Capturing and Querying Fine-grained Provenance of Preprocessing Pipelines in Data Science

Adriane Chapman
University of Southampton
Adriane.Chapman@soton.ac.uk

Paolo Missier
Newcastle University
paolo.missier@ncl.ac.uk

Giulia Simonelli
Università Roma Tre
giulia.simonelli@uniroma3.it

Riccardo Torlone
Università Roma Tre
riccardo.torlone@uniroma3.it

ABSTRACT

Data processing pipelines that are designed to clean, transform and alter data in preparation for learning predictive models, have an impact on those models' accuracy and performance, as well on other properties, such as model fairness. It is therefore important to provide developers with the means to gain an in-depth understanding of how the pipeline steps affect the data, from the raw input to training sets ready to be used for learning. While other efforts track creation and changes of pipelines of relational operators, in this work we analyze the typical operations of data preparation within a machine learning process, and provide infrastructure for generating very granular provenance records from it, at the level of individual elements within a dataset. Our contributions include: (i) the formal definition of a core set of preprocessing operators, and the definition of provenance patterns for each of them, and (ii) a prototype implementation of an application-level provenance capture library that works alongside Python. We report on provenance processing and storage overhead and scalability experiments, carried out over both real ML benchmark pipelines and over TCP-DI, and show how the resulting provenance can be used to answer a suite of provenance benchmark queries that underpin some of the developers' debugging questions, as expressed on the Data Science Stack Exchange.

PVLDB Reference Format:
Adriane Chapman, Paolo Missier, Giulia Simonelli, and Riccardo Torlone. Capturing and Querying Fine-grained Provenance of Preprocessing Pipelines in Data Science. PVLDB, 14(4): 507-520, 2021.
doi:10.14778/3436905.3436911

PVLDB Artifact Availability:
The source code, data, and other artifacts have been made available at https://github.com/GiuliaSim/DataProvenance.

1 INTRODUCTION

Dataset selection and data wrangling pipelines are integral to applied Data Science workflows. These typically culminate in the generation of predictive models through training, for a broad range of data types and application domains. A number of critical choices are made when these pipelines are designed, starting with the choice of which datasets to include or exclude, how these should be merged [18], and which transformations are required to produce a viable training set, given a choice of target learning algorithms. The main intended consequence of these transformation pipelines is to optimise the predictive performance and generalisation characteristics of the models that are derived from the ground data. There are however also un-intended consequences, as these transformations alter the representation of the domain that the learning algorithms generalise from, and they may remove or inadvertently introduce new bias in the data [11]. In turn, this may reflect on non-performance properties of the models, such as their *fairness*. Fairness, formally defined in terms of statistical properties of the model's predictions [29], broadly refers to the capability of a model to ensure that its predictions are not affected by an individual belonging to one of the groups defined by some sensitive attribute(s), such as sex, ethnicity, income band, etc.

Motivation. Fair models are also perceived as more trustworthy, an important feature at a time when machine learning models are increasingly used to support and complement human expert judgment, in areas where decisions have consequences on individuals as well as on businesses. Substantial recent research has produced techniques for explanation using: counterfactuals [27], local explanations [40], data [19] and meta-models [2]. While these techniques focus primarily on the model itself, relatively little work has been done into trying to explain models in terms of the transformations that occur *before* the data is used for learning. In this work, we enable the explanation on the effect of each transformation in a pre-processing pipeline on the data that is ultimately fed into a model. Specifically, we have developed a formal model and practical techniques for recording data derivations at the level of the atomic elements in the dataset, for a general class of data transformation operators. These derivations are a form of data provenance, and are expressed using the PROV data model [26], a standard and a widely adopted ontology. Data derivations form a corpus of graph-structured metadata that can be queried as a preliminary step to support user questions about model properties.

Problem scope. We consider transformations that apply to commonly used tabular or relational datasets and across application domains.[1] These steps have been systematically enumerated in

Proceedings of the VLDB Endowment, Vol. 14, No. 4 ISSN 2150-8097.
doi:10.14778/3436905.3436911

[1] However, we are not going to consider more specialised data pre-processing steps that may apply to data types such as video, audio, images, etc.

multiple reviews, see eg. [10, 23] and include, among others: feature selection, engineering of new features; imputation of missing values, or *listwise deletion* (excluding an entire record if data is missing on any variable for that record); downsampling or upsampling of data subsets in order to achieve better balance, typically on the class labels (for classification tasks) or on the distribution of the outcome variable (for regression tasks); outlier detection and removal; smoothing and normalisation; de-duplication, as well as steps that preserve the original information but are required by some algorithms, such as "one-hot" encoding of categorical variables. A complex pipeline may include some or all of these steps, and different techniques, algorithms, and choice of algorithm-specific parameters may be available for each of them. These are often grounded in established literature but variations can be created by data scientists to suit specific needs. We consider the space of all configured pipelines that can potentially be composed out of these operators, and we focus on relational datasets, which are arguably the most common data structures in popular analytics-friendly scripting languages like R, Spark, and Python (where they are called *dataframes*).

Overview of the approach. Firstly, we propose a formalisation and categorisation of a core set of these operators. Then, with each of those operators we associate a *provenance pattern* that describes the effect of the operator on the data at the appropriate level of detail, i.e., on individual dataframe elements, columns, rows, or collections of those. Effectively, the provenance patterns defined in this work for well-defined data science operators play a similar role to that of *provenance polynomials* [13], i.e., annotations that are associated to relational algebra operators to describe the fine-grained provenance of the result of relational as well as linear algebra operators [47, 48]. We then associate a provenance function $pf_{\mathbf{o}}()$ to each operator $\mathbf{o}$, which generates a provenance document $pf_{\mathbf{o}}(D)$ when a dataset D is processed using $\mathbf{o}$. The document is an instance of the pattern associated with $\mathbf{o}$. Provenance functions are implemented as part of a python module. Collecting all the provenance documents from each operator's execution results in a seamless, end-to-end provenance document that contains the detailed history of each dataset element in the final training set, including their creation (e.g. as a new derived feature), transformation (value imputation, for example) and possibly deletion (e.g., by feature selection, removal of null values).

Contributions. Our contributions can be summarised as follows.

- A formalisation and categorisation of a core set of operators for data reduction, augmentation, and transformation, where we show how common data pre-processing pipelines can be expressed as a composition of these operators, described in Section 4;
- An application-level provenance capture facility for Python, underpinned by the formal model, backed by a MongoDB database used as a provenance store, and discussed in Section 5;
- A validation of the query capabilities of the resulting granular provenance, using a collection of machine learning datasets using real data pre-processing pipelines, to show that using the resulting provenance we can successfully answer a suite of benchmark provenance queries. We then further tested these queries on real questions asked on the Data Science Stack Exchange[2] for machine learning pipelines in Section 6.1.
- An experimental analysis of the scalability properties of the facility. We show that while the overall provenance document can be arbitrarily large, it is created incrementally in a persistent data store, making the entire process scalable in the number of operators. We run extensive experiments on a synthetic TPC-DI dataset at multiple scales [37], and report on the time and space overhead of using the provenance functions in Section 6.2;

2 RELATED WORK

Established techniques and tools are available to generate provenance, and *provenance polynomials* in particular, through query instrumentation, however these operate in a relational database setting and assume that queries use relational operators [3, 12, 30]. While we show how some of the pipeline operators considered in this work map to relational algebra, this is not true for all of them, so we prefer to avoid techniques that are tightly linked to SQL or to first-order queries [20] as these would preclude other types of operators from being included in the future. We therefore consider this an unwise strategy in an "open world" of data pre-processing operators, consider e.g. *one-hot* and other kinds of categorical data encodings. We also note that tools that operate on a database backend, like *GProm* [30], *Smoke* [38] and older ones like *Post-it* [6] for provenance capture cannot be used in our setting. Interestingly, extensions to the polynomials approach have been proposed to describe the provenance of certain linear algebra operations, such as matrix decomposition and tensor-product construction [48]. While these can potentially be useful, it is a partially developed theory with limited and specialised applicability.

Moving beyond relational data provenance, capturing provenance within scripts is also not new, but efforts have mostly focused on the provenance of script definition, deployment, and execution [35]. Specifically, a number of tools are available to help developers build machine learning pipelines [1, 7, 43] or debug them [46], but these lack the ability to explain the provenance of a certain data item in the processed dataset. Others link provenance to explainability in a distributed machine learning setting [42] but without offering specific tools. Amazon identify that there are common and reusable components to a machine learning pipeline, but that there is no way to track the exploration of pipeline construction effectively, and call for metadata capture to support reasoning over pipeline design [41]. Vamsa [28] attempts to tackle some of these problems by gathering provenance of pipeline design, however the resulting provenance documents the invocation of specific ML libraries, by way of automated script analysis, rather than data derivations. Some systems are designed to help debug ML pipelines. BugDoc [21] looks at changes in a pre-processing pipeline that cause the models to fail, where high-level script and ordering is used to identify bad configurations. Others provide quality assurance frameworks [44] or embedded simulators to estimate fairness impacts of a particular pipeline [9]. Again, however, these are not

[2] https://datascience.stackexchange.com

geared for deep data introspection. Priu [47], helps users understand data changes, particularly deletions, that are used in regression models. Unfortunately, this work only tracks deletions, and not additions or updates to data.

Other tools record the execution of generic (python) scripts, but fail to capture detailed data provenance, like NoWorkflow [34, 36]. This has been combined with YesWorkflow [22, 49] which provides a workflow-like description of scripts, but again without a focus on data derivations.

A further class of tools instrument scripts that are specifically designed for Big Data processing frameworks: [16] (Hadoop), [14, 17, 39, 45] (Spark). They provide detailed information mostly for debugging purposes, but are restricted in their scope of applicability.

3 MODELS AND PROBLEM STATEMENT

3.1 Data model

The data collected for a ML problem is usually a single table or a single statistical data matrix in which columns represent specific features of a phenomenon being observed and rows are records of data for those features describing observations of the phenomenon. Therefore, we will refer to a generic notion of *dataset* that try to capture both formats and is similar in spirit to the concept of ordered relation [5].

A *(dataset) schema* S is an array of distinct names called *features*: $S = [\mathbf{a}_1, \ldots, \mathbf{a}_n]$. Each feature is associated with a domain of atomic values (such as numbers, strings, and timestamps). With a little abuse of notation, hereinafter we will compare schemas using set containment over their features. A *dataset* D over a schema $S = [\mathbf{a}_1, \ldots, \mathbf{a}_n]$ is an ordered collection of *rows* (or *records*) of the form: $i : (d_{i1}, \ldots, d_{in})$ where i is the unique *index* of the row and each element d_{ij} (for $1 \leq j \leq n$) is either a value in the domain of the feature $\mathbf{a}_j$ or the special symbol $\perp$, denoting a missing value. Indexes can be implemented in different ways (e.g., with RID annotations [39]). We only assume here that a row of any dataset can be uniquely identified.

Given a dataset D over a schema S we denote by $D_{i\mathbf{a}}$ the element for the feature $\mathbf{a}$ of S occurring the i-th row of D. We also denote by D_{i*} the i-th row of D, and by $D_{*\mathbf{a}}$ the column of D associated with the feature $\mathbf{a}$ of S.

Example 3.1. A possible dataset D over the schema $S = [\mathbf{CId}, \mathbf{Gender}, \mathbf{Age}, \mathbf{Zip}]$ is as follows:

	CId	Gender	Age	Zip
1	113	*F*	24	98567
2	241	*M*	28	$\perp$
3	375	*C*	$\perp$	32768
4	578	*F*	44	32768

$D_{*\mathbf{Age}}$ and D_{2*} denote the third column and the second row of D, respectively.

3.2 Data manipulation model

A general classification. As part of this work, we analyzed several packages that allow users to build data preprocessing pipelines. Table 1 contains an example overview of the available operators from the ML pipeline building tool Orange [8] and the popular SciKit packages [31]. As indicated in left hand side of the table, all of them can be classified in three main classes, according to the type of manipulation done on the input dataset D over a schema S:

- Data reductions: operations that reduce the size of D by eliminating rows (without changing S) or columns (changing S to $S' \subset S$) from D;
- Data augmentations: operations that increase the size of D by adding rows (without changing S) or columns (changing S to $S' \supset S$) to D;
- Data transformations: operations that, by applying suitable functions, transform (some of) the elements in D without changing its size or its schema (up to possible changes to the domain of the involved features of S).

In the rest of this subsection, we will introduce a number of basic operators of data manipulation over a dataset D with schema S that can be used to implement one of the above tasks, as indicated in the right hand side of Table 1. This approach is in line with the observation that most of the operations of current data exploration packages rely on a rather small subset of operators [32].

Data reductions. Two basic data reduction operators are defined over datasets. They are simple extensions of two well known relational operators.

π_C: the *(conditional) projection* of D on a set of features of S that satisfy a boolean condition C over S, denoted by $\pi_C(D)$, is the dataset obtained from D by including only the columns $D_{*\mathbf{a}}$ of D such that $\mathbf{a}$ is a feature of S that satisfy C;

σ_C: the *selection* of D with respect to a boolean condition C over S, denoted by $\sigma_C(D)$, is the dataset obtained from D by including the rows D_{i*} of D satisfying C.

The condition of both the projection and the selection operators can refer to the values in D, as shown in the following example that use an intuitive syntax for the condition.

Example 3.2. Consider the dataset D in Example 3.1. The result of the expression $\pi_{\{\textit{features without nulls}\}}(\sigma_{\mathbf{Age}<30}(D))$ is the following dataset:

	CId	Gender	Age
1	113	*F*	24
2	241	*M*	28

Data augmentations. Two basic data augmentation operators are defined over datasets. They allow the addition of columns and rows to a dataset, respectively.

$\alpha^{\rightarrow}_{f(X):Y}$: the *vertical augmentation* of D to Y using a function f over a subset of features $X = [\mathbf{a}_1 \ldots \mathbf{a}_k]$ of S is obtained by adding to D a new set of features $Y = [\mathbf{a}'_1 \ldots \mathbf{a}'_l]$ whose new values $d_{i\mathbf{a}'_1} \ldots d_{i\mathbf{a}'_l}$ for the i-th row are obtained by applying f to $d_{i\mathbf{a}_1} \ldots d_{i\mathbf{a}_k}$;

$\alpha^{\downarrow}_{X:f(Y)}$: 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 the features in X and then, for each group, by applying f to $\pi_Y(D)$ (extending the result to S with nulls if needed).

Example 3.3. Consider again the dataset D in Example 3.1 and the following functions: (i) f_1, which associates the string *young* to an age less than 25 and the string *adult* otherwise, and (ii) f_2, which

Table 1: Typical operations in ML pipelines of data preparation from Orange [7] and Scikit-Learn [31].

Orange3 Ex.	ScikitLearn Ex.	Category	Operator	Implementation
Feature Statistics	Feature_selection	Data reduction	Feature Selection	π_C
Select Data by Index	Dataframe op.		Instance Selection	σ_C
Select Columns	Feature_selection		Drop Columns	π_C
Select Rows	Dataframe op.		Drop Rows	σ_C
Data Sampler	Imbalanced-learn		Undersampling	σ_C
Impute	SimpleImputer	Data transformation	Imputation	$\tau_{f(X)}$
Apply Domain	FunctionTransformer		Value Transformation	$\tau_{f(X)}$
Edit Domain	Binarizer		Binarization	$\tau_{f(X)}$
Preprocess	Normalizer		Normalization	$\tau_{f(X)}$
Discretize	KBinDiscretizer		Discretization	$\tau_{f(X)}$
Feature Constructor	FunctionTransformer	Data augmentation	Space Transformation	$\pi_Z \circ \alpha^{\rightarrow}_{f(X):Y}$
Create Class	FunctionTransformer		Instance Generation	$\alpha^{\downarrow}_{X:f(Y)}$
Data Sampler	Imbalanced-learn		Oversampling	$\alpha^{\downarrow}_{X:f(X)}$
Corpus	Label Encoder		String Indexer	$\alpha^{\rightarrow}_{f(X):Y}$
Preprocess	OneHotEncoder		One-Hot Encoder	$\alpha^{\rightarrow}_{f(X):Y}$

computes the average of a set of numbers. Then, the expression $\alpha^{\rightarrow}_{f_1(\mathbf{Age}):\mathbf{ageRange}}(D)$ produces the following dataset:

	CId	Gender	Age	Zip	ageRange
1	113	*F*	24	98567	*young*
2	241	*M*	28	$\perp$	*adult*
3	375	*C*	$\perp$	32768	$\perp$
4	578	*F*	44	32768	*adult*

whereas $E_2 = \alpha^{\downarrow}_{\mathbf{Gender}:f_2(\mathbf{Age})}(D)$ the dataset:

	CId	Gender	Age	Zip
1	113	*F*	24	98567
2	241	*M*	28	$\perp$
3	375	*C*	$\perp$	32768
4	578	*F*	44	32768
5	$\perp$	*F*	34	$\perp$
6	$\perp$	*M*	28	$\perp$

Note that brand-new data can be added to a dataset using an horizontal augmentation in which $X = \emptyset$, $Y = S$, and f denotes the procedure for adding records (e.g., by asking them to the user). Note also that the horizontal augmentation allows us to combine, in the same dataset, entities at different levels of granularity, a feature that can be very useful to a data scientist (e.g., to compute, in the example above, the mean deviation).

Data transformation One basic data transformation operator is defined over datasets:

$\tau_{f(X)}$: the *transformation* of a set of features X of D using a function f is obtained by substituting each value $d_{i\mathbf{a}}$ with $f(d_{*\mathbf{a}})$, for each feature $\mathbf{a}$ occurring in X.

Example 3.4. Let D be the dataset in Example 3.1 and f be an imputation function that associates to the $\perp$'s occurring in a feature $\mathbf{a}$ the most frequent value occurring in $D_{*\mathbf{a}}$. Then, the result of the expression $\tau_{f(\mathbf{Zip})}(D)$ is the following dataset:

	CId	Gender	Age	Zip
1	113	*F*	24	98567
2	241	*M*	28	32768
3	375	*C*	$\perp$	32768
4	578	*F*	44	32768

We note that the data manipulation model presented here has some similarity with the Dataframe algebra [32]. The main difference is that we have focused on a restricted set of core operators (with some of those in [32] missing and others combined in one) with the specific goal of providing a solid basis to an effective technique for capturing data provenance of classical preprocessing operators. We point out that our algebra can be easily extended to include operators implementing other ETL/ELT-like transformations, such as join, intersection, and union, whose fine-grained provenance capture have been described elsewhere [50].

3.3 Data provenance model

The purpose of data provenance is to extract relatively simple explanations for the existence (or the absence) of some piece of data in the result of complex data manipulations. Along this line, we adopt as the provenance model a subset of the PROV model [25] from the W3C, a widely adopted ontology that formalises the notion of *provenance document* and which admits RDF and other serialisation formats to facilitate interoperability. This model can be graphically described as shown in Figure 1.

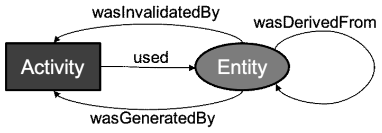


Figure 1: The core W3C PROV model.

In PROV an entity represents an element d of a dataset D and is uniquely identified by D and the coordinates of d in D (i.e., the corresponding row index and feature). 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: (i) d *wasGeneratedBy* $\mathbf{o}$, and (ii) d *wasDerivedFrom* a set of elements in D. In addition, we represent: (iii) all the elements d of D such that d was *used* by $\mathbf{o}$ and (iv) all the elements d of D such that d *wasInvalidatedBy* (i.e., deleted by) $\mathbf{o}$ (if any). Note that in PROV derivation implies usage, but the inverse is not true and this is why this notation is not redundant.

Example 3.5. Let E be the first expression in Example 3.3 and $D' = E(D)$. A fragment of the data provenance generated by this operation is reported in Figure 2.

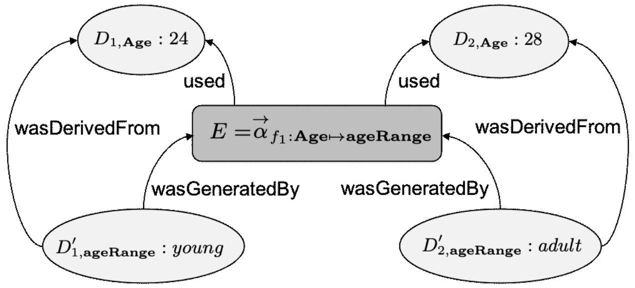

$D_{1,\mathbf{Age}} : 24$
$D_{2,\mathbf{Age}} : 28$
used
used
wasDerivedFrom
$E = \vec{\alpha}_{f_1:\mathbf{Age}\mapsto\mathbf{ageRange}}$
wasDerivedFrom
wasGeneratedBy
wasGeneratedBy
$D'_{1,\mathbf{ageRange}} : young$
$D'_{2,\mathbf{ageRange}} : adult$

Figure 2: A fragment of provenance data for the operation in Example 3.5.

3.4 Problem Statement

We consider compositions of the operators introduced in Section 3.2 into *pipelines* that take input D and produce D', denoted $D' = E(D)$. Note that although in principle any combination is possible, in practice there are limitations, because some operators may alter the dataset schema.

The outcome, accuracy and performance of the final model are dependent upon the final dataset produced by $E(D)$. As the data scientist attempts to create a performant model, she may wish to inspect and understand exactly what happened within each transformation of the dataset within the pipeline. Unfortunately, as these pipelines become complex, they become more difficult to understand and debug. Table 2 contains a set of use cases from the Data Science Stack Exchange (DSSE) of users attempting to understand what is happening within the processes and data in a machine learning pipeline. These use cases were gathered via the following methodology: DSSE was searched for all questions using the Orange framework; DSSE questions were included if they were about pipeline construction; exclusions included questions on specific operators, how to use the Orange GUI, etc. In Table 3, we describe the provenance required for a developer to identify the problems in their machine learning pipeline.

Thus, the problem within this work is to: a) define the set of operations for data manipulation available within a pipeline; b) establish a set of provenance patterns that can be used to reason over and capture the provenance of these operations over the data; c) show that our approach can support typical provenance queries in an effective and scalable way.

4 PRE-PROCESSING OPERATORS

In this section we illustrate a number of common pre-processing operators that are often used in data preparation workflows showing how they can be suitably expressed as composition of the basic operators introduced in Section 3.2

4.1 Data Reductions

Feature Selection. This operation consists of selecting a set of relevant features from a given dataset and dropping the others, which are either redundant or irrelevant for the goal of the learning process.

Feature selection over a dataset D with a schema S can be expressed by means of a simple pipeline involving only the projection operator with a condition that selects the set of features $I \subset S$ of interest:

$$FS(D) = \pi_C(D)$$

where $C = \{\mathbf{a} \in I\}$.

A special case of feature selection is an operation that drops columns with a value rate of missing values higher than a threshold t. In this case, the condition of the projection operator is more involved as it requires introspection of the dataset:

$$C = \{\mathbf{a} \in S \mid \mathsf{count}(D_{i\mathbf{a}} = \bot, 1 \leq i \leq n) < t\}.$$

Instance Selection. The aim of this operation is to reduce the original dataset to a manageable volume by removing noisy instances with the goal of improving the accuracy (and efficiency) of classification problems.

Also in this case, instance selection over a dataset D with a schema S can be expressed by means of a simple pipeline involving only the selection operator with a condition that identifies the set of relevant rows of D by means of a predicate p: $IS(D) = \sigma_C(D)$ where $C = \{D_{i*} \in S \mid p(D_{i*})\}$.

Similar to feature selection, a relevant case of instance selection drops rows with a value rate of missing values higher than a threshold t. In this case,

$$C = \{D_{i*} \in D \mid \mathsf{count}(D_{ij} = \bot, 1 \leq j \leq m) < t\}$$

4.2 Data Transformations

By data transformation we mean any operation on a given dataset that modifies its values with the goal of improving the quality of D and/or making more effective the process of information extraction from D. In general, any kind of data transformation can be expressed by means of a pipeline involving the data transformation operator: $DT(D) = \tau_{f(X)}(D)$, where f can be any scalar function that associates with one or more values from the domain of the features X of S a value. Several cases are common in preprocessing pipelines, as illustrated in the following.

Data repair. It is the process of replacing inconsistent data items with new values. In this case, f is a simple function that converts values and the data transformation possibly operates on the whole dataset.

Binarization. It is the process of converting numerical features to binary features. For instance, if a value for a given feature is greater than a threshold it is changed a 1, if not to 0.

Normalization. It is a scaling technique that transforms all the values of a feature so that they fall in a smaller range, such as from 0 to 1. There are many normalization techniques, such as Min-Max normalization, Z-score normalization and Decimal scaling normalization. This operation operates on a single feature at a time

Table 2: Issues identified in Data Science Stack Exchange (DSSE) for Machine Learning pipelines.

Id	Data Science Stack Exchange Use Cases
UC1	When applying the ’Predictions’ widget on the same training dataset, the results (i.e. probability scores) are different: https://datascience.stackexchange.com/questions/32382/orange-predictions-widget-on-same-data-gives-different-results
UC2	Differences in the predictions and corresponding goodness-of-fit R2 metric for the linear regression model on Orange and scikit-learn: https://datascience.stackexchange.com/questions/32678/orange-linear-regression-and-scikit-learn-linear-regression-gives-different-resu
UC3	After performing image classification using an ML model, prediction probabilities are constant on test images https://datascience.stackexchange.com/questions/38320/orange3-image-classification
UC4	From a constructed workflow using image classification (add on widgets), ascertain whether the workflow performs ’transfer learning’: https://datascience.stackexchange.com/questions/19240/using-orange3-to-predict-image-class
UC5	Application of the ’Test and Score’ and ’Predictions’ widget on the same data utilising the same ML model; produces differing results: https://datascience.stackexchange.com/questions/20572/why-orange-predictions-and-test-score-produce-different-results-on-the-sam
UC6	When applying the ’Impute’ widget during preprocessing on train/test dataset, the same values are predicted for all rows: https://datascience.stackexchange.com//questions/15264/orange-3-same-prediction-for-all-of-my-data-when-using-impute-widget
UC7	Inaccuracy in the prediction of target variable using k-NN and linear regression ML models in an Orange workflow: https://datascience.stackexchange.com/questions/36537/how-to-properly-predict-date-using-orange-3
UC8	Disproportionate allocation of labels after performing data analysis and modelling (inaccurate classification accuracy): https://datascience.stackexchange.com/questions/37471/dataset-with-disproportionately-more-of-a-single-label-than-any-other

Table 3: Provenance queries of interest to a data scientist designing a pipeline of preprocessing operations.

Id	Provenance Query	Input	Output	Use Case
1	All Transformations	D	Set of operations applied to D and the features they affect.	UC1
2	Why-provenance	$d_{i\mathbf{a}}$	The input data that influenced $d_{i\mathbf{a}}$.	UC2
3	How-provenance	$d_{i\mathbf{a}}$	The input data and the operations that created $d_{i\mathbf{a}}$.	UC3, UC4, UC5
4	Dataset-level Feature Operation	$D_{*\mathbf{a}}$	Set of operations that were applied to feature $\mathbf{a}$.	UC6
5	Record Operation	D_{i*}	Set of operations that were applied to record D_{i*}.	
6	Item-level Feature Operation	$d_{i\mathbf{a}}$	Set of operations that were applied to $d_{i\mathbf{a}}$.	
7	Set of Invalidations	D	Set of all D_{i*}, $D_{*\mathbf{a}}$, $d_{i\mathbf{a}}$ that were deleted.	UC7
8	Feature Invalidation	$D, \mathbf{a}$	The operation that deleted the feature $D_{*\mathbf{a}}$.	
9	Record Invalidation	D, i	The operation that deleted the record D_{i*}.	
10	Item Invalidation	$D, i, \mathbf{a}$	The operation that deleted the item $d_{i\mathbf{a}}$.	
11	Impact on Feature Spread	D	The change in feature spread of all operations applied to a feature of D.	UC6, UC8
12	Impact on Dataset Spread	D	The change in dataset spread of all operations applied to D.	

Discretization. It consists of converting or partitioning continuous features into discrete or nominal features. It performs a value transformation from categorical to numerical data.

Imputation. It is the process of replacing missing data (nulls in our data model) with valid data using a variety of statistical approaches that aim at identifying the values with the maximum likelihood.

4.3 Data augmentations

Space Transformation. This operation takes a set of features of an existing dataset and generates from them a new set of features by combining the corresponding values. Usually, the goal is to represent (a subset of) the original set of features in terms of others in order to increase the quality of learning.

The application of this operation to a dataset D over a schema S can be expressed by means of a expression involving a vertical augmentation that operates on a subset X of the features in S and produce a new set of features Y, followed by a projection operator that eliminates the features in X, thus maintaining those in $Z = (S \cup Y) - X$:

$$ST(D) = \pi_{\{features\ in\ Z\}}(\alpha^{\rightarrow}_{f(X):Y}(D))$$

Instance Generation: This process allows us to fill regions in the domain of the problem, which do not have representative examples in original data, or to summarize large amounts of instances in fewer examples. Instance generation methods are often called prototype generation methods, as the artificial examples created tend to act as a representative of a region or of a subset of the original instances.

The application of this operation to a dataset D over a schema S can be expressed by means of a expression involving a horizontal augmentation that, if needed, groups over on a subset X of the features in S and then apply a summary function f over another subset of S:

$$IG(D) = \alpha^{\downarrow}_{X:f(Y)}(D).$$

This operation can be preceded by a data reduction operator (a projection or a selection) to isolate the portion of the original dataset on which we intend to operate.

String Indexer. This operators encodes a feature involving strings into a feature of string indices. The indices are in $[0, numLabels)$. It is a special case of Space transformation.

One-Hot Encoder. This operation maps a feature involving strings to a set of boolean features. Specifically, it creates one column for each possible value occurring in the feature. Each new feature gets a 1 if the row contained that value and a 0 if not. It is a special case of space transformation.

5 CAPTURING PROVENANCE

In order to capture the provenance of a pipeline p of a sequence of preprocessing operations $\mathbf{o}_1, \ldots, \mathbf{o}_n$, we associate a provenance-generating function (*p-gen*) with each operation $\mathbf{o}_k$ occurring in p. Each such function generates a collection of provenance data whenever a dataset is processed using $\mathbf{o}_k$, which describes the effect of $\mathbf{o}_k$ on the data at the appropriate level of detail.

In concordance with the provenance model presented in Section 3.3, for each element d_{ij} (an entity in the PROV model) of a dataset D produced during the execution of p, we represent its coordinates (i.e., the row index i and feature j in D) and a progressive number k denoting the fact that d_{ij} is in the result of the k-th operation in p. For each operation $\mathbf{o}_k$ (an *activity* in the PROV model) in p, we represent the operator(s) illustrated in Section 3.2 that implement(s) $\mathbf{o}_k$ and the list of the features on which $\mathbf{o}_k$ operates.

5.1 Provenance templates

We now present the provenance-generating (*p-gen*) functions that are invoked alongside the execution of one of the operators $\mathbf{o}$ on D to obtain D'. As all specific operators in Section 4 are defined in terms of our five core pipeline operators, it is enough to define a p-gen function for each of these operators. To recall, these are: (i) data reduction: $D' = \pi_C(D)$, $D' = \sigma_C(D)$; (ii) Data augmentations: $\alpha^{\rightarrow}_{f(X):Y}$, $\alpha^{\downarrow}_{X:f(Y)}$; and (iii) Data transformations: $\tau_{f(X)}$.

Each p-gen function takes inputs D, D' (the inputs and outputs of their associated operator) along with a description of the operator itself, and produces a PROV document that describes the transformation produced by the operator on each element of D. The output PROV document is obtained by instantiating an appropriate provenance template [24], which is designed to capture the transformation at the most granular level, i.e., at the level of individual elements of D, or its rows or columns, as appropriate.

In general, the template will have a *used* set of entities, which refer to the subset of data items in D which are effectively used by $\mathbf{o}$, and a *generated* set of new entities, corresponding to new elements in D'. For projection and selection, it will have an *invalidated* set of entities instead, as these operators remove data from D.

Take for example the case of Vertical Augmentation (VA): $\alpha^{\rightarrow}_{f_1(\mathbf{Age}):\mathbf{ageRange}}(D)$ which we used in Example 3.3, where attribute **Age** is binarised into *{young, adult}* based on a pre-defined cutoff, defined as part of $f()$. The p-gen function for VA will produce a collection of small PROV documents, one for each input-output pair $\langle D_{i,\mathbf{Age}}, D'_{i,\mathbf{AgeRange}} \rangle$ as shown in the example. As these documents all share the same structure, we specify p-gen by giving two elements. First, a single PROV template for (VA) as shown in Figure 3, where we use the generic attribute names X, Y to indicate the old and new feature names, as per the operator's general definition in Section 3.2. Notice that, since we want to express that new data elements after transformation are indeed derived from corresponding old elements, we also add an explicit *wasDerivedFrom* relationship in addition to *used* and *wasGeneratedBy*.

A template is simply a PROV document where: (i) variables, indicated by the namespace **var:**, are used as placeholders for values and (ii) a set of rules is used to specify how the "used" and the "generated" sides of the template are repeatedly instantiated, by binding the variables to each of the data items involved in the transformation. We refer to each instantiated template produced by a p-gen function as a *provlet.*

The VA example is particularly simple, as the transformation between D and D' is 1:1 and thus a new PROV document instance is created for each value of column $D_{*,\mathbf{Age}}$. Using a list comprehension notation, the bindings for the variables used in the template in Figure 3 are defined as:

$$[\langle F = \mathbf{Age}, I = i, V = D_{i,\mathbf{Age}},$$
$$F' = \mathbf{AgeRange}, J = i, V' = f(D_{i,\mathbf{Age}}) \rangle | i : 1 \ldots n]$$

These are the new entities for the newly created data elements in the new column $D_{*,\mathbf{AgeRange}} \in \{young, adult\}$. One of the n PROV documents for this specific example is shown in Figure 3.

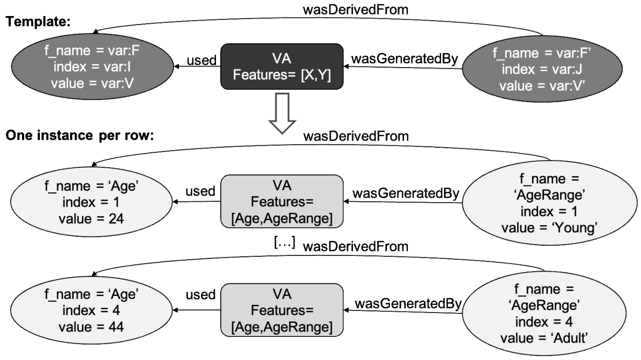


Figure 3: Example of PROV template for Vertical Augmentation and corresponding instances.

5.2 Template binding rules

Generalising, we define templates for each of the five core operators, shown in Figure 4 and the corresponding binding generators for *used*, *generated*, and *invalidated* sets of entities.

Note that we do not need to create a new provenance record for all entities in any given output dataset. If $f(D)$ does not change d_{ij}, then no provenance record needs to be generated. If $f(D)$ throws away elements only invalidation records are required. Only in the case where a new entity is generated, i.e. when $f(D)$ creates a new or updated value in d_{ij}, is a provenance record required. In other words, we only require provenance statements that capture the *delta* for elements in the dataframe.

Data reduction, selection: Data reduction *invalidates* existing entities. For selection: $D' = \sigma_C(D)$, the bindings specify that an entire row i is invalidated whenever condition C is False when evaluated on that row. This affects all features $X \in S$:

$$[\langle F = X, I = i \rangle | X \in S, i : 1 \ldots n, C(D_{i,*} = \text{False})]$$

A *wasInvalidatedBy* relationship is established between each of these entities and a single *Activity*, representing the selection.

Data reduction, projection: Conditional projection $D' = \pi_C(D)$ invalidates all elements in column $X \in S$ whenever C returns True when evaluated on elements of X:

$$[\langle F = X, I = i \rangle | X \in S, i : 1 \ldots n, C(D_{*,X} = True)]$$

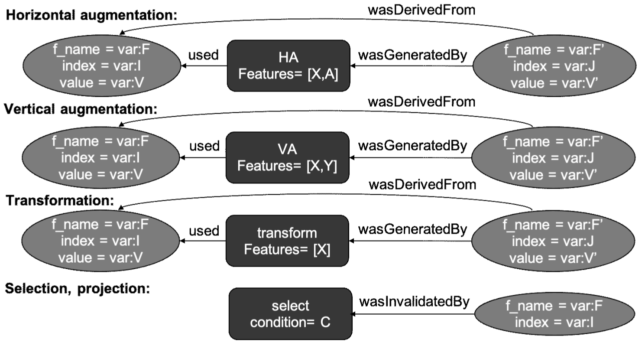


Figure 4: PROV templates used by the p-gen functions.

Similar to selection, here too a *wasInvalidatedBy* relationship is established between each of these entities and a single *Activity*, representing the projection.

Vertical augmentation: $\alpha^{\rightarrow}_{f(X):Y)}$ takes a set $X \subset S$ of features and adds a new set Y of features, $Y \cap S = \emptyset$ to D' as shown in Ex. 3.3. The provenance consists of n PROV documents, one for each row i of D, and in each such document entities for $D_{i,X_m}, X_m \in X$ are used to generate entities for the new features $Y_h \in Y$. Thus, the bindings are defined as follows:

For $i : 1 \ldots n$:
used entities:$[\langle F = X_m, I = i, V = D_{i,X_m}\rangle | X_m \in X]$
generated entities:$[\langle F' = Y_h, J = i, v = f(D_{i,X})\rangle | Y_h \in Y]$

These entities are then connected to a single *Activity*, as shown in Figure 4 and in the examples (Figg. 3, 6), using *Used* and *wasGeneratedBy* relationship. For each pair of *used*, *generated* entities having the same index on each side (i.e., where **var:I** = **var:J** after template instantiation), a *wasDerivedFrom* relationship is also added, to assert a stronger relationship (derivation occurs through the Activity that connects the entities).

Horizontal augmentation: The $\alpha^{\downarrow}_{X:f(A)}$ operator groups records according to columns $X \subset S$, producing a list $G = [g_1 \ldots g_h]$ of h groups. Then for each $g_i \in G$ it computes $f(A)$ from the records in the group, producing a new record containing the aggregated value in column A, the values that define the group in each column $X_m \in X$, which we denote $val(X_m, g_i)$, and *null* in all other columns (see Ex. 3.3 in Section 3.2). Thus, the operator produces h records, and let $rows(G) = [n+1, n+2, \ldots n+h]$ denote their new row indexes in the dataframe.

The corresponding provenance template is the same as for Vertical Augmentation (Figure 4), however the bindings differ, and they are defined as follows.

Used entities. For each $g_i \in G$, let $rows(g_i)$ denote the set of row indexes for records in g_i. The bindings associated with g_i are:

$$[\langle F = A, I = i, V = D_{i,A}\rangle | i \in rows(g_i)].$$

Generated entities. For each g_i, the new record with index $n + i$ is represented by a set of generated entities, with bindings:

$$[\langle F' = A, I = i, V = f(D_{rows(g),A})\rangle]$$
$$[\langle F' = Y, I = i, V = null\rangle | Y \in S \setminus X, Y \neq A]$$
$$[\langle F' = X_m, I = i, V = val(X_m, g_i)\rangle | X_m \in X]$$

Like for Vertical Augmentation, a single *Activity* is also created, which connects *Used* and *Generated* entities through *Used* and *wasGeneratedBy* relationships between each pairs of entities representing data in the same column, that is, where **var:F** = **var:F'**, and an additional *wasDerivedFrom* relationship is also added.

Data transformation: $\tau_{f(X)}$ takes features $X \subset S$ and computes derived values, which are used to update elements of D, but without generating new elements. The bindings reflect such in-place update, but as the new value for each element is defined by $f()$, we assume for simplicity that *all* values are updated, although in reality some will stay the same, as shown for instance in Ex. 3.4 (imputation). The resulting bindings reflect this many-many relationship, where (potentially) all values in a column $X_m \in X$ are used to update (potentially) all values in that same column (and this applies to each column). Thus, the provenance document consists of $|X|$ provlets, one for each K_k, with bindings defined as follows. *Used* entities:

$$[\langle F = X_m, V = D_{i,X_m}, I = i\rangle | i : 1 \ldots n]$$

Generated entities:

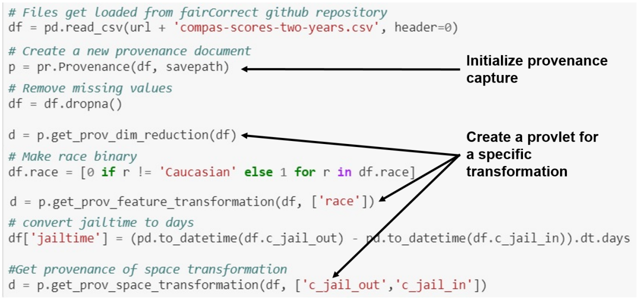
$$[\langle F' = X_m, V' = f(D_{*,X_m}), J = i\rangle | i : 1 \ldots n]$$

Used and *wasGeneratedBy* relationships, mediated by an *Activity*, are created between each *Generated* entity and all of the *Used* entities having the same X_m, along with the corresponding *wasDerivedFrom* relationships.

5.3 Code instrumentation

Approaches for automated provenance capture, such as by using the python call stack as in NoWorkflow [36], or capturing model intermediates as in Mistique [46], have been mentioned in Section 2. These, however, fail to capture data provenance at the level of the individual element within a dataframe. To accomplish this, in this initial prototype, we opted for explicit and analyst-controlled instrumentation at the script level. We have packaged the implementation of the p-gen functions described in the previous section as a python library that analysts can add to their code where provenace capture is desired. Figure 5 shows an example. Note also that it may be possible to automate function call injection, at least in part, by leveraging mature code annotation tools like YesWorkflow [22], where formally written code comments are interpreted to generate derived representations of the scripts (ie as a workflow). This mechanism can be used to declaratively specify directives into the code where provenance functions need to appear.

While this does not completely eliminate the need for manual intervention, this is now a simple comment/ annotation effort (which can be driven by a smart UI) rather than requiring additional programming.



```
# Files get loaded from fairCorrect github repository
df = pd.read_csv(url + 'compas-scores-two-years.csv', header=0)

# Create a new provenance document
p = pr.Provenance(df, savepath)

# Remove missing values
df = df.dropna()

d = p.get_prov_dim_reduction(df)

# Make race binary
df.race = [0 if r != 'Caucasian' else 1 for r in df.race]

 d = p.get_prov_feature_transformation(df, ['race'])

# convert jailtime to days
df['jailtime'] = (pd.to_datetime(df.c_jail_out) - pd.to_datetime(df.c_jail_in)).dt.days

#Get provenance of space transformation
d = p.get_prov_space_transformation(df, ['c_jail_out','c_jail_in'])
```

Initialize provenance capture
Create a provlet for a specific transformation

Figure 5: Capture calls embedded in pipeline code.

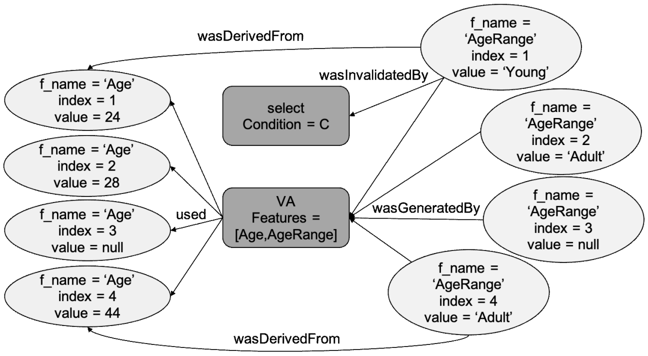

wasDerivedFrom
f_name = 'AgeRange' index = 1 value = 'Young'
wasInvalidatedBy
f_name = 'Age' index = 1 value = 24
select Condition = C
f_name = 'AgeRange' index = 2 value = 'Adult'
f_name = 'Age' index = 2 value = 28
VA Features = [Age,AgeRange]
wasGeneratedBy
f_name = 'AgeRange' index = 3 value = null
f_name = 'Age' index = 3 value = null
used
f_name = 'Age' index = 4 value = 44
f_name = 'AgeRange' index = 4 value = 'Adult'
wasDerivedFrom

Figure 6: Provlet composition.

5.4 Generating provenance documents

A complete provenance document is produced by combining the collection of provlets that results from calling p-gen functions. Specifically, one provlet is generated for every transformation and every element in the dataframe that are affected by that transformation. The document is represented by such collection of provlets, where entity identifiers match across provlets, and never needs to be fully materialised, as explained shortly.

To illustrate how provlets are generated, consider the following pipeline: $\sigma_C(\alpha_{\overrightarrow{f_1}(\mathbf{Age}):\mathbf{ageRange}}(D))$ where $C = \{\mathbf{AgeRange} \neq \text{'Young'}\}$ and D is the dataset of Ex. 3.3. The corresponding provenance document is represented in Figure 6.

Applying vertical augmentation produces one provlet for each record in the input dataframe, showing the derivation from **Age** to **AgeRange**. The second step, selecting records for 'not young' people, produces the new set of provlets on the right, to indicate invalidation of the first record, as per the template at the bottom of Figure 4. Note that the "used" side on the left refers to existing entities, which are created either into the pipeline from the input dataset, or by an upstream data generation operator.

Provlet composition requires looking up the set of entities already produced, whenever a new provlet is added to the document. One simple way to accomplish this is by eagerly keeping the entire document in memory, along with an index for all entities, as well as a mapping between each entity and the corresponding data element

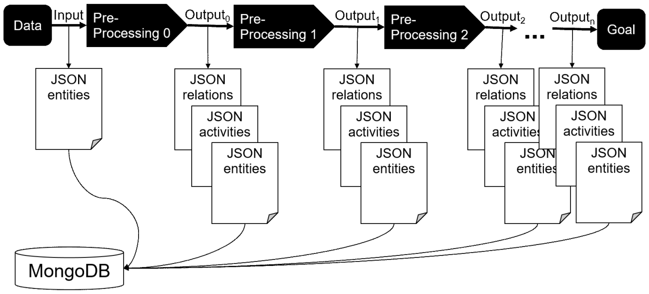


Figure 7: Pipeline and provenance architecture.

it represents. While this can be accomplished using readily available Python PROV libraries [15], this approach does not scale well to the volume of entities required to represent large dataframes, when more than a handful of transformation operators are involved. Instead, we have built a bespoke architecture as shown in Figure 7 that allows lazy provenance composition. Each p-gen function generates a set of provlets, one for each element in the dataframe (in the worst case), constructs a partial document, and stores it to a persistent MongoDB back end. This allows the provenance to be collected quickly at execution of each script, and be assembled later, minimizing execution dependencies and possible bottlenecks during the actual execution of the pipeline.

MongoDB was specifically chosen, instead of other pre-existing provenance storage systems, because it is a tool actively used by the data science community when building machine learning pipelines; i.e. the community for which we are providing provenance is conversant in the tool, and comfortable issuing provenance queries in it. Moreover, MongoDB provides a flexible data model, which allows us to store complex provenance data in a natural way, and is scalable, facilitating distributed provenance capture in the future.

Concretely, each *p-gen* function creates a provenance object containing all provlets, and an input json file representing the input dataframe. The MongoDB back end is strucured into folders, one for each p-gen function (i.e., one for each operator). Each folder contains three json data structures, containing for an array of entities, an array of activities, and an array of relations. These objects are only composed into a complete provenance document *at query time*, to provide a complete trace of the data and processes used within the pipeline. By capturing provlets from each p-gen function, it is possible to compose these provlets into a complete graph. This graph can be traversed as a bipartite graph for any d_{ij}. The process for composing provlets, and tracing the influence (either direct of *indirect*) of data and operations on d_{ij} is summarized in Algorithm 1 for returning "why provenance" [4]. Note that this tracing function is similar to functions for provenance analytics in [14].

6 EVALUATION

All experiments were performed on a server with 32 Intel(R) Xeon(R) CPU E5-2620 v4 (2.10GHz).

6.1 Analysis with Real World Pipelines

Datasets. In Table 3 we show classic provenance queries in terms of data input and output. In order to evaluate if we can answer

Algorithm 1: Why-provenance obtaining the inputs that directly or indirectly influenced a single element

```
Input: An element $d_{ij}$
Output: Print out all the inputs that influenced $d_{ij}$
activities ← findActivities($d_{ij}$)
while activities ≠ ∅ do
    entities ← findUsedEntities(activities)
    activities ← findActivities(entities)
return entities
```

these queries, we have captured data provenance in three real world pipelines involving different types of preprocessing steps. The datasets are described in Table 4.

Table 4: Datasets used for evaluation.

	German Credit	Compas Score	Census
Records	1000	7214	32561
Features	21	53	15
# Operations	4	7	5
Output Records	1000	6907	32561
Output Features	60	8	104
Provenance Entities	85000	349970	3874264
Provenance Activities	26	7	20
Provenance Relations	255000	451412	9703396

The goal of the *German Credit* pipeline is to predict whether an individual is a good lending candidate. The goal of the *Compas Score* pipeline is to predict the recidivism risk of an individual. The goal of the *Census* pipeline is to predict whether annual income for an individual exceeds $50K. Table 5 shows the preprocessing steps for each of these machine learning pipelines.

Basic provenance instrumentation (BP). In order to compare our work against the manner typically used by data scientists outside of a workflow management system, we also instrument coarse-grained provenance capture within scripts. This method requires the script owner to embed provenance capture calls within her scripts. We place basic calls within the scripts by hand, using the standard PROV libraries [15]. This mechanism is similar to that used in YesWorkflow [22, 33, 49]. This approach will give fairly coarse-grained provenance; we refer to this method as Basic Provenance (BP).

Fine-grained provenance instrumentation (FP). We explored two distinct approaches for capturing FP provenance: (i) using classic, eager, capture libraries [15] that create a single monolithic provenance document during execution of the entire pipeline, and (ii) using the lazy, provlet, provenance composition strategy described in Section 5.

During initial experimentation, it became apparent that the monolithic approach is not performant in even the most basic machine learning pipelines because of the size of provenance generated via each operator.

Table 5: The processing operations for the machine learning pipelines used in the evaluation.

German Credit Pipeline	
Id	**Description**
Op A0	Value transformation of 13 distinct columns from codes to interpretable words.
Op A1	Generation of two new columns from the column *personal_status.*
Op A2	The column *personal_status* was deleted.
Op A3	11 categorical column were OneHot encoded.
Compas Score Pipeline	
Id	**Description**
Op B0	Selection of 9 relevant columns.
Op B1	Missing values were deleted.
Op B2	The column *race* was binarized.
Op B3	Value transformation of the *label* column for consistency.
Op B4	Conversion of *c_jail_in* and *c_jail_out* columns to days.
Op B5	Drop *jail_in* and *jail_out* dates.
Op B6	Value transformation of column *c_charge_degree.*
Census Pipeline	
Id	**Description**
Op C0	Remove whitespace from 9 columns.
Op C1	Replace '?' charater for NaN value.
Op C2	7 categorical columns were OneHot encoded.
Op C3	Two columns were binarized.
Op C4	*fnlwgt* column was deleted.

Table 6: Analysis of which provenance queries from Table 3 are answerable via each capture method.

Id	Provenance Query	BP	FP
1	All Transformations	✓	✓
2	Why-provenance	-	✓
3	How-provenance	-	✓
4	Dataset-level Feature Operation	✓	✓
5	Record Operation	-	✓
6	Item-level Feature Operation	-	✓
7	Set of Invalidations	✓	✓
8	Feature Invalidation	✓	✓
9	Record Invalidation	-	✓
10	Item Invalidation	-	✓
11	Impact on Feature Spread	-	✓
12	Impact on Dataset Spread	-	✓

Capturing provenance. How provenance is captured changes how much information about the underlying processes and data items can be gathered. Because of this, some provenance queries become unanswerable.

Table 6 contains the analysis of which provenance queries, as defined in Table 3 can be answered with each method described in this work. For each of the general provenance queries expressed in Table 3 that wish for the provenance of a single data item, that data item was randomly selected from the output dataset, and the queries run for that instance. Each query was performed three times, as discussed further below. As expected, the basic provenance captured by embedding provenance capture calls at the script level (BP) does not lead to useful provenance in many cases.

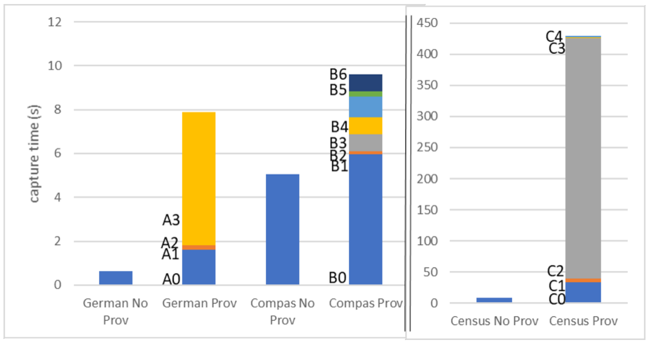


Figure 8: Comparison of cumulative provenance capture times, broken down by individual operator.

Because the BP approach is too coarse-grained to capture useful provenance that can answer a wide range of possible provenance queries, as shown in Table 6 and derived from the provenance queries expressed in Table 3, it is imperative that the fine-grained approach is performant, and does not impact overall system runtime significantly. On average, each pipeline takes the following amount of time to run without any provenance capture: German Credit is 648,56 ms; COMPAS is 5045 ms; Census is 8289,12 ms. Figure 8 show the impact of adding provenance capture to a pipeline. As expected, provenance capture adds time to any pipeline execution. However, there are some operations that add an inordinate amount of time. In the Census pipeline there is 1 operation (C2) that adds 386s. This is the One Hot encoding of 7 different columns, which generates 90 new features (from 15 to 105 columns). The number of records remains unchanged (32,561), so there will be 32561*90 new provenance entities. Other operations that take time include B0, which selects 9 columns of data, removes 44 features, and generates 7214*44 provenance records and A3, another One Hot encoding of 11 different columns, generating 38 new features, and thus 1000x38 new provenance records. For operations such as A2, B2, B5, C3, C4, in which the number of provenance records are small, and “cover" entire column manipulations, the capture times are very small. The total size of the provenance captured for each pipeline is: German Credit 75 MB; Compas Score 199 MB; Census 3.8 GB.

Querying Provenance. Provenance would be useless without the ability to query over it. We instantiate queries representative of all of queries expressed in Table 3. Each query was run three times and the resulting time is the average of the three runs. Queries 2 through 6 are related to a single item d_{ij}, record D_{i*} or feature D_{*j}, while the others are related to the entire dataset. For these queries, a data item, record or feature is chosen randomly from the output dataset each time the query is run.

As shown in Figure 9, when the dataset is as small as German pipeline, the query execution time is low. As the dataset increases, query execution times increase proportionally. Notice that Provenance queries that look for invalidations (Queries 7 and 10) and Feature Spread (Queries 11 and 12) are very time consuming. These provenance queries require processing of information over the entire dataset. On average, across all query types, the processing of a single provenance entry costs $3.07E^{-06}$.

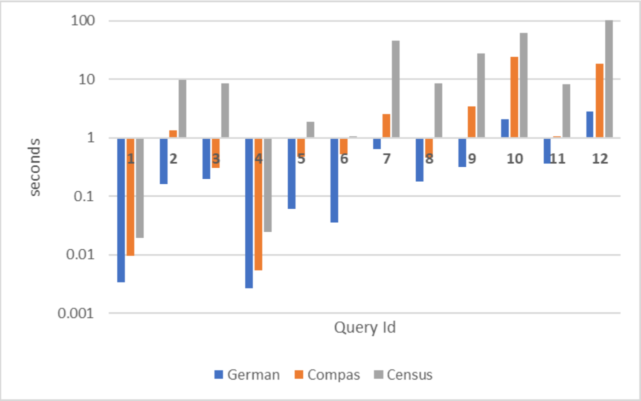


Figure 9: The provenance query times for each type of provenance query shown in Table 3

Table 7: Datasets created with the DIGen generator

	Dataset1	Dataset2	Dataset3
Scale Factor	3	5	9
Records	390978	650412	1171107
Features	45	45	45
Size	5,2 GB	8,6 GB	16 GB

6.2 Scalability with TPC-DI

The previous experiments look at the utility of the provenance gathered and the performance of the capture method across realistic machine learning pipelines. However, they do not test performance. To accomplish this, we turn to TPC-DI [37]. Using DIGen, the data generator program provided by the TPC for creating Source Data and audit information, three initial datasets were created using the fact trade table and the dim account table, as described in Table 7.

In order to test the provenance capture times, preprocessing operations have been applied to each of these datasets outside of a pipeline. Each operation is described in Table 8. Because the architecture used within FP creates provenance fragments after each pipeline operator, the experimental setup that tests each operation outside of a pipeline is valid.

Figure 10 shows how long it takes to capture and record provenance for each provenance pattern in a large dataset. Table 9 shows the initial requirements for FP in storage space for each operation. The capture mechanism scales with the size of the dataset. Processing operations that only affect a small number of data values, such as Instance Generation (IG), are fast. In addition, Feature Selection (FS), which touches every data item, only creates "wasInvalidatedBy" entries. Value Transform (VT) and Imputation (I), in this particular evaluation setup, only function on a small number of instances.

Obviously, in datasets with more missing or dirty values, these readings would change. On the other hand, the activities that create more provenance, Feature Transformation (FT) and Space Transformation (ST), take more time. Like DR, ST touches every data item. However, unlike DR, ST creates entities for every new attribute in the new column, in addition to the "wasInvalidatedBy" entries.

Table 8: The operations performed on the TCI-DI datasets to test each provenance pattern.

Operation ID	Provenance Pattern Tested	Description of Operation performed on TPC-DI generated data
FS	Feature Selection	A column (D_{*j}) is removed from the initial dataset.
FT	Feature Transformation	Transformation on *C_GNDR* column. Values of gender column are corrected.
I	Imputation	Imputation on *T_COMM* column. Null values of trade price column are filled with the average value of the column.
ST	Space Transformation	A new column with boolean values is added. 0 if commission value is null, 1 otherwise.
IG	Instance Generation	Generation of one new record.
VT	Value Transformation	Value transformation on *C_DOB* column. Invalid date of birth are replaced with NaN values.

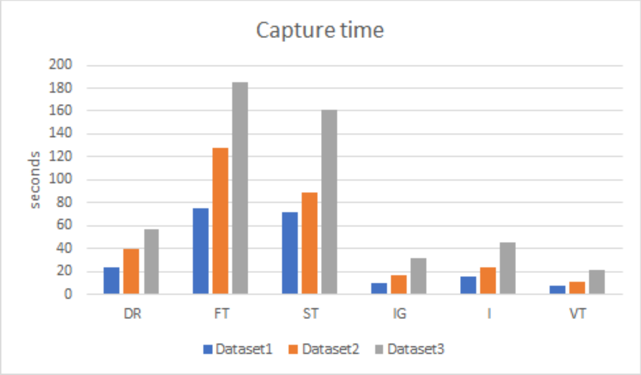


Figure 10: FP in capture time for each operation

Table 9: FP in storage space for each operation

Operation ID	Dataset 1	Dataset 2	Dataset 3
FS	77 MB	128 MB	231 MB
FT	418 MB	696 MB	1,3 GB
I	214 MB	357 MB	644 MB
ST	342 MB	568 MB	1023 MB
IG	73 MB	121 MB	217 MB
VT	576 KB	2,2 MB	2,9 MB

Furthermore, in the best case of ST, only one column is added starting from a previously existing column. In FT, a substitution of the values is performed, therefore the old entities are invalidated. Thus, in the best case ST ≤ FT.

6.3 DSSE Use Case Analysis

In Table 2, we identify a selection of real questions from data scientists in the Data Science Stack Exchange (DSSE) that analysis of provenance could help answer. We will use UC6 as an example to highlight how the fine-grained provenance available via these methods can be used to answer real-world problems.

In this example, the user separates their data into Test and Train datasets, applies an impute preprocessing step onto the Test and Train data respectively, uses Train to create a regression model, and then the Test data is used to generate predictions. Ultimately, this is an incorrect pipeline. The impute step, which creates data around a mean value parameter, should only be applied to the Training dataset. This value is then matched in the Predictions step, and imputed values automatically created in the Training dataset. By creating a second Impute step, a different mean value parameter is used for the Test data, and no values are imputed with the correct parameter later. Using the Provenance Query *Impact on Feature Spread* from Table 3 on the Test and Train datasets, it is possible to see the divergence of the values in the features of the two datasets as imputing with different mean values changes that spread.

7 CONCLUSIONS AND FUTURE WORK

In this work, we focus on fine-grained data provenance for machine learning pipelines irrespective of the pipeline tool used. Because a substantial effort goes into selecting and preparing data for use in modelling, and because changes made during preparation can affect the ultimate model, it is important to be able to trace what is happening to the data at a fine-grain level.

We highlight several real use cases to motivate the need for fine-grained provenance from the Data Science Stack Exchange (DSSE)[1]. We identify the classic provenance queries that are needed to provide information to answer these use cases. We then identify a set of provenance patterns that can be deployed across a set of machine learning pipeline operators and implement them.

Using our implementation of this system, we have tested it over real-world ML benchmark pipelines for utility and basic performance. In order to investigate scalability issues with our design, we also use the TCP-DI generator and apply several operators over that data at scale. Our results indicate that we can collect fine-grained provenance that is both useful and performant.

Future investigation into optimization techniques that aim at reducing the provenance data to the minimum that is needed to support given provenance queries, as well as methods for taking advantage of collected provenance data to support the design of new pipelines is required to continue making provenance more efficient and useful. Also, other natural extensions of our approach are under investigation, including new operators supporting advanced preprocessing data manipulations (such as join and set operations) as well as features that allows the user to specify iterative processes and to operate over multidimensional arrays of data.

ACKNOWLEDGMENTS

The authors thank Carlos Vladimiro Gonzales for making his research pipelines available for our experiments. This work was partially funded by EPSRC (EP/SO28366/1).

REFERENCES

[1] Pulkit Agrawal, Rajat Arya, Aanchal Bindal, Sandeep Bhatia, Anupriya Gagneja, Joseph Godlewski, Yucheng Low, Timothy Muss, Mudit Manu Paliwal, Sethu Raman, and et al. 2019. Data Platform for Machine Learning. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1803–1816. https://doi.org/10.1145/3299869.3314050

[2] Ahmed M Alaa and Mihaela van der Schaar. 2019. Demystifying Black-box Models with Symbolic Metamodels. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 11301–11311.

[3] Bahareh Sadat Arab, Su Feng, Boris Glavic, Seokki Lee, Xing Niu, and Qitian Zeng. 2018. GProM - A Swiss Army Knife for Your Provenance Needs. *IEEE Data Eng. Bull.* 41, 1 (2018), 51–62.

[4] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. 2001. Why and where: A characterization of data provenance. In *International conference on database theory*. Springer, Springer-Verlag, 316–330.

[5] Alvin Cheung. 2015. *Rethinking the Application-Database Interface.* Ph.D. Dissertation. Massachusetts Institute of Technology.

[6] Laura Chiticariu, Wang Chiew Tan, and Gaurav Vijayvargiya. 2005. DBNotes: a post-it system for relational databases based on provenance. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, Fatma Özcan (Ed.). ACM, 942–944.

[7] Janez Demšar, Tomaž Curk, Aleš Erjavec, Črt Gorup, Tomaž Hočevar, Mitar Milutinovič, Martin Možina, Matija Polajnar, Marko Toplak, Anže Starič, et al. 2013. Orange: data mining toolbox in Python. *The Journal of Machine Learning Research* 14, 1 (2013), 2349–2353.

[8] Janez Demšar, Tomaž Curk, Aleš Erjavec, Črt Gorup, Tomaž Hočevar, Mitar Milutinovič, Martin Možina, Matija Polajnar, Marko Toplak, Anže Starič, Miha Štajdohar, Lan Umek, Lan Žagar, Jure Žbontar, Marinka Žitnik, and Blaž Zupan. 2013. Orange: Data Mining Toolbox in Python. *Journal of Machine Learning Research* 14 (2013), 2349–2353. http://jmlr.org/papers/v14/demsar13a.html

[9] Alexander D'Amour, Hansa Srinivasan, James Atwood, Pallavi Baljekar, D. Sculley, and Yoni Halpern. 2020. Fairness is Not Static: Deeper Understanding of Long Term Fairness via Simulation Studies. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency* (Barcelona, Spain) *(FAT* '20)*. Association for Computing Machinery, New York, NY, USA, 525–534. https://doi.org/10.1145/3351095.3372878

[10] Salvador García, Sergio Ramírez-Gallego, Julián Luengo, José Manuel Benítez, and Francisco Herrera. 2016. Big data preprocessing: methods and prospects. *Big Data Analytics* 1, 1 (dec 2016), 9. https://doi.org/10.1186/s41044-016-0014-0

[11] Amirata Ghorbani and James Y. Zou. 2019. Data shapley: Equitable valuation of data for machine learning. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 2242–2251.

[12] Boris Glavic and Gustavo Alonso. 2009. Perm: Processing Provenance and Data on the Same Data Model through Query Rewriting. In *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, Yannis E. Ioannidis, Dik Lun Lee, and Raymond T. Ng (Eds.). IEEE Computer Society, 174–185.

[13] Todd J. Green, Gregory Karvounarakis, and Val Tannen. 2007. Provenance semirings. In *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Leonid Libkin (Ed.). ACM, 31–40.

[14] T. Guedes, V. Silva, M. Mattoso, M. V. N. Bedo, and D. de Oliveira. 2018. A Practical Roadmap for Provenance Capture and Data Analysis in Spark-Based Scientific Workflows. In *Workflows in Support of Large-Scale Science (WORKS)*. IEEE/ACM, 31–41. https://doi.org/10.1109/WORKS.2018.00009

[15] Trung Dong Huynh. 2018. Prov Python. https://prov.readthedocs.io/en/latest/index.html

[16] Robert Ikeda, Junsang Cho, Charlie Fang, Semih Salihoglu, Satoshi Torikai, and Jennifer Widom. 2012. Provenance-Based Debugging and Drill-Down in Data-Oriented Workflows. In *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, Anastasios Kementsietsidis and Marcos Antonio Vaz Salles (Eds.). IEEE Computer Society, 1249–1252.

[17] Matteo Interlandi, Kshitij Shah, Sai Tetali, Muhammad Gulzar, Seunghyun Yoo, Miryung Kim, Todd Millstein, and Tyson Condie. 2016. Titian: Data Provenance Support in Spark. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases* 9 (01 2016), 216–227.

[18] Nikolaos Konstantinou, Martin Koehler, Edward Abel, Cristina Civili, Bernd Neumayr, Emanuel Sallinger, Alvaro A.A. Fernandes, Georg Gottlob, John A. Keane, Leonid Libkin, and et al. 2017. The VADA Architecture for Cost-Effective Data Wrangling. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 1599–1602. https://doi.org/10.1145/3035918.3058730

[19] Himabindu Lakkaraju, Ece Kamar, Rich Caruana, and Jure Leskovec. 2017. Interpretable & Explorable Approximations of Black Box Models. *CoRR* abs/1707.01154 (2017). arXiv:1707.01154 http://arxiv.org/abs/1707.01154

[20] Seokki Lee, Sven Köhler, Bertram Ludäscher, and Boris Glavic. 2017. A SQL-Middleware Unifying Why and Why-Not Provenance for First-Order Queries. In *33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017*. IEEE Computer Society, 485–496.

[21] Raoni Lourenço, Juliana Freire, and Dennis Shasha. 2020. BugDoc: Algorithms to Debug Computational Processes. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 463–478. https://doi.org/10.1145/3318464.3389763

[22] Timothy McPhillips, Tianhong Song, Tyler Kolisnik, Steve Aulenbach, Khalid Belhajjame, Kyle Bocinsky, Yang Cao, Fernando Chirigati, Saumen Dey, Juliana Freire, et al. 2015. YesWorkflow: a user-oriented, language-independent tool for recovering workflow information from scripts. *arXiv preprint arXiv:1502.02403* (2015).

[23] Bilal Mirza, Wei Wang, Jie Wang, Howard Choi, Neo Christopher Chung, and Peipei Ping. 2019. Machine Learning and Integrative Analysis of Biomedical Big Data. *Genes* 10, 2 (jan 2019), 87. https://doi.org/10.3390/genes10020087

[24] Luc Moreau, Belfrit Victor Batlajery, Trung Dong Huynh, Danius T. Michaelides, and Heather S. Packer. 2018. A Templating System to Generate Provenance. *IEEE Transactions on Software Engineering* 44 (2018), 103–121.

[25] Luc Moreau, James Cheney, and Paolo Missier. 2013. Constraints of the PROV data model. http://www.w3.org/TR/2013/REC-prov-constraints-20130430/

[26] Luc Moreau, Paolo Missier, Khalid Belhajjame, Reza B'Far, James Cheney, Sam Coppens, Stephen Cresswell, Yolanda Gil, Paul Groth, Graham Klyne, et al. 2013. Prov-dm: The prov data model. W3C Recommendation REC-prov-dm-20130430. *WWW Consortium* (2013). https://www.w3.org/TR/prov-dm/

[27] Ramaravind Kommiya Mothilal, Amit Sharma, and Chenhao Tan. 2019. Explaining machine learning classifiers through diverse counterfactual explanations. *arXiv preprint arXiv:1905.07697* (2019).

[28] Mohammad Hossein Namaki, Avrilia Floratou, Fotis Psallidas, Subru Krishnan, Ashvin Agrawal, and Yinghui Wu. 2020. Vamsa: Tracking Provenance in Data Science Scripts. arXiv:2001.01861 [cs.LG]

[29] Arvind Narayanan. 2018. Translation tutorial: 21 fairness definitions and their politics. In *Proc. Conf. Fairness Accountability Transp., New York, USA*.

[30] Xing Niu, Raghav Kapoor, Boris Glavic, Dieter Gawlick, Zhen Hua Liu, and Venkatesh Radhakrishnan. 2017. Provenance-Aware Query Optimization. In *33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017*. IEEE Computer Society, 473–484.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[32] Devin Petersohn, William W. Ma, Doris Jung Lin Lee, Stephen Macke, Doris Xin, Xiangxi Mo, Joseph Gonzalez, Joseph M. Hellerstein, Anthony D. Joseph, and Aditya G. Parameswaran. 2020. Towards Scalable Dataframe Systems. *Proc. VLDB Endow.* 13, 11 (2020), 2033–2046.

[33] João Felipe Pimentel, Saumen Dey, Timothy McPhillips, Khalid Belhajjame, David Koop, Leonardo Murta, Vanessa Braganholo, and Bertram Ludäscher. 2016. Yin & Yang: demonstrating complementary provenance from noWorkflow & YesWorkflow. In *International Provenance and Annotation Workshop*. Springer, 161–165.

[34] João Felipe Pimentel, Juliana Freire, Leonardo Murta, and Vanessa Braganholo. 2016. Fine-grained provenance collection over scripts through program slicing. In *International Provenance and Annotation Workshop*. Springer, 199–203.

[35] João Felipe Pimentel, Juliana Freire, Leonardo Murta, and Vanessa Braganholo. 2019. A survey on collecting, managing, and analyzing provenance from scripts. *ACM Computing Surveys (CSUR)* 52, 3 (2019), 1–38.

[36] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. 2017. noWorkflow: a Tool for Collecting, Analyzing, and Managing Provenance from Python Scripts. *Proc. VLDB Endow.* 10, 12 (2017), 1841–1844.

[37] Meikel Poess, Tilmann Rabl, Hans-Arno Jacobsen, and Brian Caufield. 2014. TPC-DI: The First Industry Benchmark for Data Integration. *Proc. VLDB Endow.* 7, 13 (Aug. 2014), 1367–1378. https://doi.org/10.14778/2733004.2733009

[38] Fotis Psallidas and Eugene Wu. 2018. Smoke: Fine-grained Lineage at Interactive Speed. *Proc. VLDB Endow.* 11, 6 (2018), 719–732.

[39] Fotis Psallidas and Eugene Wu. 2018. Smoke: Fine-grained lineage at interactive speed. *Proceedings of the VLDB Endowment* 11, 6 (2018), 719–732.

[40] Marco Túlio 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, San Francisco, CA, USA, August 13-17, 2016*, Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, and Rajeev Rastogi (Eds.). ACM, 1135–1144.

[41] Sebastian Schelter, Joos-Hendrik Böse, Johannes Kirschnick, Thoralf Klein, Stephan Seufert, and Amazon. 2018. Declarative Metadata Management: A Missing Piece in End-To-End Machine Learning. In *SysML Conference*.

[42] Stefanie Scherzinger, Christin Seifert, and Lena Wiese. 2019. The Best of both Worlds: Challenges in Linking Provenance and Explainability in Distributed Machine Learning. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1620–1629.

[43] Zeyuan Shang, Emanuel Zgraggen, Benedetto Buratti, Ferdinand Kossmann, Philipp Eichmann, Yeounoh Chung, Carsten Binnig, Eli Upfal, and Tim Kraska. 2019. Democratizing Data Science through Interactive Curation of ML Pipelines. In *Proceedings of the 2019 International Conference on Management of Data* (Amsterdam, Netherlands) *(SIGMOD '19)*. Association for Computing Machinery, New York, NY, USA, 1171–1188. https://doi.org/10.1145/3299869.3319863

[44] Stefan Studer, Thanh Binh Bui, Christian Drescher, Alexander Hanuschkin, Ludwig Winkler, Steven Peters, and Klaus-Robert Mueller. 2020. Towards CRISP-ML (Q): A Machine Learning Process Model with Quality Assurance Methodology. *arXiv preprint arXiv:2003.05155* (2020).

[45] MingJie Tang, Saisai Shao, Weiqing Yang, Yanbo Liang, Yongyang Yu, Bikas Saha, and Dongjoon Hyun. 2019. SAC: A System for Big Data Lineage Tracking. In *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019*. IEEE, 1964–1967.

[46] Manasi Vartak, Joana M. F. da Trindade, Samuel Madden, and Matei Zaharia. 2018. MISTIQUE: A System to Store and Query Model Intermediates for Model Diagnosis. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). ACM, 1285–1300.

[47] Yinjun Wu, Val Tannen, and Susan B. Davidson. 2020. PrIU: A Provenance-Based Approach for Incrementally Updating Regression Models. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 447–462.

[48] Zhepeng Yan, Val Tannen, and Zachary G. Ives. 2016. Fine-grained Provenance for Linear Algebra Operators. In *8th USENIX Workshop on the Theory and Practice of Provenance, TaPP 2016, Washington, D.C., USA, June 8-9, 2016*, Sarah Cohen Boulakia (Ed.). USENIX Association.

[49] Qian Zhang, Paul J Morris, Timothy McPhillips, James Hanken, David Lowery, Bertram Ludäscher, James Macklin, Robert Morris, and John Wieczorek. 2017. Using YesWorkflow hybrid queries to reveal data lineage from data curation activities. *Biodiversity Information Science and Standards* 1 (2017), e20380.

[50] Nan Zheng, Abdussalam Alawini, and Zachary Ives. 2019. Fine-Grained Provenance for Matching & ETL. *Proceedings. International Conference on Data Engineering* 2019 (04 2019), 184–195. https://doi.org/10.1109/ICDE.2019.00025