we consider the type of transformation task conducted on the data. For example, when instrumenting a normalization mechanism on a column, one would expect the ranges to shift, but not the mean. Thus, a first step would be to define types of transformations and create a mapping from those types to data characteristics which are likely to be affected by them. Effects on other data characteristics would hint at potential errors. In a next step, it is measured to which *extent* a transformation manipulated the dataset. For this, we plan to use distance measures between the data profiles generated for the intermediates. Large deviations could point to *significant* changes.

*Displaying pipeline information.* Our proposed system also provides an UI which presents the changes conducted during preprocessing in an interpretable manner. A central component of the UI is an illustration of the preprocessing DAG. It serves as an overview of the whole workflow and depicts dependencies between the processing steps. Changes conducted in preprocessing can be observed and assessed interactively by clicking on tasks and data flows. Here, data profiles generated and stored before are shown. Also, plots illustrating the information are generated and displayed to the user.

There are also different approaches for presenting the extracted pipeline information to the user. Various visualization techniques can be applied in this step. What is especially of interest is to find ways to visualize the differences between intermediates processed

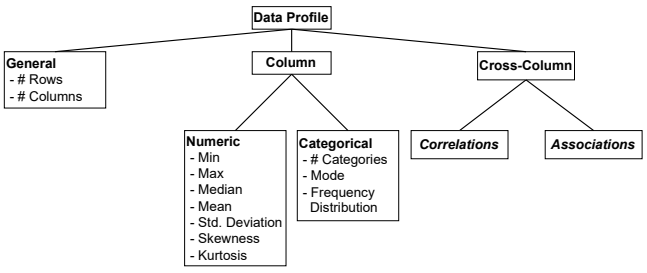



**Figure 5: Classification of data profile components**

in the pipeline in order to enable users to better compare them, thus giving them the possibility of evaluating changes conducted by processing steps.

## 4.2 Example

In this section, we present an example for the planned usage of our envisioned transparency system. A simple preprocessing pipeline is given which processes data for churn prediction. Two data sets are used in this example: one where general customer info (like age and gender) is stored and another where information about the contract (e.g., monthly charges, included services, etc.) is stored. These data sets are joined on the customer id. Afterwards, missing values are imputed and numerical attributes are normalized.

Our library *hawk* provides intuitive functions to log data. To start, one has to initialize a *PipelineRun*. Processing steps can then be added to this pipeline run. The library calculates data profiles for all intermediate data sets which are processed and created by the processing steps. The information logged and created by the library can then be sent to the server.

```
from hawk import PipelineRun, log_data
import pandas

pr = PipelineRun()

@log_data(run, "data_ingestion")
def read_data_from_file(filename):
    try:
        return pandas.read_csv(filename)
    except FileNotFoundError:
        print("Insert a valid file")
        return None

...

customer_info = read_data_from_file("customers.csv")
contract_info = read_data_from_file("contracts.csv")

merged_data = customer_info.join(
    contract_info,
    on="customer_id"
)
pr.add_processing_step(
    description="merge",
    input_datasets=[customer_info, contract_info],
    output_datasets=[merged_data]
)
pr.send_to_server(host=<host>, port=<port>)
```

**Listing 1: Example for logging data in preprocessing scripts with hawk**

Users can then evaluate the pipeline in the transparency dashboard. An important component of the UI is the preprocessing DAG which is depicted in Figure 4 for our example. From this DAG, data and changes of data can be observed by selecting processing steps or data which is flowing through the nodes. Data profiles from the intermediates before and after the processing step are both depicted for the user to compare (see Figure 6 as an example).

Once implemented, we are planning to include change profiles and also significance measures to the UI in order to provide a more interpretable and specific evaluation of a preprocessing pipeline.

Columns | Plots

Numeric columns:

| name | missing_rate | min | max | mean | median | std | mad | skewness | kurtosis |
|---|---|---|---|---|---|---|---|---|---|
| MonthlyCharges | 0.00 | 18.25 | 118.75 | 64.76 | 70.35 | 30.09 | 24.05 | -0.22 | -1.26 |
| tenure | 0.00 | 0 | 72 | 32.37 | 29.00 | 24.56 | 22.00 | 0.24 | -1.39 |

Categorical columns:

| name | missing_rate | num_of_categories | mode |
|---|---|---|---|
| Churn | 0.00 | 2 | No |
| Contract | 0.00 | 3 | Month-to-month |
| DeviceProtection | 0.00 | 3 | No |
| InternetService | 0.00 | 3 | Fiber optic |

**Figure 6: Excerpt of the data profile for the raw contract info data set**

# 5 RESEARCH CHALLENGES AND LIMITATIONS

The transparency system is in an early development stage. Therefore, the prototype is not fully functional yet and various challenges in the overall process are yet to be tackled. This section highlights these research challenges and discusses first thoughts on how they could be solved.

*Research challenges in metadata collection.* Parts of the library for logging the data are already implemented. One idea to improve the metadata collection addresses the creation of annotations for transformation tasks. In the presented stage, annotations have to be manually created by calling methods and functions provided by the developed logging library. This means additional manual effort for users. Therefore, approaches for an automatic generation of these annotations are called for. One possible idea is to utilize a large language model fine-tuned for source code to generate a summarization of the preprocessing pipeline. From this summary, annotations can be extracted – again – by utilizing a large language model or other NLP methods. The challenge in this context is engineering the prompts [15] in a way such that the model outputs the desired annotations. A benchmark is required in order to evaluate this approach, too.

*Research challenges in extracting transparency information.* Research challenges remain in the context of extracting information about the preprocessing pipeline. An important information is a representation of changes between intermediate datasets. For this, we are planning to introduce a formal notation which captures the problem of finding differences between data sets before and after a preprocessing step. This will help to make changes more explicit and interpretable for users. To achieve this, approaches for creating change profiles and measuring *significance* are called for. We plan to capture the following kind of changes in this step:

- Structural changes
- Changes in data characteristics (e.g., the mean of a numeric feature)
- Changes in data distribution

Data scientists can then evaluate the whole pipeline by utilizing these change representations. Research challenges here include

the formalization of the problem and the design of change profiles. First inspiration can be drawn from *Data Diff* [20] which aims at generating interpretable summaries of data distribution changes.

An additional challenge is to find a way to evaluate the output, i.e., transparency information extracted from the pipeline. This includes the semantic correctness of summaries of datasets, change profiles, and significance of changes. Also, user studies are necessary in order to evaluate how useful the transparency information is for creating or changing preprocessing pipelines.

*Research challenges in displaying transparency information.* Another interesting research direction is the visualization of changes conducted by preprocessing operators. This further aids the user in the observation process. One major requirement is the display of information on different granularities, starting with a high-level overview of the process and the possibility of zooming into changes on single columns. A challenge in this context is to find suitable visual representations of changes which enables users to quickly capture them.

*Performance considerations.* Performance is an important factor to consider in our approach. In metadata collection, the computational overhead added to the actual training pipeline has to be minimal. Simply logging all intermediates in the preprocessing pipeline and creating data profiles during the execution adds significant overhead to training process which are often costly already. An advantage is that metadata collection and pipeline execution do not have to be synchronous. If data stored in files is to be tracked, an option would be to just log file locations and read them asynchronously. In the case of huge datasets processed as runtime-variables, sampling could help to decrease the computational effort.

Additionally, the complexity of creating change profiles and other abstractions conveying transparency information has to be studied and tried to be kept minimal.

*Limitations.* While our proposed transparency system can give detailed insight into the effects of preprocessing, it can not evaluate the data preparation process in a fully automated manner. Thus, the final decision if data changes conducted by transformation tasks are significant has to be made by the data scientist. Also, a resolution of errors is not the goal of the tool. The transparency system rather focuses on the tracking of changes in datasets and intends to aid the data scientist in the process of deciding on the best preprocessing pipeline.

## 6 CONCLUSION

Evaluating the effects of preprocessing on data and ultimately the model is an important task which is not always trivial. In this paper, we firstly examined the state of the art in the field of evaluating and debugging data preprocessing and detected a lack of general-purpose tools which evaluate the effects of data preparation by looking at the data itself. Therefore, we proposed a transparency system for data preprocessing which collects comprehensive metadata on input, intermediate, and output data. The system then extracts information useful for the data scientist in order to evaluate data transformations. This allows the data scientist to make informed decisions about the design of the pipeline. A first prototype and an example use case was developed as starting point for further research. Research challenges we identified include an automatic collection of metadata about the preprocessing pipeline. Also, approaches which enable a formal and visual representation of changes in data between data transformations are to be found.

Another research direction could be to utilize transparency information extracted from the pipeline to recommend preprocessing mechanisms. This could be implemented by comparing different preprocessing schemes based on data and change profiles generated from our proposed transparency system.

## REFERENCES


[1] Ziawasch Abedjan, Lukasz Golab, and Felix Naumann. 2015. Profiling relational data: a survey. *The VLDB Journal* 24, 4 (2015).

[2] Haseeb Ali et al. 2019. A review on data preprocessing methods for class imbalance problem. *International Journal of Engineering & Technology* 8, 3 (2019).

[3] H. Benhar, A. Idri, and J.L. Fernández-Alemán. 2020. Data preprocessing for heart disease classification: A systematic literature review. *Computer Methods and Programs in Biomedicine* 195 (2020).

[4] Joymallya Chakraborty, Suvodeep Majumder, and Tim Menzies. 2021. Bias in Machine Learning Software: Why? How? What to Do?. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*.

[5] Adriane Chapman et al. 2020. Capturing and Querying Fine-Grained Provenance of Preprocessing Pipelines in Data Science. *Proc. VLDB Endow.* 14, 4 (2020).

[6] Fabiana Clemente et al. 2023. ydata-profiling: Accelerating data-centric AI with high-quality data. *Neurocomputing* 554 (2023).

[7] Jaimie Drozdal et al. 2020. Trust in AutoML: exploring information needs for establishing trust in automated machine learning systems. In *Proceedings of the 25th International Conference on Intelligent User Interfaces (IUI)*.

[8] Isa Ebtehaj et al. 2019. Evaluation of preprocessing techniques for improving the accuracy of stochastic rainfall forecast models. *International Journal of Environmental Science and Technology* 17 (2019).

[9] Will Epperson et al. 2024. Dead or Alive: Continuous Data Profiling for Interactive Data Science. *IEEE Transactions on Visualization and Computer Graphics* 30, 1 (2024).

[10] Salvador García et al. 2016. Big data preprocessing: methods and prospects. *Big Data Analytics* 1, 9 (2016).

[11] Carlos Vladimiro Gonzalez Zelaya. 2019. Towards Explaining the Effects of Data Preprocessing on Machine Learning. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*.

[12] Stefan Grafberger et al. 2022. Data distribution debugging in machine learning pipelines. *The VLDB Journal* 31, 5 (2022).

[13] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems* 212 (2021).

[14] Scott M. Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS)*.

[15] Ggaliwango Marvin et al. 2024. Prompt Engineering in Large Language Models. In *Data Intelligence and Cognitive Informatics*.

[16] Leonardo Murta et al. 2015. noWorkflow: Capturing and Analyzing Provenance of Scripts. In *Provenance and Annotation of Data and Processes*.

[17] El Kindi Rezig et al. 2020. Dagger: a data (not code) debugger. In *Online Proceedings of the 10th Conference on Innovative Data Systems Research (CIDR)*.

[18] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*.

[19] Sebastian Schelter et al. 2018. Automating Large-Scale Data Quality Verification. *Proc. VLDB Endow.* 11, 12 (2018).

[20] Charles Sutton et al. 2018. Data Diff: Interpretable, Executable Summaries of Changes in Distributions for Data Wrangling. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (SIGKDD)*.

[21] Manasi Vartak et al. 2016. ModelDB: a system for machine learning model management. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA)*.

[22] Ke Yang et al. 2020. Fairness-Aware Instrumentation of Preprocessing Pipelines for Machine Learning. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA)*.

[23] Matei A. Zaharia et al. 2018. Accelerating the Machine Learning Lifecycle with MLflow. *IEEE Data Eng. Bull.* 41 (2018).