Design and Development of a Provenance Capture Platform for Data Science

Luca Gregori*, Paolo Missier†, Matthew Stidolph‡, Riccardo Torlone* and Alessandro Wood*
*Department of Engineering
Roma Tre University, Italy
Email: {gregori,torlone,wood}@uniroma3.it
†School of Computer Science, University of Birmingham, UK
Email: p.missier@bham.ac.uk
‡Newcastle University, UK
Email: matthewstidolph@gmail.com

***Abstract*—As machine learning and AI systems become more prevalent, understanding how their decisions are made is key to maintaining their trust. To solve this problem, it is widely accepted that fundamental support can be provided by the knowledge of how data are altered in the pre-processing phase, using data provenance to track such changes. This paper focuses on the design and development of a system for collecting and managing data provenance of data preparation pipelines in data science. An investigation of publicly available machine learning pipelines is conducted to identify the most important features required for the tool to achieve impact on a broad selection of pre-processing data manipulation. This reveals that the operations that are used in practice can be implemented by combining a rather limited set of basic operators. We then illustrate and test implementation choices aimed at supporting the provenance capture for those operations efficiently and with minimal effort for data scientists.**

I. Introduction

The question of how to make AI *explainable* has been receiving much attention in the past few years, especially in areas of science where AI-generated results are required to stand the scrutiny of human experts [1]. Consider for example a model aimed at predicting disease risk, based on features that describe each individual's clinical but also socio-demographic profile, family history, and other variables. Once the model has been trained, its users will want to know how a specific prediction was generated, which variables have the most impact, and why. Many techniques have been proposed to answer these questions, which variously apply depending on the type of data. For tabular datasets, for example, Shapley values are very commonly used [2].

Before the model can even be deployed and used in clinical settings, however, it needs to be validated as a requirement for trust [3], [4]. This requires not only the ability to inspect details of the model training process (was the right model used, how was overfitting avoided, how was model performance assessed) but also the ability to query the data preparation pipeline through which the training set was constructed [5]. Such pipelines can be complex but they typically include standard categories of operators. For example, the process may start with the extraction of two complementary datasets that represent the same cohort of patients from two independent data sources, by applying specific inclusion criteria to each of them. Each of the two sets may require ad hoc cleaning, and the results will then need to be merged, possibly requiring a record linkage operation (approximate join) when there is no common identifier. The resulting integrated dataset may then undergo further processing, for instance, imputation across multiple variables, upsampling, or downsampling depending on the downstream machine learning task.

Validators may question each of these steps: how did you make sure that the correct inclusion criteria were implemented[1]? Which algorithm was used to perform record linkage, was there any control on false positives and false negatives? what criteria were used for downsampling, which algorithm was used for upsampling, etc.

The example above evidences the need to keep a record of the operations that are performed at each step. However, it is also important to track the effect that those operations have on the data, in a way that makes it possible to answer questions at the level of individual data items, eg "why has patient X been included in /excluded from the study?", "where does this unusual value for variable Y in patient X come from?" (it could have been imputed or originated from a data source, or obtained through normalisation). Note that these questions are relevant not only to establish the scientific validity of a data transformation process but also to explain how issues of data bias were addressed, especially when dealing with data that are imbalanced concerning sensitive attributes [6], [7].

In this paper we advocate the need for comprehensively tracking the provenance of datasets that are used to train predictive models, at the level of individual data items, to enable answering questions such as those above. In this setting, the term data provenance, originally proposed with reference to database queries [8], refers to records of data transformations, from raw data sources to the point where the data are used to inform (train) Machine Learning and

[1]An example of inclusion criteria is the following: include all and only the patients who have been admitted to the hospital at least once in the past five years. This seemingly simple selection may be tricky to implement correctly, for instance when the codelists used to indicate the type of hospital admission are not fully understood.

especially AI models (a formal and more general definition of provenance that is not restricted to data is given by the World Wide Web Consortium (W3C) as "a record of the entities and processes that are involved in the production and delivery of or otherwise influence a particular resource" [9]).

Adding provenance to the data, we argue, engenders trust in AI-based models by improving their transparency, enabling explainability of the end-to-end process, but also facilitating its reproducibility. Specifically, provenance records associated with a dataset at some intermediate processing step typically include items such as references to its original source, references to the addition or removal of records and features relative to the original, and references to linkage or join operations. At a more granular level, in some cases, it may also be possible to track the derivation of individual items within the dataset through specific operations, for instance, normalisation across a whole column.

The main contribution in this paper complements our earlier work on Data Provenance for Data Science [10] and consists of a system, implemented at prototype level, for automatically tracking granular provenance through Python scripts where data is in the form of Pandas dataframes, in a way that is maximally transparent and minimally intrusive to the programmer. We begin by presenting an analysis of typical usage of data operators in the wild, looking at pipelines available from MLBazaar and Kaggle, and then frame the problem in terms of a taxonomy of data processing operations, before illustrating the system in action on three well-known benchmark pipelines.

II. A SURVEY OF DATA PROCESSING PIPELINES

To establish an understanding of the importance of pre-processing operations, a survey of publicly available machine learning pipelines is conducted, with each operation present in the pre-processing stage recorded. Operations are only recorded if they act upon the master dataset before the start of model training, excluding operations conducted during the exploration of the data and alterations made to the results such as reverting scaled values to their original size. Pipelines are sourced from the ML Bazaar [11], a collection of machine learning solutions to different data type and task combinations, and the popular data science community Kaggle[2].

The ML Bazaar pipelines are composed of a selection of pre-made components known as primitives, improving the pipeline's ease of understanding but limiting the range of operations present. In addition, as the pipelines are all sourced from the same location there is a potential for bias. Nonetheless, an analysis of the results is undertaken to provide a basis for understanding into general trends, though the results are given a significantly reduced level of importance when compared to those obtained from Kaggle. We have considered only pipelines operating on tabular data, given that are the most used in practice, leaving 311 of the 386 pipelines present in the database to be considered. Some details of such pipelines are reported in Table I.

[2]https://www.kaggle.com/

ML Task	Pipelines	DFS	Encoder	Imputer	Scaler	DT Feat
Forecasting	35	6	34	35	35	0
Regression	77	34	55	77	77	0
Classification	199	86	199	199	199	4
Total	311	126	288	311	311	4

TABLE I
PRE-PROCESSING OPERATIONS IN ML BAZAAR PIPELINES.

Only five types of operation are identified from the ML Bazaar pipelines, with some form of imputing and scaling present in all 311 pipelines, and encoding in over 90%. Of the other two operations, a DateTime Featuriser (DT Feat) is extremely rare and is therefore discounted, though this is partially due to the contents of datasets, whereas Deep Feature Synthesis (DFS), an automated tool used to perform feature engineering, appears in roughly 40% of pipelines.

While the near homogeneity present in the ML Bazaar limits the potential conclusions and demonstrates the bias of a single source, it does highlight four main operation types commonly used in pre-processing: scaling, encoding, imputation, and feature engineering. This evidence is compared to the survey of Kaggle pipelines to identify similarities.

As Kaggle is a public platform that hosts pipelines, referred to by Kaggle as notebooks, created for a variety of reasons including competitions and tutorials, it is important that the correct information is investigated. As we focus on data preparation for machine learning, the survey is conducted on the top 200 most upvoted python notebooks related to machine learning on the site. Of these 200 pipelines, a small percentage are deemed irrelevant to the survey as they either do not perform pre-processing operations, or they relate to non-standard data types. The principal operations involved in such pipelines and their frequency are reported in Figure 1.

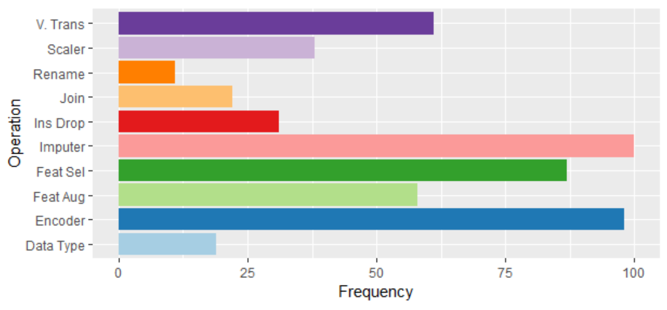


Fig. 1. Frequency of 10 most common operation types in Kaggle survey.

Across the survey, 29 unique pre-processing operations are detected, of which 12 appear in less than 10 pipelines including transposing and changing index values. Among the 10 most common operation types, imputing and encoding again lead the way appearing in 50% and 49% of the 200 pipelines respectively. The remaining two operations highlighted in the ML Bazaar survey also feature heavily, with 58 instances of feature augmentation and 38 scaling operations.

Due to the high frequency of these four operations in both the Kaggle and ML Bazaar surveys, they can be considered highly important for the wider machine learning community and therefore it is vital that a tool supporting the analysis of pre-processing operations can correctly track and map their provenance. This also applies to additional operations found in the Kaggle survey, with value transformations and feature selection among the most common findings.

The information obtained through these surveys is used in conjunction with existing knowledge of pre-processing practices to assign a level of importance to each operation [12]. The pre-existing knowledge is introduced to account for the under-representation of some key operations due to the contents of the survey. For example, adding instances to a dataset appeared in only five pipelines across the two surveys, but is a common solution to reduce bias in imbalanced datasets, and so was assigned a higher level of importance. While the final aim of our tool is to fully support all operations, testing the tool's support for essential operations is required for an initial release.

III. Overview of the approach

A. A taxonomy of data pre-processing operations

Based on the analysis done in the previous section, we have observed that the majority of pre-processing operations, including all the most used in practice, can be implemented by combining a rather small set of basic operators of data manipulation over datasets belonging to four main classes, according to the type of manipulation done on the input dataframe(s), as follows.

Data reductions: operations that take as input a dataset D and reduce its size by eliminating rows or columns from D. Two basic operators of this kind are defined over datasets. They are simple extensions of two well-known relational operators.

- the *(conditional) projection* of D on a set of features in S, given a boolean condition C, is the dataset obtained from D by including only the columns of D that satisfy C;
- the <u>selection</u> of D, given a boolean condition C, is the dataset obtained from D by including the rows of D satisfying C.

Data augmentations: operations that take as input a dataset D on a schema S and increase the size of D by adding rows or columns to D. Two basic operators of this kind are defined over datasets. They allow the addition of columns and rows to a dataset, respectively.

- the *vertical augmentation* of D to Y using a function f over a set X of features of D, is obtained by adding to each row of D a new set of features whose values are obtained by applying f to the features in X;
- the *horizontal augmentation* of D using an aggregative function f is obtained by adding one or more new rows to D obtained by first grouping over a set of features of D and then by applying f to each group.

Data transformation: operations that take as input a dataset D and, by applying suitable functions, transform (some of) the elements in D. This class includes one basic operator:

- the *transformation* of a set of features X of D using a function f is obtained by applying f to all the values occurring in X.

Data fusion: operations that take as input two datasets D_1 and D_2 and somehow combine them into a new dataset D. The two basic data fusion operators *join* and *append* allow the combination of a pair of datasets.

- the *join* of the two datasets based on a boolean condition C is the dataset obtained by applying a standard join operation (inner, (left/right/full) outer) based on the condition C;
- the *append* of the two datasets is the dataset obtained by appending D^2 to D^1 and possibly extending the result with nulls on the mismatching columns.

In Figure II are reported some common data pre-processing operators and the way in which they can be implemented by combining the above basic operators.

B. Data provenance description and collection

Figure 2 shows graphically the elements of the PROV data model [9] that we have adopted for generating explanations for the existence (or the absence) of some elements as the result of the above data manipulations.

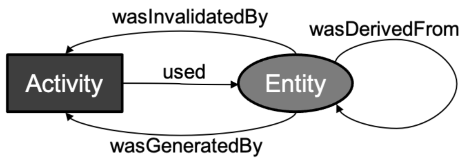


Fig. 2. The core W3C PROV model.

In PROV an entity represents an atomic element d of a dataset D. An activity represents any pre-processing data manipulation that operates over datasets. For each element d in a dataset D' generated by an operation $\mathbf{o}$ over a dataset D we represent the facts that d *wasGeneratedBy* $\mathbf{o}$, and d *wasDerivedFrom* a set of elements in D. In addition, we represent all the elements d of D such that d was *used* by $\mathbf{o}$ and all the elements d of D such that d *wasInvalidatedBy* (i.e., deleted by) $\mathbf{o}$ (if any).

Now, with each of the basic operators presented in the previous section, we associate a *provenance template* that describes the effect on the data of each of them at the appropriate level of detail, i.e., on individual data elements, columns, rows, or collections of those. By mapping effective operators available in standard libraries to these fundamental templates, we are then able to identify the transformation type based on observation of the operator's input and outputs alone. By abstracting to this level, we can automatically create the appropriate provenance for an operator in a data science pipeline if it follows the pre-identified input-output patterns, even if the operator itself has never been seen before. This approach strongly simplifies the capture mechanisms since it does not require a specific method for each practical operation.

PRE-PROCESSING OPERATIONS	BASIC OPERATORS	DESCRIPTION
Feature Selection	Conditional projection	One or more features are removed.
Instance Drop	Selection	One or more records are removed.
Feature Augmentation	Vertical Augmentation	One or more features are added.
Space Transformation	Vertical Augmentation + Conditional projection	New features are derived from old features, which can be later dropped.
One-hot encoding	Vertical Augmentation + Conditional projection	New features are derived from old features, which can be later dropped.
Instance Generation	Horizontal Augmentation	One or more records are added.
Imputation, Data Type Conversion, Renaming, Normalization, Scaler, Encoding	Transformation	Values are modified using various functions.
Dimensionality Reduction	Transformation + Conditional projection	Some features are modified, others are removed.
Integration, Cartesian Product	Join	Two or more datasets are combined based on a common attribute or key.
Concatenate	Append	Two datasets are combined by taking their union.

TABLE II
COMMON DATA PRE-PROCESSING OPERATIONS AND CORRESPONDING (COMBINATION OF) BASIC OPERATORS

IV. A PROVENANCE SYSTEM FOR DATA PREPARATION

A. *Overview of the system*

We have implemented the approach to provenance generation illustrated in Section III in a software platform called DPDS (Data Provenance for Data Science). The main component of DPDS is the Provenance-Tracker that automates the process of detecting and tracking the provenance of a user-defined pipeline of data preparation written in pandas/python, a largely used library for data manipulation and analysis[3]. The Provenance-Tracker includes a Prov-generator component that produces the provenance of each operator in the pipeline at execution time by: (i) observing the execution of operators that consume and generate datasets, (ii) analyzing the value and structural changes between the input(s) and output datasets for that command, (iii) based on the observed change pattern, select and instantiate one or more *provenance templates* to capture the dependencies between the elements of the datasets that have changed, and (iv) given its graphical nature, storing the provenance data produced by the prov-gen function in Neo4j, a world-leading and scalable graph database management system[4].

Basically, the Provenance-Tracker acts as a wrapper that encapsulates the original functions of the Pandas library and adds provenance capture capabilities by leveraging the concept of metaclass in Python, which allows us to customize all the properties and functions for a pandas dataframe.

From an operational standpoint, the Provenance Tracker is equipped with a `subscribe()` function that allows developers to subscribe to one or multiple dataframes for tracking their provenance. The invocation of this function returns the corresponding wrapped dataframes, enabling provenance generation in a transparent way during dataframe transformations.

The only required instrumentation for subscribing, e.g., two dataframes ($df1$ and $df2$ in the example) is the following:

```
tracker = ProvenanceTracker()
df1,df2 = tracker.subscribe([df1,df2])
```

After the subscription of all the dataframes involved in the preprocessing process, the signature and syntax of methods that operate on them remain unchanged, but they now operate on the wrapped dataframes and invoke the internal provenance-capture functionality through the Provenance-Tracker. As an example, a data imputation operation followed by a feature transformation can be implemented in a standard way, as follows.

```
# Imputation
df1 = df.fillna('Imputation')
# Feature transformation of column D
df1['D'] = df1['D'].apply(lambda x:x*2)
```

Similarly, provenance generation for the join operation can be done without the need to invoke additional auxiliary functions, as follows.

```
df = df.merge(right=df2,
on=['key1','key2'], how='left')
```

The activity of the Provenance Tracker can be explicitly controlled, for instance when capturing the provenance of a complex operation that consists of a sequence of more basic data transformations. This is done by using the `dataframe_tracking` property as in the example that

[3]https://pandas.pydata.org/
[4]https://neo4j.com/

follows, which implements the provenance capture of the one-hot encoding. This requires a vertical augmentation, obtained by mapping a feature involving strings to a set of boolean features, followed by the elimination of the input feature.

```
tracker.dataframe_tracking = false
dummies = pd.get_dummies(df['B'])
df = df.concat(dummies.add_prefix('B'+'_'))
tracker.dataframe_tracking = true
df = df.drop([c], axis=1)
```

In this example, the changes made by the vertical augmentation on the original dataframe are produced but taken into account only during the subsequent operation, when the `dataframe_tracking` property is set to 'true'.

B. DPDS in action

Three commonly used benchmark datasets are used for testing the tool we have developed: Iris [13], German credit [14] and Census [15]. The datasets are chosen due to the difference in instance and feature count, as well as the mixture of numeric and categorical data present in Census and Iris, as illustrated in Table III. While the datasets cannot be considered large in the context of some modern datasets, the difference in scale aims to provide some guidance on the scalability of the provenance graphs.

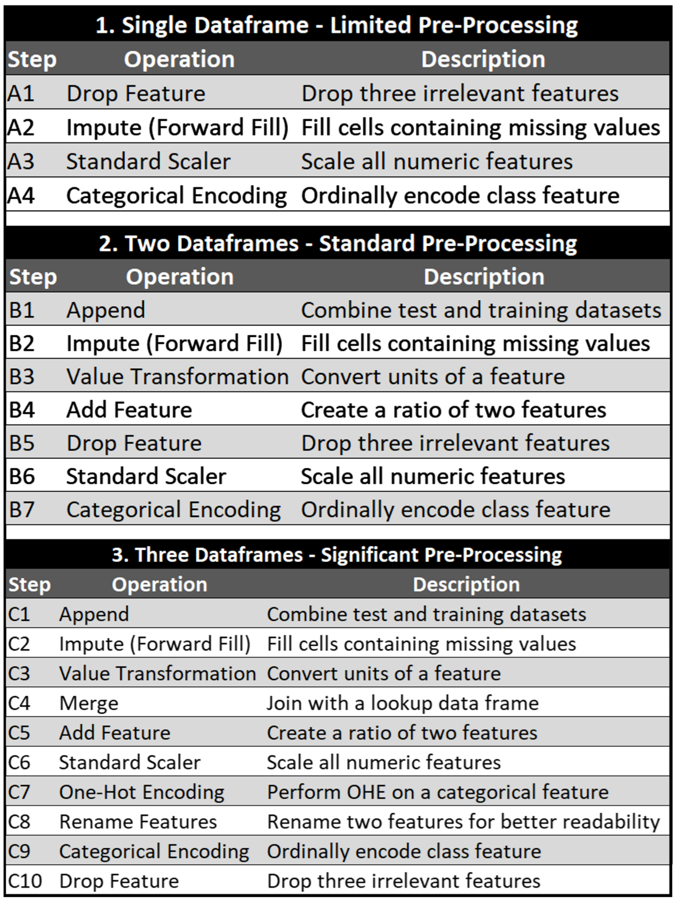
Dataset	Data Types	Features	Instances	Total Size
Iris	Num + Cat	6	150	900
German	Num	21	1000	21000
Census	Num + Cat	15	32561	488415

TABLE III
DIMENSIONS AND DATA TYPES OF BENCHMARK DATASETS.

To ensure a fair comparison between the three datasets, operations are enacted upon each using the same arguments, with exceptions only made for feature names. Results are recorded for the duration of operation, dimensions of the output dataset, and the size of the produced provenance graph. Most analyses conducted concern the size of the graph produced, as operation runtimes are not recorded to a high enough precision to draw significant conclusions in most situations. Output dimensions provide confirmation of operations executing correctly, but provide no additional information about tool performance.

Three sample pipelines are created to replicate typical pre-processing pipelines of increasing complexity, shown in Figure 3. To ensure the created pipelines are representative of typical pipelines, operations of higher priority are included. The main goal of pipeline testing is to investigate how graph size scales as more operations are introduced.

It should be noted that the operations within the pipelines are not configured to create an optimal data preparation for a real-world machine learning task, and are instead constructed to maintain similar conditions for each dataset. As such, some operations do not make sense in a practical setting due to the differing characteristics of the three datasets, for example, performing both one-hot encoding and ordinal encoding on the singular categorical feature present in the iris data in pipeline 3.



1. Single Dataframe - Limited Pre-Processing		
Step	**Operation**	**Description**
A1	Drop Feature	Drop three irrelevant features
A2	Impute (Forward Fill)	Fill cells containing missing values
A3	Standard Scaler	Scale all numeric features
A4	Categorical Encoding	Ordinally encode class feature

2. Two Dataframes - Standard Pre-Processing		
Step	**Operation**	**Description**
B1	Append	Combine test and training datasets
B2	Impute (Forward Fill)	Fill cells containing missing values
B3	Value Transformation	Convert units of a feature
B4	Add Feature	Create a ratio of two features
B5	Drop Feature	Drop three irrelevant features
B6	Standard Scaler	Scale all numeric features
B7	Categorical Encoding	Ordinally encode class feature

3. Three Dataframes - Significant Pre-Processing		
Step	**Operation**	**Description**
C1	Append	Combine test and training datasets
C2	Impute (Forward Fill)	Fill cells containing missing values
C3	Value Transformation	Convert units of a feature
C4	Merge	Join with a lookup data frame
C5	Add Feature	Create a ratio of two features
C6	Standard Scaler	Scale all numeric features
C7	One-Hot Encoding	Perform OHE on a categorical feature
C8	Rename Features	Rename two features for better readability
C9	Categorical Encoding	Ordinally encode class feature
C10	Drop Feature	Drop three irrelevant features

Fig. 3. Operation breakdown of the three sample pipelines used in testing.

The three pipelines are tested with each of the datasets, with no errors identified. In all cases, the provenance captured for each operation matches the results of the individual testing. The total size of the pipeline provenance graphs is then compared to the sum of the size of the provenance produced by each operation that composes the various pipelines. The result is shown in Table IV.

One issue with this comparison is caused by the entities created by the graph to represent the initial data points. While this occurs only once in the pipeline graph, the summation of the provenance for the single operations will include these initial entities for every operation in a pipeline. To avoid this issue of double counting, the sizes for the component operations are adjusted to ensure the initial data is only included in the sum once. For example, the component value for pipeline 1, which has four stages, has three times the initial data size subtracted. This adjusted value is used for the comparison, with results expressed as the total size of the provenance graph of the whole pipeline divided by the adjusted sum of the sizes of the provenance of the single component operations (PT/PO).

	Iris 1	Iris 2	Iris 3	Germ 1	Germ 2	Germ 3	Cens 1	Cens 2	Cens 3
prov. of operations (sum)	9544	14497	24126	212583	286786	492175	3581705	5277880	9966797
prov. of operations (adjusted)	6844	9097	16026	149583	160786	303175	2116460	2347390	5571062
prov. of pipelines (total size)	4237	7484	16517	127191	145173	286271	1549901	2002247	5877131
PT/PO	0.619	0.823	1.031	0.850	0.903	0.944	0.732	0.853	1.055

TABLE IV
COMPARISON OF PROVENANCE SIZE FOR PIPELINES AND COMPONENT OPERATIONS. ADJUSTED VALUES INCLUDED FOR COMPONENT OPERATIONS

For each dataset, the graphs relating to pipeline 1 are significantly smaller than the component operations, ranging from 61.9% of the size for Iris to 85.0% for German. This is mainly due to the order of the operations, as features are dropped in the first step leaving fewer remaining entities for the following steps to act on. This is also observed to a smaller extent in pipeline 2 where two operations occur after features are dropped and all three datasets have results around 85% of the size. While a strong conclusion cannot be drawn from these two pipelines, the relative size of the graph remaining close to the component graphs is a promising sign that suggests the tool will scale well with more complex pipelines.

Further proof is provided when examining the results for pipeline 3, in which the features are dropped as the final step and so do not affect the results. On the contrary, pipeline 3 introduces a merge operation in step 4 which adds additional entities to be acted upon. Despite this, all three datasets have graphs of similar sizes to the adjusted component values, with Iris and census less than 6% larger and the German test in fact produces a graph that is smaller than the components.

Overall, this indicates the DPDS tool performs well when tracking the provenance of an entire pipelines, with provenance correctly tracked for each operation and scales well with increases in data size and pipeline complexity.

V. CONCLUSIONS

In this paper, we have illustrated the development of a tool, called DPDS, for the collection and management of data provenance in pre-processing pipelines for the subsequent activity of machine learning . DPDS aims to address the critical need for transparency and interpretability in machine learning and AI systems, as they become increasingly integral to various domains. The current emphasis on the explanation of data preparation for the subsequent activity of machine learning underscores the significance of data provenance in ensuring trust and accountability of the whole process.

Through an investigation of publicly available machine learning pipelines, we have identified key features essential for the widespread adoption of such a tool across diverse pre-processing data manipulations. Interestingly, our findings reveal that practical pre-processing operations often stem from a relatively constrained set of fundamental operators.

By minimizing the effort required from data scientists and ensuring the effectiveness of provenance capture, our platform endeavors to facilitate the integration of data provenance practices into everyday data science workflows.

REFERENCES

[1] A. Adadi and M. Berrada, "Peeking inside the black-box: A survey on explainable artificial intelligence (xai)," IEEE Access, vol. 6, pp. 52 138–52 160, 2018.

[2] M. Sundararajan and A. Najmi, "The many shapley values for model explanation," in Proceedings of the 37th International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 9269–9278. [Online]. Available: https://proceedings.mlr.press/v119/sundararajan20b.html

[3] A. Jacovi, A. Marasović, T. Miller, and Y. Goldberg, "Formalizing trust in artificial intelligence: Prerequisites, causes and goals of human trust in ai," in Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency, ser. FAccT '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 624–635. [Online]. Available: https://doi.org/10.1145/3442188.3445923

[4] E. Alshdaifat, D. Alshdaifat, A. Alsarhan, F. Hussein, and S. M. F. S. El-Salhi, "The effect of preprocessing techniques, applied to numeric features, on classification algorithms' performance," Data, vol. 6, no. 2, 2021. [Online]. Available: https://www.mdpi.com/2306-5729/6/2/11

[5] C. V. G. Zelaya, "Towards explaining the effects of data preprocessing on machine learning," in 2019 IEEE 35th international conference on data engineering (ICDE). IEEE, 2019, pp. 2086–2090.

[6] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan, "A survey on bias and fairness in machine learning," ACM computing surveys (CSUR), vol. 54, no. 6, pp. 1–35, 2021.

[7] A. Jobin, M. Ienca, and E. Vayena, "The global landscape of AI ethics guidelines," Nature Machine Intelligence, vol. 1, no. 9, pp. 389–399, Sep. 2019, number: 9 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/s42256-019-0088-2

[8] B. Glavic et al., "Data provenance," Foundations and Trends® in Databases, vol. 9, no. 3-4, pp. 209–441, 2021.

[9] L. Moreau, P. Missier, K. Belhajjame, R. B'Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne et al., "Prov-dm: The prov data model," 2013.

[10] A. Chapman, P. Missier, L. Lauro, and R. Torlone, "DPDS: Assisting Data Science with Data Provenance," PVLDB, vol. 15, no. 12, pp. 3614 – 3617, 2022. [Online]. Available: https://vldb.org/pvldb/vol15/p3614-torlone.pdf

[11] M. J. Smith, C. Sala, J. M. Kanter, and K. Veeramachaneni, "The machine learning bazaar: Harnessing the ml ecosystem for effective system development," in Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, ser. SIGMOD '20. New York, NY, USA: ACM, 2020.

[12] F. Kamiran and T. Calders, "Data preprocessing techniques for classification without discrimination," Knowledge and information systems, vol. 33, no. 1, pp. 1–33, 2012.

[13] R. A. Fisher, "Iris," UCI Machine Learning Repository, 1988, DOI: https://doi.org/10.24432/C56C76.

[14] "South German Credit (UPDATE)," UCI Machine Learning Repository, 2020, DOI: https://doi.org/10.24432/C5QG88.

[15] R. Kohavi, "Census Income," UCI Machine Learning Repository, 1996, DOI: https://doi.org/10.24432/C5GP7S.