Juneau: Data Lake Management for Jupyter*

Yi Zhang
University of Pennsylvania
Philadelphia, PA 19104
yizhang5@seas.upenn.edu

Zachary G. Ives
University of Pennsylvania
Philadelphia, PA 19104
zives@cis.upenn.edu

ABSTRACT

In collaborative settings such as multi-investigator laboratories, data scientists need improved tools to manage not their data *records* but rather their *data sets and data products*, to facilitate both provenance tracking and data (and code) reuse within their data lakes and file systems. We demonstrate the Juneau System, which extends computational notebook software (Jupyter Notebook) as an instrumentation and data management point for overseeing and facilitating improved dataset usage, through capabilities for indexing, searching, and recommending "complementary" data sources, previously extracted machine learning features, and additional training data. This demonstration focuses on how we help the user find related datasets via *search*.

PVLDB Reference Format:
Yi Zhang and Zachary G. Ives. Juneau: Data Lake Management for Jupyter. *PVLDB*, 12(12): 1902-1905, 2019.
DOI: https://doi.org/10.14778/3352063.3352095

1. INTRODUCTION

As data science has emerged as its own discipline, one of the most important activities is *exploratory data analysis*: taking data sets, combining and cleaning them, formulating an initial hypothesis, evaluating this hypothesis (via queries, visualization, etc.), and repeating.

Sites such as `kaggle.com` provide many illustrative examples of this type of activity, and in fact their goal is to foster the development of new algorithms and classifiers through such exploratory analysis. For Kaggle, but also beyond, data scientists are increasingly leveraging *computational notebook* software (Jupyter, Apache Zeppelin, RStudio) for such tasks. In essence, such tools are the fledgling "integrated development environments" of the data science era: interactive, Web-driven, documented-oriented environments for performing computational steps and seeing output, which can link to computational tools such as Python and Pandas, Apache Spark, TensorFlow, and so on; as well as to data visualization and web technologies.

*Funded in part by NIH grant 1U01EB020954-01 and NSF grants 1547360 and 1640813.

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3352063.3352095

A challenge lies in transitioning from exploratory data analysis, possibly over a small amount of data, to something that can be regularized into a production workflow with full reproducibility and much larger datasets. Towards this goal, recent work has proposed using notebooks as a way of encoding repeated computational workflows [9], and others have developed extensions to ensure the code within notebooks is fully versioned and reproducible [10, 1, 6].

However, we argue that the next step must be to look not at notebooks as *documents* of code steps that access and produce data files — but rather as compilations of (possibly shared, possibly parameterized) computational steps operating on objects in a data lake. We seek to *accelerate and regularize data science tasks* by finding and recommending data *related to* current objects of interest to the user. We do this by tracking the *relationships* between data sets, data products, and code [5]. With the appropriate indexing and search capabilities, data import and data cleaning steps are made visible to future users to be *reused*; data scientists may find other related datasets with similar history provenance; users are able to query, based on a given source table or intermediate result, whether someone else has already linked two datasets or extracted sets of features. Ultimately, just as shared libraries and open-source repositories have accelerated and improved software engineering — reusable datasets, schemas, and computational workflow steps may improve the quality of data engineering.

In this demonstration, we present a prototype of JUNEAU system, which provides these capabilities. Our demonstration illustrates how indexing, searching, and reusing tabular data are supported for tabular, CSV, and relational datasets. JUNEAU addresses scientists' need to search for prior tables (and related code) not merely by keyword, but by querying using an existing table *and its provenance*, to find other related tables. Within the Jupyter environment, users may select a table (dataframe) and directly search for related tables for different purposes.

Motivating use cases. We outline the four use cases for finding related tables.

EXAMPLE 1.1 (AUGMENTING TRAINING DATA). *Often, data is captured in multiple sessions (perhaps by multiple users) using the same sensor device or tool. Given a table from one such session, the user may wish to* augment *his or her data, to form a bigger training or validation set for a machine learning algorithm.*

EXAMPLE 1.2 (LINKING DATA VIA ONTOLOGIES). *Particularly in the life sciences, records in one database may have identifiers (e.g., "accession numbers") linking to entries in another database or ontology. Such entries may transitively reference other entries, and each brings in additional fields that may be useful. It*

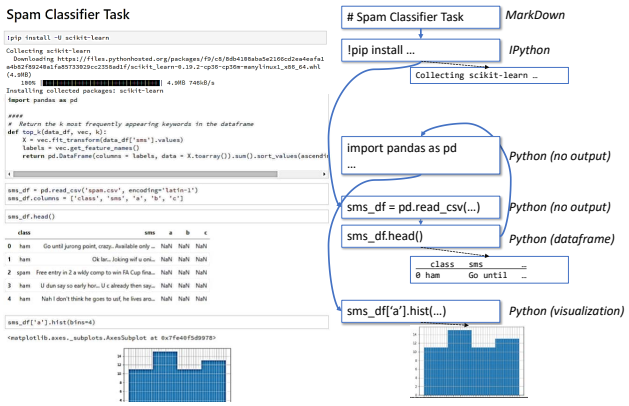

Figure 1: Example of a computational notebook, abstracted into a data model

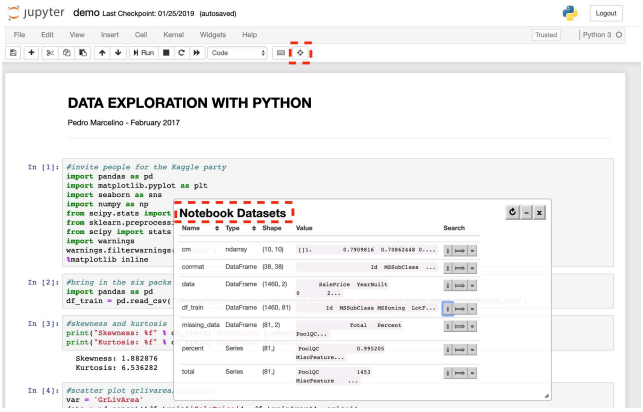

Figure 2: Demo Step 1. The user clicks on the Show Notebook Datasets button in the tool bar, and JUNEAU lists all datasets (tables) in the current notebook.

can be helpful for users to know about such links that are implicit in the data.

EXAMPLE 1.3 (AUGMENTING FEATURES). *Another common task for data scientists is to find additional or alternative features for the given data instances that may lead to a better performance. Especially in the collaborative setting, one data scientist may perform a specific feature engineering on a data set, while another may do it in a different way. It can be helpful for data scientists to be recommended with other feature engineering possibilities.*

EXAMPLE 1.4 (FINDING WORKFLOWS FOR DATA). *Given a widely used and related table, a data scientist may want to see examples of how the table is loaded or cleaned, what analysis have been performed on it, and so on. Generally, this requires us to* search for workflows using the table *or related tables, potentially featuring specific operations.*

Related work. Our work relates to both data link discovery systems [2] and those for discovery of related and unionable data [4, 7, 8]. Our novelty is in incorporating data provenance information, code, and a more general notion of search – all from within a computational notebook environment. Our work directly builds upon techniques for reproducible notebooks [10, 1, 6].

2. THE JUNEAU SYSTEM

2.1 Preliminaries: Computational Notebooks

A Jupyter Notebook, or more broadly, a computational notebook, is an editable document in which data scientists may add *cells* with code (to be executed in a *kernel* sitting outside the notebook with its own internal state) and text (in the MarkDown language). As the code in each cell is executed, its output, whether scalar, tabular, matrix, or visualization, may be directly embedded in the notebook. The web-based Jupyter Notebook environment enables interactive data analysis.

Figure 1 shows a sample computational notebook, as well as a higher-level abstraction of the notebook. The notebook's content is a sequence of *cell*s, which may be *code cell*s (conceptually, these are in effect code modules in a workflow defined by the notebook), or *MarkDown cells*, which let the user document the computational process using rich text. As these are entered by the user, code cells are executed and their output is injected into the notebook, or MarkDown cells are translated into formatted text and images and embedded into the notebook.

Importantly, standard computational notebook software does not reason about how cells produce side effects in terms of state passed from one cell to another, or about files that are created (which may be read by a future cell execution that may even occur in another notebook). Moreover, computational notebooks do not fully preserve either the history of cell versions and outputs, nor the order in which cells were executed — which may result in non-reproducible notebooks. However, several recent projects have introduced *reproducible notebooks* [10, 1, 6]. Building upon those ideas, we have replaced the notebook software's storage layer with JUNEAU's data lake management system to index and store external files as well as intermediate and final data products produced within notebooks. Additionally, we have enhanced the notebook software interface itself.

2.2 Juneau Architecture and Components

The JUNEAU System replaces Jupyter Notebook's back-end and extends it user interface. Our back-end "data lake management" subsystem integrates relational and key-value stores to capture and index (1) any external files loaded by the notebooks; (2) intermediate data products produced by computational steps (cells) within the notebooks; (3) versioned cell content and notebook content, as in the right-hand side of Figure 1; (4) indices for rapidly retrieving tables and their provenance.

We illustrate the basic architecture and functionality in Figure 4. As in the existing Jupyter Notebook software, the notebook interface interacts with a *kernel* (language interpreter) every time the user executes a cell. The cell contents are executed in the kernel, thus updating state in the kernel as well. JUNEAU fetches any new or changed tables (dataframes) from the kernel after each step, and it imports and indexes those in the backend.

The user may interactively select any table within the notebook, and query the JUNEAU search engine for other tables already stored and indexed in the data lake which are related to the selected item. As we described in the introduction, users often want to search for other related tables using an existing table as a model, and possibly adding other filter criteria such as author, attribute name or content, or the name of a computational process that was involved in the provenance of the search result. The search may not purely be based on whether other tables have a common schema or joinable fields, but may also consider similarity of computational (provenance) steps.

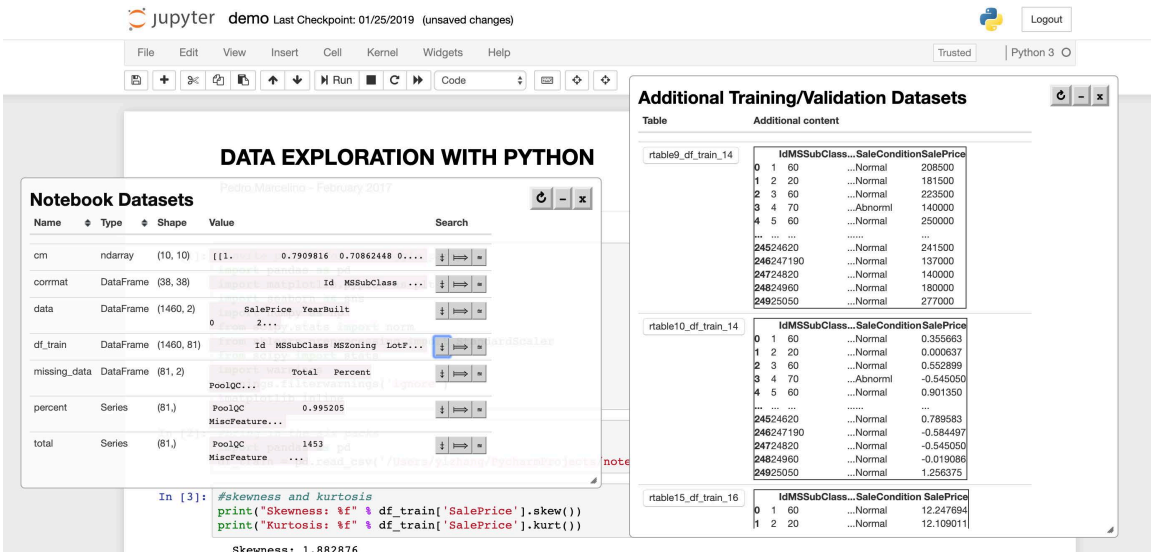

Figure 3: Demo Step 2. The user selects a table with a search mode, and the system will rapidly return a ranked list of tables that are related. In this example, the user selects "D", which means the user is looking for augmenting training or validation data.

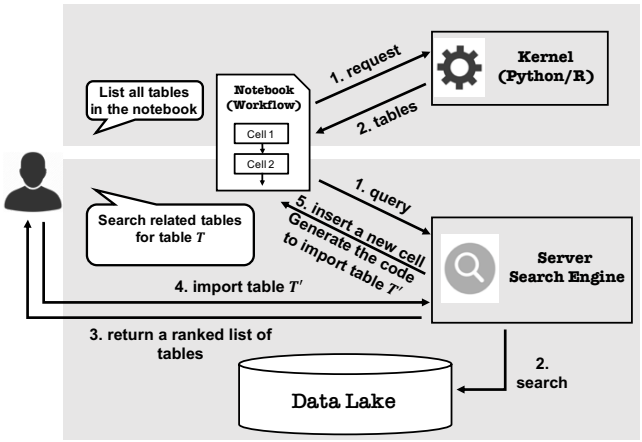

Figure 4: The architecture of JUNEAU

Our main technical focus has been on JUNEAU's search and indexing capabilities. We target three classes of table search, which each address a different task.

Additional Training/Validation Data. Data scientists accumulate additional data, e.g., as additional training or test sets. The data usually shares the same schema and similar computational steps, while having minimal row overlap.

Linkable tables. The second class of search involves discovering "linkable tables." The most obvious version of this comes from data integration: we want to find conceptually related base tables in a dataset, with which we can join [2].

Alternative features. In many data science settings, the tables of greatest interest to the user may not be base data — but rather other users' *derived tables* that take rows from the base data, and extract features or do additional processing for each row (e.g., it adds a record-linking attribute to describe a mapping to a row in another table). Such tables should be linkable by the query tables, and the user would prefer those with minimal column overlap.

Searching In JUNEAU. To support the search capabilities for different use cases, we design a ranking function that mainly incorporate the following three ranking components. To further make the search process efficient, JUNEAU uses a novel extension to the top-k threshold algorithm [3] (TA), which adaptively prioritizes its ranking components.

Schema Mapping. To compute the *relatedness* between tables, we first map between columns within the query and target tables. The schema mapping is detected based on the similarity between instances of columns from two tables, and we speed it up via sketches (as done by Aurum [4]) and reusing the mapping already detected.

Key-based Row Mapping. Once we have discovered a candidate schema mapping, we find a row-to-row mapping, which generates the row similarity for our ranking. This mapping is detected based on the predicted key-to-key and key-to-foreign key relationships for detecting both linkable tables and alternative features.

Provenance Similarity. Provenance similarity augments data that may have minimal row overlap, but is semantically similar and useful for training or testing. To compute the provenance similarity, we represent the workflow (notebook) producing the table as a dataflow graph between tables, and approximately match the tables by their content, position and dependencies. Then the similarity can be computed by some edit distance between dataflow graphs.

Our TA-based algorithm starts by computing a list of matches along the most efficient components, then estimates the others, so that we can avoid computing computation-heavy components for most unrelated tables. We also use approximation for further performance improvement.

3. DEMONSTRATION DESCRIPTION

Our demonstration is based on common data science use cases. On the Kaggle data science competition site, data scientists are given new datasets and tasks. They frequently share their notebook code — which starts with simple "eyeballing" and wrangling of data, followed by various exploratory analyses to understand the data relationships, before a final approach is determined and pursued. If one browses the Kaggle site, many notebooks exhibit many of the same computational steps.

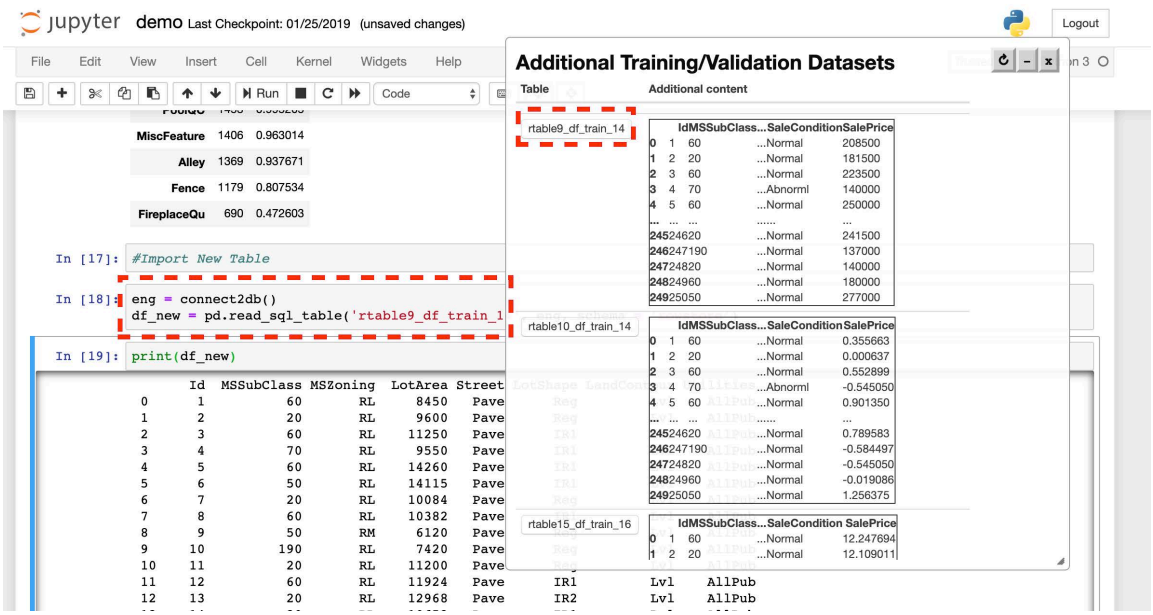

Figure 5: Demo Step 3. The user clicks on a table of interest, and JUNEAU adds a new cell to the notebook that loads the table.

JUNEAU accelerates the process for a new data scientist to come "up to speed." We leverage a corpus of notebooks crawled from public Kaggle competitions (Housing prices, biomechanical features of orthopedic patients, and several others) and store, in the JUNEAU data lake, all intermediate tables generated by those notebooks. Our demonstration illustrates how the data scientist can begin by loading a starter notebook that reads the Kaggle datasets. From this, JUNEAU can let them quickly see excerpts from all tables (lists, arrays and dataframes) in the current notebook (see the red box of Figure 2).

Subsequently, the user will identify which dataset he or she wishes to start with, and will search by task: looking for *additional training and validation data* , *linkable tables* or *additional, extracted features*.

Figure 3 illustrates a set of matches when the user searches for additional training data. The rankings adjust based on usage patterns, as some tables are incorporated into larger numbers of notebooks than others. (Table popularity is one element of our ranking algorithm.)

Once the user finds a table that would be useful within the notebook, she or he can select whether to import the table as a materialized artifact, or to import the *sequence of cells* (workflow) that were responsible for generating the table (Figure 5).

Our demonstration subsequently presents common options of actions to apply to the resulting table: to train and test a classifier (using k-fold validation), visualize the data, or cluster the data. We track how much work was saved by the search capability (how many lines and cells would have been necessary), and we also show how many Kaggle competitors performed a similar task.

This initial demonstration of JUNEAU shows that (1) an important data science task can be addressed by database technologies, (2) JUNEAU offers a visual and intuitive tool for data lake management, and (3) it provides a new type of content- and provenance-based search and indexing technology to support data lake management for data science.

4. REFERENCES

[1] L. A. Carvalho, R. Wang, Y. Gil, and D. Garijo. Niw: Converting notebooks into workflows to capture dataflow and provenance. In *Conference on Knowledge Capture (K-CAP)*, 2017.

[2] D. Deng, R. C. Fernandez, Z. Abedjan, S. Wang, M. Stonebraker, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, and N. Tang. The data civilizer system. In *CIDR*, 2017.

[3] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of computer and system sciences*, 66(4):614–656, 2003.

[4] R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. Aurum: A data discovery system. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1001–1012. IEEE, 2018.

[5] Z. G. Ives, S. Han, Y. Zhang, and N. Zheng. Data relationship management systems. 2019.

[6] D. Koop and J. Patel. Dataflow notebooks: encoding and tracking dependencies of cells. In *TaPP*, 2017.

[7] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas. Exemplar queries: Give me an example of what you need. *PVLDB*, 7(5):365–376, 2014.

[8] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller. Table union search on open data. *PVLDB*, 11(7):813–825, 2018.

[9] nteract team. Papermill: Parameterize, execute, and analyze notebooks. `https://papermill.readthedocs.io/en/latest/`, 2018.

[10] T. Petricek, J. Geddes, and C. Sutton. Wrattler: Reproducible, live and polyglot notebooks. In *TaPP*. USENIX Association, 2018.