Dataset Relationship Management

Zachary G. Ives, Yi Zhang, Soonbo Han, Nan Zheng
University of Pennsylvania
Philadelphia, PA
zives,yizhang5,soonbo,nanzheng@cis.upenn.edu

ABSTRACT

The database community has largely focused on providing improved transaction management and query capabilities over records (and generalizations thereof). Yet such capabilities address only a small part of today's data science tasks, which are often much more focused on discovery, linking, comparative analysis, and collaboration across holistic datasets and data products.

Data scientists frequently point to a strong need for data management — with respect to their many datasets and data products. We propose the development of the *dataset relationship management system* to support five main classes of operations on datasets: **reuse** of schema, data, curation, and work across many datasets; **revelation** of provenance, context, and assumptions; rapid **revision** of data and processing steps; system-assisted **retargeting** of computation to alternative execution environments; and metrics to **reward** individuals' contributions to the broader data ecosystem. We argue that the recent adoption of computational notebooks (particularly JupyterLab and Jupyter Notebook), as a unified interface over data tools, provides an ideal way of gathering detailed information about how data is being used, i.e., of transparently capturing dataset provenance and relationships, and thus such notebooks provide an attractive mechanism for integrating dataset relationship management into the data science ecosystem. We briefly outline our experiences in building towards JuNEAU, the first prototype DRMS.

ACM Reference Format:
Zachary G. Ives, Yi Zhang, Soonbo Han, Nan Zheng. 2019. Dataset Relationship Management. In *Proceedings of Conference on Innovative Database Systems Research (CIDR 19)*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

The database field has evolved over the decades, but even today the "core" of the discipline is focused on *representing* and *reliably, consistently storing* individual data items, and on supporting SQL-style queries over such data. The big data era has brought new desires to interface with external datasets, run machine learning algorithms, and visualize results. Our community's response has been to adapt our existing, record- and transaction-oriented platforms to interface with such capabilities, rather than changing their basic functionality or abstractions.

Unfortunately, traditional database capabilities play a smaller and smaller role in data science at scale — especially in research and in large organizations, where data is assembled by many users and comes in many versions. Data analysis is not simply being performed over an existing data warehouse. Instead, datasets are pulled from distributed filesystems and the open web, and possibly stored in a data lake. The emphasis is not on managing individual records, but on finding, integrating, and analyzing *actionable*, human-curated datasets, or even assemblies of datasets. Data cannot be safely assumed to be largely clean or consistent, and it often comes from different parties with different levels of trustworthiness or relevance. Data curation, hypothesis testing, (machine learning) feature or algorithm selection, record linking / coregistration, and comparative analysis are the common tasks of the day. Moreover, these tasks are often done *in teams*, incorporate open data and code, and need to be applied *repeatedly* to other (possibly future) data.

As an example of the "sea change," consider the Penn Database Group's experiences in working with collaborators in several disparate areas of biomedical data science. Scientific communities have, for years, promoted public sharing of data in Web- (and today cloud-) backed databases [32, 37, 38], with a goal of facilitating data reuse. Yet our experience, both as providers and consumers of such data, has shown that access to tables and records plays a minor role in promoting an "open data ecosystem" for data production, sharing, and analysis.

Example 1.1. *For more than seven years, we have worked with neuroscientists [16, 31] to collect and share large multimodal datasets from across epilepsy and behavioral research. Data hosted on our platform, IEEG.org, includes (HIPAA-compliant, anonymized) patient case histories, MRI and CT imaging, long (up to 1.5-year), high-frequency (up to 32KHz) time series data (primarily intracranial EEG recordings). Perhaps the most vital data in our database is actually* annotations *over the imaging (for seizure onset information, implanted electrodes, etc) and time-series data (information about seizure state or behavior). Data from IEEG.org was successfully used to develop a series of dramatically better seizure detection and prediction algorithms [9, 15]. Yet despite these successes, the presence of over 4500 datasets, and nearly 3200 users, we have found surprising limitations in the community's usage patterns (as we have previously described in part [26]). First, scientists are concerned that data remains heterogeneous, in terms of quality, assumptions (provenance), curation, and schema. Few mechanisms exist (other than shared code on GitHub or Jupyter notebooks on Kaggle) to promote reuse, and especially to find commonalities. Ultimately this means scientists are unlikely to reuse or trust others' data, even if that data is technically available. Second, data science often relies on testing hypotheses and training or comparing algorithms over a subset of the data, then repeating over larger relevant datasets. The path from a one-off-script over a single dataset* *on a local machine, to a more regularized, parameterized program that can run across a cluster, is fraught with challenges. Finally, too few metrics exist to fully recognize those who share curated results, annotations, etc.*

Our experiences in other data sharing communities, revolving around mobile health data [30] and in multi-site high-throughput gene sequencing, have shown similar issues. Reasoning about data becomes less valuable than reasoning about datasets and *data products* (derived datasets, machine learning classifiers, data visualizations, etc.). We argue that the data management community should broaden its focus beyond transactions and queries, and even beyond issues of data provenance management and version control, to manage the **relationships among datasets and data products** and aid the user in making **best use of overwhelming volumes of not-equally-useful and code data**:

Reuse. How do data scientists reuse the best-curated data or ensure that corrections persist over updates to remote data sources? Once they have developed a new algorithm or workflow, can we help them (1) generalize it into a reusable module or script, (2) apply it *en masse* to other (or future) datasets?

Reveal. Can we help data scientist teams understand the provenance [12] (and context [21]) of a data product or code module, including how it relates to other tasks and other data?

Revise. Data and code both evolve over time. Version control systems [7] manage the storage of versioned data and code, but do not aid in ensuring *consistent* processing and refreshing of all results.

Retarget. Today there are a multitude of data analysis platforms and languages, which handle different scales and execution environments. Through code analysis and rewriting techniques, we can aid the user in retargeting certain data analysis, e.g., from an in-memory implementation to one that runs on the cloud.

Reward. It has become clear that measures such as publication count are inadequate for measuring impact. We need better infrastructure for tracking individuals' value-add to the data science process — whether it is code, data, or curation.

Even as we consider how to meet these needs, we must also acknowledge that a solution should fit naturally into a scientist's data science workflow, and that to be most useful it must have a nearly global perspective on how data and datasets are being used. That workflow is largely outside of the realm of the DBMS and SQL.

Fortunately, although data scientists use a plethora of languages (Matlab, R, Python, Julia, sometimes even SQL in limited cases), machine learning toolkits (PyTorch, TensorFlow, Keras, MLLib, etc.), visualization tools, and storage subsystems (cloud storage, HDFS, graph databases, RDBMSs) — there has been a convergence on *computational notebook* software, particularly Jupyter Notebook [41] and its successor JupyterLab, as a single integrated environment for controlling the majority of data science tasks[1].

User interactions via computational notebooks. JupyterLab provides a unified user experience, largely based on a single interactive Web document ("notebook") containing a sequence of HTML "cells." Cells may be simple Markdown descriptions and headings, providing important *contextual or descriptive information* to the reader; program fragments written in a particular target language, which may bring in external libraries and link to cloud execution environments; and visual or tabular outputs produced by code. Since JupyterLab runs within a Web browser, it can incorporate a wide variety of Javascript, Web, and other plugins including visualization tools and Web services. JupyterLab also provides mechanisms for invoking programs via the operating system shell — thus allowing notebooks to interact with programs and scripts via shared files. Broadly, it can be viewed as the most popular integrated development environment for data science, and it has an added virtue of being open-source and extensible via plug-ins.

Changing the underlying data management infrastructure. Unfortunately, JupyterLab has an extremely rudimentary data model: a notebook is simply a JSON file that contains code that references externally stored files and URLs. Programmatic state (e.g., variables and other side effects) is carried from cell to cell according to the sequence in which the cells have been executed, which does not have to match the order in which the cells occur in the document. Cells can be overwritten in ways that prevent the notebook from even being correctly re-executed. Recent work [11, 29, 42] has developed more sophisticated models for tracking dependencies among cells, thus enabling provenance capture and reproducibility.

Going beyond these, we argue that the needs of data management for data science — including the "five R" tasks listed above — can be best addressed by building a full data management layer *underneath* JupyterLab and other notebook software, which not only individually manages the cells within the notebook and the relationships among notebooks, but links to filesystem / external data, and derived data. Our goals include **reveal**ing provenance — but also exploiting data and execution semantics to find and propose *relationships* among data and code artifacts, and ultimately to produce modular **reuse** and **retarget**ing of human effort across data **revision**s, and also metrics for helping **reward** contributions to any data product.

In this paper, we describe our early efforts to build such a *dataset relationship management system*, JuNEAU[2]. JuNEAU has the ability to extract *semantic information* from the cells and content of the notebooks, and to enable *reasoning about relationships among datasets and data products* (via cells and their code) and *reasoning about cells* (the datasets and data products). Its main objective is in some sense to promote the effective reuse of the "best" data products and code.

The paper is organized as follows. Section 2 outlines the core capabilities of our proposed class of data relationship management systems. Section 3 outlines our early prototype, capabilities, experiences, and open challenges. We describe related work in Section 4 and conclude in Section 5

2 DATASET RELATIONSHIP MANAGEMENT

A relational database system is primarily responsible for managing metadata (schemas), data records (as tables), and certain kinds of data transformations (e.g., views). At its heart, the JuNEAU dataset relationship management system (DRMS) is a middleware system

[1]RStudio and Apache Zeppelin are also popular notebook platforms, but seem to be much less so than Jupyter.

[2]"JUpyter Notebook with Enhanced Access and Understanding"

that combines information about data and code usage and **revision** patterns, with information about data and code's semantics and provenance — to help data scientists and data analysts best leverage, **reuse**, and **retarget** existing code, curation, metadata and data. To promote data sharing across users, it both **reveal**s context and provenance, and provides metrics to **reward** data, code, and curation that have measurable impact.

To the user, JuNEAU provides the default abstraction of a computational notebook and the JupyterLab integrated development environment. However, internally it "deconstructs" the notebook into richer data model, comprised of a (versioned) DAG of **data products**, **annotations and descriptors**, and **code cells**. See Figure 1 for an example. It also tracks relationships that span across notebooks, and even between notebooks and code run from the JupyterLab shell.

The DRMS targets large multi-user, multi-dataset settings in which the challenge is not so much in acquiring access to data, but in finding, reusing, and standardizing upon datasets. The data may exist in different versions; analysis code may get updated; different users may have different means of cleaning and curating the data. We consider *code-to-code*, *code-to-data*, *data-to-code*, and *data-to-data* relationships.

Code-to-data relationships. Much of data science involves building code snippets and scripts — often in a way that fails to promote effective reuse (desirable for modularity and maintainability) or readability (critical for collaboration and reproducibility). Worse, data scientists are typically not trained software engineers, and computational notebooks encourage a copy-paste-and-tweak mentality. Thus a key capability in the DRMS is to help identify how code interfaces to and produces data.

In more detail, a cell or assembly of cells in Jupyter typically has *dependencies* — potentially to a set of input files or URLs from which source data is initially loaded, or to a collection of variables loaded in a prior cell — and typically produces output data products (including embedded visualizations, tables or dataframes, and/or files). Using code analysis, the DRMS attempts to (1) infer what the inputs and outputs are, and (2) identify the expected types and schema restrictions. From this, we can extract the assembly of cells into a reusable block with parameterizable interfaces, and we can determine where the assembly can be applied.

Data-to-code relationships. Much the way code has an interface, we can think of datasets also having an interface — in terms of the operations that may be applied to them, how they were produced, and so on. Here, the DRMS relies on its ability to extract provenance for the data, as well as provenance information about how the data is itself used. Provenance in a DRMS is, of course, a first-class citizen, and provenance querying and visualization [27] as well as usage tracking are core capabilities. However, in an active data environment, the DRMS should amass enough provenance information to help with common data wrangling and integration tasks: (1) find cells/code modules that are used to import files and map them to tables or dataframes; (2) suggest candidate schemas for a data product stored as a file; (3) identify other datasets that have been joined/coregistered/record-linked with a data product. It may also be able to suggest data processing steps / cells that perform cleaning and normalization. Such steps can be converted into reusable pipelines for future datasets, and they can also be leveraged by other users. We note that reuse of processing steps is likely to also lead to more standard schemas as well as practices.

Perhaps more interesting (and described in more detail in Section 3), the DRMS allows us to **holistically compare across data products and workflows**, for instance seeing if different workflow versions produce identical output or identifying where they differ. Such capabilities are essential to reproducible research, as they allow regression testing and uniform treatment of all datasets.

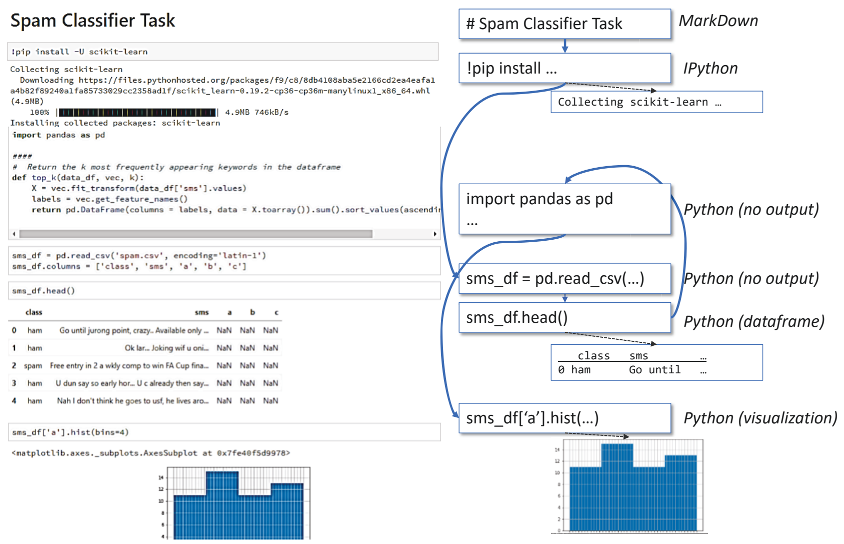
Reasoning about code semantics and relationships. Perhaps not surprisingly, the vast majority of data analysis tools today include high-level operations over arrays, matrices, and — especially relevant to the DBMS community — collections of structured objects. Microsoft's LINQ, Java's streams interface, Python and R's dataframes, and the multitude of map-reduce implementations in all environments, mean that is possible to perform simple code and dataflow analysis to derive relational algebra-like descriptions of some of the lines of code within a cell. Given a high-level declarative specification for a piece of code, the DRMS can infer how the code processes and produces individual records. This can be extremely useful in reasoning about differences in results across workflows. It has another advantage, namely that we can easily generate *retarget*ed code, e.g., to move from Python Pandas to Apache Spark, or potentially even convert from a local SciKit-Learn implementation to a more scalable learning platform like TensorFlow.

Reasoning about data semantics and relationships. Traditionally, the database community has explored how to search for datasets by keywords, facets, or even taxonomies. We argue that search-by-example-table and search-by-similar-provenance are more useful in many data science settings. If a user is analyzing a table with a small set of results, he or she may wish to scale up to a bigger training set or may want to identify potentially interesting data with similar characteristics. Here, similarity sometimes that values substantially overlap — but it may also mean similar provenance, similar fields, and similar data distributions. Another form of search demanded by users is for *the best curated version of a dataset* — ideally, a version of the dataset with fresh data but also the most authoritative corrections.

Finally, many data scientists are not experienced at designing schemas, nor at thinking comprehensively about what attributes to capture. As the user stores or imports data, the DRMS should aid in schema design by suggesting schema elements that are commonly associated with the data and the currently specified fields — a "schema auto-complete" [25].

3 PROTOTYPE AND OPEN CHALLENGES

Over the past year, motivated by our interactions with data scientists at Penn, we have been building an early prototype of the JuNEAU DRMS. Our implementation uses a combination of Python (front-end components serving the Web interface, currently as a storage manager plugin for Jupyter Notebook and JupyterLab) and Java (back-end components), and stores its content in a combination of object key-value store (Minio or Amazon S3), graph database (Neo4J), and relational DBMS (PostgreSQL).


Spam Classifier Task

```
!pip install -U scikit-learn
```

```
Collecting scikit-learn
  Downloading https://files.pythonhosted.org/packages/f9/c8/8db4108aba5e2166cd2ea4eafa1
a4b82f89240a1fa85733029cc2358ad1f/scikit_learn-0.19.2-cp36-cp36m-manylinux1_x86_64.whl
(4.9MB)
    100% |████████████████████████████████| 4.9MB 746kB/s
Installing collected packages: scikit-learn
```

```
import pandas as pd

####
#  Return the k most frequently appearing keywords in the dataframe
def top_k(data_df, vec, k):
    X = vec.fit_transform(data_df['sms'].values)
    labels = vec.get_feature_names()
    return pd.DataFrame(columns = labels, data = X.toarray()).sum().sort_values(ascendi
```

```
sms_df = pd.read_csv('spam.csv', encoding='latin-1')
sms_df.columns = ['class', 'sms', 'a', 'b', 'c']
```

```
sms_df.head()
```

	class	sms	a	b	c
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

```
sms_df['a'].hist(bins=4)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe40f5d9978>
```

Figure 1: Example of a JupyterLab notebook and its associated cell-oriented data model. Note the execution order and program dataflow dependencies (blue lines) may not match the presentational order.

Our early prototype focuses on three aspects of the DRMS: (1) taking procedural, data-centric code and finding a way to capture its semantics (mostly) declaratively; (2) identifying data usage patterns; (3) finding data relationships.

3.1 "Lifting" Procedural Code to Declarative Representations

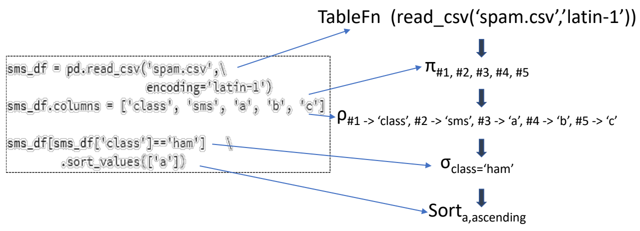
JUNEAU can make use of "opaque" programs and code cells, as workflow "black boxes." However, to do deeper analysis and to support retargeting of the code to different platforms, it must be able to extract semantic information (or it must be provided such information, e.g., by a programmer). As described previously, most data analysis code heavily emphasizes arrays, matrices, and collections of structured objects (or tabular data). It makes use of "bulk operations" such as matrix transformations, bulk arithmetic operations, and relational algebra (filter, join) or map-reduce operations, as well as calls to high-level libraries and toolkits.

It is fairly straightforward to develop limited program analysis modules for different target languages, which can parse the contents of code cells, track operations, and track dataflow dependencies. In JUNEAU we currently support Python, and inspired by the noWorkflow system [36] we track dataflow between statements and across cell boundaries. We use reflection and debugger support to understand the types, e.g., to detect dataframes and arrays, and we have developed a prototype dictionary of function signatures to detect when relational operators are being applied to the tabular data. Figure 2 shows an example of how a segment of Python code (using Pandas) makes a variety of function calls producing a dataframe, which we map into a series of relational operators. Obviously, not all code (especially iterative or recursive code) maps naturally to relational expressions. However, we can generally break the lines of code into a DAG in which nodes represent blocks of relational-algebra-equivalent code and "other" code. Sometimes the relational code will include calls to user-defined functions (e.g., a user-defined map operation).

Often, JupyterLab cells make calls to external libraries or modules, e.g., for gene sequence analysis. Moreover, users may invoke programs from the JupyterLab shell. Thus, to complement our cell-level code analysis, we also allow developers to associate *module descriptors* with black-box programs [53]. Module descriptors are declarative specifications (in the form of relational algebra trees specifying the operations, and schemas of the inputs and outputs) of the structured data extraction and record-to-record transformations being done in each program, combined with selected user-defined functions. Module descriptors allow the platform to selectively recompute fine-grained record-to-record provenance, and to capture provenance for hierarchical, string, and image datatypes.

Revealing fine-grained provenance. Our study of "lifting" procedural code to a declarative form was motivated by the problem of helping data scientists reason about revisions to code (algorithm implementations) and data (reference libraries or datasets). Often, as code and data are revised, the new version produces *different output* from the old one. To ensure reproducibility, it becomes critical to understand *where and why* such differences arise, at a *fine* level of granularity (records as opposed to complete files).

Our early JUNEAU prototype's PROVision sub-module [53] facilitates this type of reasoning about the differences between revisions. It starts with a *diff3* comparison of the outputs between different code versions, then localizes which output records are different. Finally, it uses the declarative description of workflow modules to compute *fine-grained provenance* showing which input records contributed to the different outputs. This work develops techniques for pruning and optimization, as well as methods for embedding user-defined extraction, record linking, and aggregation functions into declarative code while producing provenance. Results from ETL tasks and gene sequence alignment workflows showed that, thanks to its data indexing and pruning techniques, JUNEAU could



```
sms_df = pd.read_csv('spam.csv',\
                     encoding='latin-1')
sms_df.columns = ['class', 'sms', 'a', 'b', 'c']

sms_df[sms_df['class']=='ham']  \
        .sort_values(['a'])
```

TableFn (read_csv('spam.csv','latin-1'))
$\pi_{\#1, \#2, \#3, \#4, \#5}$
$\rho_{\#1 \to \text{'class'}, \#2 \to \text{'sms'}, \#3 \to \text{'a'}, \#4 \to \text{'b'}, \#5 \to \text{'c'}}$
$\sigma_{\text{class='ham'}}$
$\text{Sort}_{\text{a,ascending}}$

Figure 2: Example of converting Python Pandas code into a relational algebra expression

typically trace the provenance of specific output results within a few seconds, even when re-running the entire workflow could take hours.

Open challenges. Our initial JuNEAU prototype executes declarative specifications of Python code in its own proprietary query engine, which interfaces seamlessly with UDFs in CPython. We are developing a cross-compiler for executing the same code under Apache Spark, which would provide a mechanism for "scaling out" the code to distributed data. More broadly, we foresee opportunities to retarget the code to other runtime systems (with the caveat that performance will depend on workload and data distribution). A natural question is whether certain code segments could be automatically targeted to the most appropriate runtime environments — like some polystore systems [1, 17] seek to do. We also believe that the concept of provenance needs to be clarified for "holistic" machine learning operations such as training and clustering.

We also believe there is an opportunity to exploit database view materialization and incremental maintenance strategies to speed up the execution of certain notebooks. There has been a good deal of work on compiler optimizations for data-driven code, including heterogeneous environments [14, 45]. However, many more opportunities exist for incorporating reasoning about workload patterns (e.g., across notebooks and across files) and for combining fine-grained provenance computation with data driven execution.

3.2 Capturing and Identifying Data Usage Patterns

Applying techniques from graph pattern mining [49] and from recommendation systems [4, 51] to the provenance (data usage) graph, the DRMS can promote the **reuse** of data: it can detect datasets with similar provenance or that are used in similar ways, as well as code modules and cells applied to similar kinds of data. Moreover, if a dataset is the subject of frequently occurring cleaning steps or record-level manual edits, the system should be able to detect this pattern and recommend the derived dataset instead. (Recent work [8] has suggested a basic algebra for reasoning about dataset changes.)

Of course, such capabilities rely on having detailed provenance information to mine. A wide variety of methods and platforms have been proposed for capturing coarse-grained (file-to-file) provenance over workflows within dedicated environments [5, 19, 34, 39], low-level provenance provided by operating system-level instrumentation [20, 35, 35, 47], and fine-grained (record-to-record) provenance over SQL-like computations [3, 13, 18, 23, 24, 27, 33, 50].

A key challenge lies in knowing how much provenance, at what granularity, to capture *a priori*, and what to reproduce as needed. With deterministic code augmented by effective version management, one must only know the base data and the code cells or programs (and their parameters) in order to fully reproduce provenance. However, to facilitate DRMS reasoning (such as detection of common data processing patterns) as well as to aid the user (such as in answering user queries about a dataset's detailed provenance, or reasoning about the effects of a revision), we may wish to materialize more in order to reduce the amount of work needed on-demand. We may also need to "lift" the provenance from low-level steps into higher-level relationships, and/or to convert it into a normalized representation.

One approach to addressing the challenges cited above is to develop mechanisms for *provenance views* that can convert subgraphs comprised of low-level provenance information (e.g., program read and write operations) into higher-level representations (e.g., nodes representing derivations). Such views could also be used to normalize the structure of a graph, e.g., removing order information that does not matter. We have been developing infrastructure for efficiently computing views over provenance graphs (and, more generally, graph databases). For generality, views may be comprised of multiple pattern-matching and substitution rules. Given that such rules may interact, we have developed typechecking algorithms and means of expressing precedence, so our views will produce deterministic output. Inspired by access support relations [28], we are also exploring subgraph indexing techniques that speed query answering.

Open challenges. The regularity within data science workflows, and even queries, offers many opportunities to apply data compression and selective materialization. We expect to develop cost-based optimization techniques to address this problem.

3.3 Detecting and Exploiting "Other" Data Relationships

Provenance indirectly relates data sources to data products. However, data relates to data, both directly and indirectly, through sharing or overlap: one dataset may be a revision of another, or they may be related via a common "ancestor"; one dataset may represent a derivation from another (e.g., through a cleaning step); one dataset may reference another using foreign keys; two datasets may represent independent observations of the same phenomenon. In all of these cases, the overlap between datasets (on some subset of their fields) is an extremely useful measure of *dataset relatedness*, which can help JuNEAU aid the user in leveraging and reusing others' work.

Early work in JuNEAU develops search and recommendation capabilities to promote dataset reuse. Our focus is on letting users search a data lake not merely by keywords, but by existing tables or table fragments, combined with their provenance and with optional filter criteria. Our definitions for table relatedness include *similarity*, capturing tables with similar schemas or values, which can help data scientists accumulate additional data, e.g., more training data for a machine learning module; *linkability*, which discovers other relations that can be joined with a given table, and is useful for adding features to datasets; and *provenance similarity*, which finds tables that were produced in the same or similar ways. To enable such capabilities, we have developed ranking schemes for *table relatedness* as well as indexing techniques.

We further utilize the relatedness and overlap measures to link new and existing data products (base tables and intermediate results). This provides a form of index that supports rapid reasoning about data overlap between these results, and results computed in other notebooks, perhaps by other users. Our results are still preliminary, but they show promise for both compressing datasets and finding related datasets.

Open challenges. Our early efforts to support ranked search over tables are primarily based on measures of data relatedness for reusability, as described above. A full search engine should further consider user intent, provenance similarity, dataset authoritativeness and freshness, and filter predicates over schema and data elements. Such goals require us to tackle questions of how to interact with the user to learn to rank tables [48], learn trustworthiness [40], and balance among multiple scoring dimensions. Much as with existing table search systems [10, 43], a good deal of tuning and user validation will be required to get things right. However, from speaking with data scientists it is clear that such "signals" are absolutely essential to assessing data.

4 RELATED WORK

While our work has been especially shaped by the experiences of working with modern data scientists, some of the underlying issues have been articulated in the past (though often with different goals). Bernstein et al.'s efforts to build generic model management systems [6] were based on the needs of managing schema evolution, data integration, and a plethora of different data representations within the enterprise. Some of those same goals were targeted by the Clio project [44]. The Ground [22] project sought to establish a common representation for data relationships across a multitude of sources and environments, not dissimilar to the provenance graph in the DRMS. Bleifuß et al. [8] have proposed a set of data cube-style operators for analyzing changes within datasets.

Scientific data management has also been a topic of study in the database community and beyond, with scientific workflow management systems with integrated provenance capture [19, 34, 39] as perhaps the most important area of related work. Our work builds upon these ideas to augment them with fine-grained provenance and to look holistically *across* usage patterns within a community. Finally, tools for managing data outside the database [2] and for array and matrix processing [46, 50, 52] have all had impact on how today's big data platforms and libraries optimize their computation. We assume the use JupyterLab to integrate across various data management platforms.

5 CONCLUSIONS AND FUTURE WORK

Much of data science is focused beyond individual records, instead looking at heterogeneous datasets with different provenance, a multitude of interrelated data products, and a plethora of heterogeneous data analysis and visualization tools. We argue that the DBMS community should embrace the management not only of data and its relationships, but datasets and their relationships — to help users navigate the overwhelming number of data products and code versions, and to promote the five R's: reuse of the best resources, revelation of context to aid in collaboration, understanding how revisions impact results, retargeting of code to alternative platforms, and metrics that enable communities to reward their best contributors. Our early experiences with JuNEAU point to some of the key issues, but a good deal of open work remains.

REFERENCES

[1] Divy Agrawal, Sanjay Chawla, Bertty Contreras-Rojas, Ahmed Elmagarmid, Yasser Idris, Zoi Kaoudi, Sebastian Kruse, Ji Lucas, Essam Mansour, Mourad Ouzzani, et al. 2018. RHEEM: enabling cross-platform data processing: may the big data be with you! *Proceedings of the VLDB Endowment* 11, 11 (2018), 1414–1427.

[2] Ioannis Alagiannis, Renata Borovica, Miguel Branco, Stratos Idreos, and Anastasia Ailamaki. 2012. NoDB: efficient query execution on raw data files. In *SIGMOD*. ACM, 241–252.

[3] Yael Amsterdamer, Daniel Deutch, and Val Tannen. 2011. Provenance for aggregate queries. In *PODS*. 153–164.

[4] S. Baluja, R. Seth, D. Sivakumar, Y. Jing, J. Yagnik, S. Kumar, D. Ravichandran, and M. Aly. 2008. Video suggestion and discovery for YouTube: taking random walks through the view graph. In *WWW*.

[5] Louis Bavoil, Steven P. Callahan, Patricia J. Crossno, Juliana Freire, Carlos E. Scheidegger, Claudio T. Silva, and Huy T. Vo. 2005. VisTrails: Enabling Interactive Multiple-View Visualizations. *IEEE Visualization* (2005), 135–142.

[6] Philip A. Bernstein, Alon Y. Halevy, and Rachel Pottinger. 2000. A Vision of Management of Complex Models. *SIGMOD Record* 29, 4 (December 2000), 55–63.

[7] Souvik Bhattacherjee, Amit Chavan, Silu Huang, Amol Deshpande, and Aditya Parameswaran. 2015. Principles of Dataset Versioning: Exploring the Recreation/Storage Tradeoff. *Proc. VLDB Endow.* 8, 12 (Aug. 2015), 1346–1357.

[8] Tobias Bleifuß, Leon Bornemann, Theodore Johnson, Dmitri V. Kalashnikov, Felix Naumann, and Divesh Srivastava. 2018. Exploring Change - A New Dimension of Data Analytics. *PVLDB* 12, 2 (2018), 85–98. http://www.vldb.org/pvldb/vol12/p85-bleifuss.pdf

[9] Benjamin H Brinkmann, Joost Wagenaar, Drew Abbot, Phillip Adkins, Simone C Bosshard, Min Chen, Quang M Tieng, Jialune He, FJ Muñoz-Almaraz, Paloma Botella-Rocamora, et al. 2016. Crowdsourcing reproducible seizure forecasting in human and canine epilepsy. *Brain* 139, 6 (2016), 1713–1722.

[10] Michael J. Cafarella, Alon Y. Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. WebTables: exploring the power of tables on the web. *PVLDB* 1, 1 (2008), 538–549.

[11] Lucas AMC Carvalho, Regina Wang, Yolanda Gil, and Daniel Garijo. 2017. NiW: Converting Notebooks into Workflows to Capture Dataflow and Provenance. In *Proceedings of Workshops and Tutorials of the 9th International Conference on Knowledge Capture (K-CAP2017)*.

[12] James Cheney, Laura Chiticariu, and Wang Chiew Tan. 2009. Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases* 1, 4 (2009), 379–474.

[13] James Cheney, Laura Chiticariu, and Wang Chiew Tan. 2009. Provenance in Databases: Why, How, and Where. *Foundations and Trends in Databases* 1, 4 (2009), 379–474.

[14] Alvin Cheung, Armando Solar-Lezama, and Samuel Madden. 2013. Optimizing Database-backed Applications with Query Synthesis. *SIGPLAN Not.* 48, 6 (June 2013), 3–14. https://doi.org/10.1145/2499370.2462180

[15] Francis Collins. 2015. Epilepsy Research Benefits from the Crowd. directorsblog.nih.gov/2015/01/20/epilepsy-research-benefits-from-the-crowd.

[16] Francis Collins. 2015. NIH Director's Blog: Epilepsy Research Benefits from the Crowd. http://directorsblog.nih.gov/2015/01/20/epilepsy-research-benefits-from-the-crowd/.

[17] Jennie Duggan, Aaron J Elmore, Michael Stonebraker, Magda Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier, Tim Mattson, and Stan Zdonik. 2015. The BigDawg polystore system. *ACM Sigmod Record* 44, 2 (2015), 11–16.

[18] Boris Glavic and Gustavo Alonso. 2009. Perm: Processing Provenance and Data on the Same Data Model through Query Rewriting. In *ICDE*. 174–185.

[19] Jeremy Goecks, Anton Nekrutenko, and James Taylor. 2010. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome biology* 11, 8 (2010), R86.

[20] Soonbo Han. [n. d.]. Provenance Tracker. https://pennprovenance.net/index.php?n=Main.Tracker.

[21] Joseph M. Hellerstein, Vikram Sreekanti, Joseph E. Gonzalez, James Dalton, Akon Dey, Sreyashi Nag, Krishna Ramachandran, Sudhanshu Arora, Arka Bhattacharyya, Shirshanka Das, Mark Donsky, Gabriel Fierro, Chang She, Carl Steinbach, Venkat Subramanian, and Eric Sun. 2017. Ground: A Data Context Service. In *CIDR*. Available from http://cidrdb.org/cidr2017/papers/p111-hellerstein-cidr17.pdf.

[22] Joseph M. Hellerstein, Vikram Sreekanti, Joseph E. Gonzalez, James Dalton, Akon Dey, Sreyashi Nag, Krishna Ramachandran, Sudhanshu Arora, Arka Bhattacharyya, Shirshanka Das, Mark Donsky, Gabriel Fierro, Chang She, Carl Steinbach, Venkat Subramanian, and Eric Sun. 2017. Ground: A Data Context Service. In *CIDR*.

[23] Robert Ikeda, Hyunjung Park, and Jennifer Widom. 2011. Provenance for generalized map and reduce workflows. (2011).

[24] Matteo Interlandi, Kshitij Shah, Sai Deep Tetali, Muhammad Ali Gulzar, Seunghyun Yoo, Miryung Kim, Todd D. Millstein, and Tyson Condie. 2015. Titian: Data Provenance Support in Spark. *PVLDB* 9, 3 (2015), 216–227. Available from http://www.vldb.org/pvldb/vol9/p216-interlandi.pdf.

[25] Zachary G. Ives, Craig A. Knoblock, Steven Minton, Marie Jacob, Partha Pratim Talukdar, Rattapoom Tuchinda, José Luis Ambite, Maria Muslea, and Cenk Gazen. 2009. Interactive Data Integration through Smart Copy & Paste. In *CIDR*.

[26] Zachary G. Ives, Zhepeng Yan, Nan Zheng, Joost Wagenaar, and Brian Litt. 2015. Looking at Everything in Context. In *CIDR*.

[27] Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. 2010. Querying Data Provenance. In *SIGMOD*.

[28] Alfons Kemper and Guido Moerkotte. 1990. Advanced Query Processing in Object Bases Using Access Support Relations. In *VLDB*. San Francisco, CA, USA, 290–301.

[29] David Koop and Jay Patel. 2017. Dataflow notebooks: encoding and tracking dependencies of cells. In *9th USENIX Workshop on the Theory and Practice of Provenance (TaPP 17). USENIX Association.*

[30] Santosh Kumar, Gregory Abowd, William T Abraham, Mustafa al'Absi, Duen Horng Chau, Emre Ertin, Deborah Estrin, Deepak Ganesan, Timothy Hnat, Syed Monowar Hossain, et al. 2017. Center of Excellence for Mobile Sensor Data-to-Knowledge (MD2K). *IEEE Pervasive Computing* 16, 2 (2017), 18–22.

[31] Brian Litt and Zachary Ives. 2011. The International Epilepsy Electrophysiology Database. In *Proceedings of the Fifth International Workshop on Seizure Prediction.*

[32] Brian Litt, Greg Worrell, and Zachary G. Ives. [n. d.]. The International Epilepsy Electrophysiology Portal. www.ieeg.org.

[33] Dionysios Logothetis, Soumyarupa De, and Kenneth Yocum. 2013. Scalable lineage capture for debugging disc analytics. In *Proceedings of the 4th annual Symposium on Cloud Computing*. 17.

[34] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. 2006. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience* (2006), 1039–1065.

[35] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo I. Seltzer. 2006. Provenance-Aware Storage Systems. In *USENIX Annual Technical Conference, General Track*. 43–56.

[36] Leonardo Murta, Vanessa Braganholo, Fernando Chirigati, David Koop, and Juliana Freire. 2014. noWorkflow: capturing and analyzing provenance of scripts. In *International Provenance and Annotation Workshop*. Springer, 71–83.

[37] National Center for Biotechnology Information. [n. d.]. GenBank. Available from www.ncbi.nlm.nih.gov/GenBank/.

[38] Lucila Ohno-Machado, Susanna-Assunta Sansone, George Alter, Ian Fore, Jeffrey Grethe, Hua Xu, Alejandra Gonzalez-Beltran, Philippe Rocca-Serra, Anupama E Gururaj, Elizabeth Bell, et al. 2017. Finding useful data across multiple biomedical data repositories using DataMed. *Nature genetics* 49, 6 (2017), 816.

[39] T. Oinn, M. Greenwood, M. Addis, N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. 2006. Taverna: lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience* 18, 10 (2006), 1067–1100.

[40] Jeff Pasternack and Dan Roth. 2013. Latent credibility analysis. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, 1009–1020.

[41] Fernando Perez and Brian E Granger. 2015. Project Jupyter: Computational narratives as the engine of collaborative data science. *Retrieved September* 11 (2015), 207.

[42] Tomas Petricek, James Geddes, and Charles Sutton. 2018. Wrattler: Reproducible, live and polyglot notebooks. In *10th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2018)*. USENIX Association.

[43] Rakesh Pimplikar and Sunita Sarawagi. 2012. Answering Table Queries on the Web using Column Keywords. *PVLDB* 5, 10 (2012), 908–919.

[44] Lucian Popa, Yannis Velegrakis, Renée J. Miller, Mauricio A. Hernández, and Ronald Fagin. 2002. Translating Web Data.. In *VLDB*.

[45] Christopher J. Rossbach, Yuan Yu, Jon Currey, Jean-Philippe Martin, and Dennis Fetterly. 2013. Dandelion: A Compiler and Runtime for Heterogeneous Systems. In *SOSP*. ACM, New York, NY, USA, 49–68. https://doi.org/10.1145/2517349.2522715

[46] SciDB Development Team. 2010. Overview of SciDB. In *SIGMOD*.

[47] Manolis Stamatogiannakis, Hasanat Kazmi, Hashim Sharif, Remco Vermeulen, Ashish Gehani, Herbert Bos, and Paul Groth. 2016. Trade-offs in automatic provenance capture. In *International Provenance and Annotation Workshop*. Springer, 29–41.

[48] Partha Pratim Talukdar, Marie Jacob, Muhammad Salman Mehmood, Koby Crammer, Zachary G. Ives, Fernando Pereira, and Sudipto Guha. 2008. Learning to Create Data-Integrating Queries. In *VLDB*.

[49] Wei Wang, Chen Wang, Yongtai Zhu, Baile Shi, Jian Pei, Xifeng Yan, and Jiawei Han. 2005. Graphminer: a structural pattern-mining system for large disk-based graph databases and its applications. In *SIGMOD*. ACM, 879–881.

[50] Eugene Wu, Samuel Madden, and Michael Stonebraker. 2013. SubZero: A fine-grained lineage system for scientific databases. In *ICDE*. 865–876. https://doi.org/10.1109/ICDE.2013.6544881

[51] Zhepeng Yan, Nan Zheng, Zachary Ives, Partha Talukdar, and Cong Yu. 2013. Actively Soliciting Feedback for Query Answers in Keyword Search-Based Data Integration. *PVLDB* (2013).

[52] Yi Zhang, Herodotos Herodotou, and Jun Yang. 2009. RIOT: I/O-efficient numerical computing without SQL. *arXiv preprint arXiv:0909.1766* (2009).

[53] Nan Zheng, Abdussalam Alawini, and Zachary G. Ives. [n. d.]. Fine-Grained Provenance for ETL and Alignment Tasks. Submitted for publication.