MLProvLab: Provenance Management for Data Science Notebooks

Dominik Kerzel[1], Birgitta König-Ries[2], Sheeba Samuel[3]

Abstract: Computational notebooks are a form of computational narrative fostering reproducibility. They provide an interactive computing environment where users can run and modify code and repeat the exploration, providing an iterative communication between data scientists and code. While the ability to execute notebooks non-linearly benefits data scientists for exploration, the drawback is that it is possible to lose control over the datasets, variables, and methods defined in the notebook and their dependencies. Thus, in this process of user interaction and exploration, there can be a loss of execution history information. To prevent this, a possibility is needed to maintain provenance information. Provenance plays a significant role in data science, especially in facilitating the reproducibility of results. To this end, we developed a provenance management tool to help data scientists track, capture, compare, and visualize provenance information in notebook code environments. We conducted an evaluation with data scientists, where participants were asked to find specific provenance information from the execution history of a machine learning Jupyter notebook. The results from the performance and user evaluation show promising aspects of provenance management features of the tool. The resulting system, MLProvLab, is available as an open-source extension for JupyterLab.

Keywords: Data Science; Information Extraction; Provenance; Jupyter Notebook; Reproducibility

1 Introduction

Data science and machine learning (ML) techniques significantly impact the scientific community in developing relevant and practical applications for society. With the rapid publication of results in the data science and ML field, it is increasingly important for scientists also to be able to reproduce and recreate results. Jupyter notebook [KR+16] is one of the adopted approaches researchers use to publish results of their data science and ML projects to enable reproducible computational research. The notebooks are computational narratives that data scientists widely use in multiple ways for scientific computing, exploration, tutorials, documentation, interactive manuals, publications, etc. This is possible because the notebooks encapsulate code and explanatory text, computational results, visualizations, etc., in a single document. In addition to being a stand-alone tool, it is also integrated with different data science platforms like Kaggle, Colab notebooks, etc. As a result of exploration

[1] Friedrich Schiller University Jena, Germany dominik.kerzel@uni-jena.de
[2] Heinz-Nixdorf Chair for Distributed Information Systems, Friedrich Schiller University Jena, Germany Michael Stifel Center Jena birgitta.koenig-ries@uni-jena.de
[3] Heinz-Nixdorf Chair for Distributed Information Systems, Friedrich Schiller University Jena, Germany Michael Stifel Center Jena sheeba.samuel@uni-jena.de

doi:10.18420/BTW2023-66

and constant changes done in a data science pipeline, it is essential to understand how the results are derived under which choices and assumptions of researchers. Provenance plays a significant role to record and reference the history of results. This, in turn, helps data scientists to enable reproducibility [SK21]. However, scientists do not have direct access to the history of their frequent experimentation in these notebooks. The rapid development of data science and ML methods and algorithms results in new releases of libraries and modules. As a result, running old notebooks without knowing the versions of libraries and modules can cause execution errors and incompatibility issues. Another significant barrier is the possibility of running notebooks non-linearly. It becomes difficult to reproduce and get the same or close-by results without understanding how each cell and the variables defined in them are dependent on each other.
To address these issues, in this paper, we present MLProvLab as an extension to JupyterLab[4], providing provenance management for reproducibility. This tool provides significant benefits for data scientists to track, compare, manage, and visualize provenance information of their computational experiments written in Jupyter notebooks. We can track, at runtime, the datasets, variables, libraries, and functions used in the notebook and their dependencies between cells. This is also visualized as a provenance dependency graph with temporal information. We evaluated its efficiency and features through a performance test and a user evaluation with 15 participants using practical tasks. The study shows that these 15 data scientists using MLProvLab for the first time correctly answered an average of 82% of the tasks they were provided in a machine learning notebook which was totally new to them and consisted of 55 executions.

2 Background and Related Work

Though data scientists use multiple tools and software for their computational tasks, writing code using programming languages like Python and R is common in data science and ML. The open-source libraries like Scikit-Learn[5], PyTorch[6], Numpy[7], etc., provide accessible and reusable tools for data analysis and are heavily used for data science, ML, and deep learning applications. Jupyter notebooks that support over 40 programming languages, including Python and R, are widely used by millions of scientists. This is clearly seen in the availability of millions of notebooks on GitHub. Hence, in this research, we focus on the provenance management of Jupyter notebooks, with specific attention on providing support for the reproducibility of data science workflows.
Tools for capturing provenance from scripts and programs at different levels of granularity have been actively developed [Da12; Mc15; Pi15]. The noWorkflow tool [Pi15] is one such tool that captures the definition, deployment, and execution provenance of Python scripts. With the current wide adoption of Jupyter notebooks [KR+16], research works have

[4] https://jupyterlab.readthedocs.io
[5] https://scikit-learn.org
[6] https://pytorch.org/
[7] https://numpy.org/

also focused on tracking provenance from computational notebooks [Ca17; He19; Ho14; KM18; KP17; Ma21; PGS18; Pi15; Sh23; SK18; Wa20; We19]. Prov-o-matic provides a provenance-tracking extension for older versions of IPython Notebooks, which saves the provenance traces to Linked Data file [Ho14]. Another approach to track provenance in computational notebooks is by integrating noWorkflow [Pi15]. As a result, the features provided by noWorkflow are available in the IPython notebooks. However, this approach allows a python script to be run from inside IPython notebooks capturing the provenance of scripts instead of notebooks. Systems like Verdant [Ke19] are more closely aligned with MLProvLab. Verdant helps data scientists examine the execution history and notebook events. However, we analyze the code and provide artifacts used in the code and the dependencies between the cells based on the artifacts.

Recent approaches have also developed custom Jupyter kernels to trace runtime user interactions and automatically manage the lineage of cell execution [KP17; Ma21]. In their approach, the tool is developed as a separate Jupyter kernel, allowing users to update all cells affected by a change in a cell. This is possible by adding unique and persistent identifiers to each cell and providing references to results in other cells. This is different from our approach as these approaches [KP17; Ma21] introduce changes to the kernel and requires its installation.

Recent works have also focused on the provenance and model management of data science and ML pipelines beyond computational notebooks. An overview of conceptual, data management, and engineering challenges in the ML model management is given in [Sc18]. One of the data management challenges concerning the provenance management of ML is automatically tracking and querying model metadata. Several tools have been developed as metadata capturing systems in recent years [Or20; Va16; Za18]. ModelDB [Va16] provides a feature to manage ML models with metadata logging of metrics, artifacts, tags, and user information. Some systems track detailed provenance data by depending on the users to understand their complex schema and integrate their code with the corresponding API provided by the system [Sc17]. These provenance-capturing systems generally require users to actively configure their code, e.g., by annotating functions, hyperparameters, and operations. Due to the extra time and effort required, users may omit to configure and annotate their code. Therefore, tools that automatically extract and manage metadata are preferable to systems that require human intervention.

It is essential to provide provenance management without changing the code environment for the user. It is also essential that such platforms provide metadata management to all their users, irrespective of their skills and experience in data science. JupyterLab is a great basis for such projects, as shown in other works [Ke19]. Hence, in this paper, we target the users of JupyterLab and allow automatic provenance extraction from data science notebooks.

3 MLProvLab

Kerzel et al. [KSK21] describe the use case, challenges, and design goals of our data science and ML provenance management tool to automatically expose the metadata. Based

on the design goals, we present MLProvLab, a provenance management tool to track, manage, compare, and visualize the provenance of data science notebooks. The tool is available as an open-source extension for JupyterLab[8]. MLProvLab is composed of two components. A Python backend component provides event listeners for user interactions, an Abstract Syntax Tree (AST) generator for analyzing the code, and a core messaging plugin to request information from the kernel and notebook panel. On the visualization side, MLProvLab provides a Javascript frontend component that captures user interactions, renders visualization, and generates provenance graph. The tool contains provenance capture, visualization, comparison, and export modules.

Provenance Capture. The provenance capture module of MLProvLab collects and stores the provenance of a user session triggered by the start of the kernel. We call the lifetime of a kernel an *epoch*. For every new kernel, the provenance of epochs is created and stored in the notebook metadata. The tool defines event listeners for different user actions like the execution, addition, and deletion of a cell. When a code cell is executed, the cell content is returned to the backend. The executed code is then analyzed using Abstract Syntax Tree (AST) and string pattern matching techniques to get data provenance. We capture information on the definition and usage of variables, functions, and classes. The import statements are also tracked to extract information on the libraries and modules used and their version information. Additional operations are performed to find data sources for ML provenance management using string matching. In summary, the MLProvLab tracks and manages every variable declared in the cell, the dependencies of variables that are not defined in the evaluated cell, used datasets and the corresponding variables, imported libraries, and modules, etc.

Provenance Visualization. For the provenance visualization module, the MLProvLab uses a provenance graph to visualize the provenance of the notebook, including the execution order of cells and the data dependencies between cells. The tool can be invoked using the 'MLProvLab' button in the notebook toolbar. Figure 1 shows the provenance visualization graph of a sample ML notebook. The data sources and execution provenance are shown in the graph. A node is created in the graph for every cell in the notebook. Edges show the dependencies between cells using variables or methods declared in a cell. The outgoing edge from a node indicates that a data source was defined and is used in the other corresponding node. The colors of the nodes and edges represent their status. Cells that are colored orange represent cells with data sources, and green represents cells with output. Cell half colored with orange and green show that the cell contains both datasets and output. Users can move the sliders at the bottom of the panel to see the history of the changes and runs performed by the user. The 'Epoch' slider provides the history of the execution of the Jupyter Notebook every time a new user session of the kernel is started. The 'Execution' slider depicts the execution history of the Jupyter Notebook every time an event on the notebook cell is registered. Correspondingly, the information for the execution environment, datasets, and libraries used are shown to the user for the selected execution. The tool also shows the number of user sessions, executions, and execution time. MLProvLab also provides a

[8] https://github.com/fusion-jena/MLProvLab/

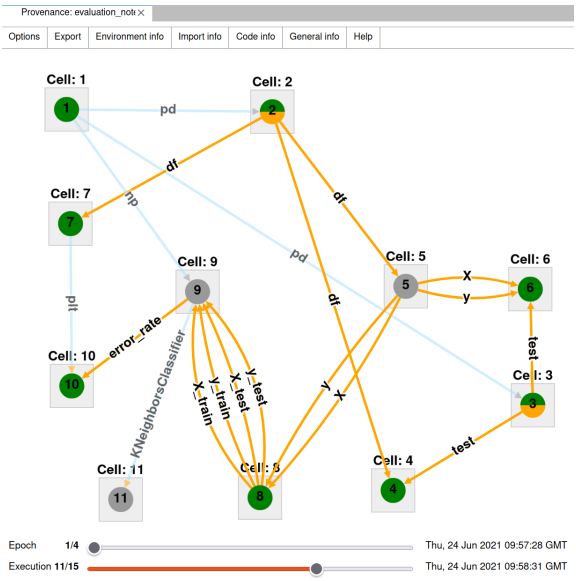

Fig. 1: Provenance Execution Graph in MLProvLab. The notebook cells are depicted as vertices of the graph. Green nodes in the graph show cells with any output type, orange show cells with data source, grey shows cells where no data source or output is detected. Edges in the graph show the dependencies between cells. The footer allows to slide the execution history of the notebook.

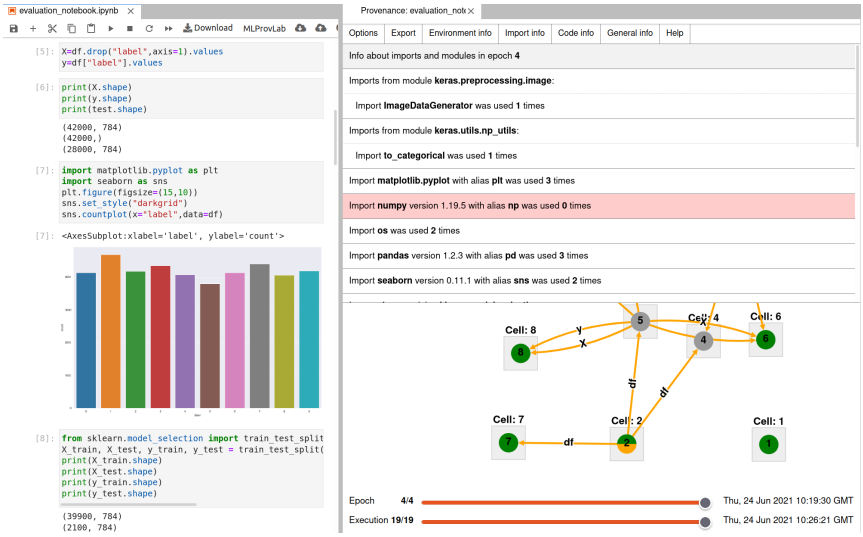

Fig. 2: Libraries and modules used in the notebook in the 4th epoch.

general menu with options to customize the graph to get additional provenance information. Figure 2 shows the *Import Info* menu in MLProvLab. It shows the information about the imported libraries, the module used in the notebook, and their version information. The libraries which are imported but not used are marked as red. To visualize the definition provenance, users can click on a node and open a radial context menu. This gives detailed information on the used datasets, functions, variables, outputs, etc. Users can also compare the definition provenance from previous runs. The graph is built using Cytoscape.js[9].

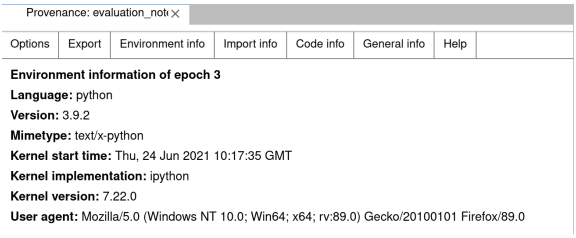

Fig. 3: Execution environment information of the notebook

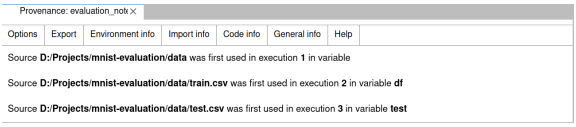

Fig. 4: Information on the datasets used in the notebook

Figure 3 and 4 shows the information of the execution environment of the notebook and the datasets used in the code, respectively. The execution environment of the notebook provides information on the programming language, kernel, operating system, and the versions of the selected epoch. Similarly, the *General info* provides information on the datasets used in each execution with their variable name.

Provenance Comparison. In the provenance comparison module of MLProvLab, the changes made to a notebook cell can be examined by users (Fig. 5). Users can select the execution of previous ML experiments and compare it with the current execution. We use the react-diff-view[10] component to visualize the differences.

Provenance Export. The provenance export module of MLProvLab allows users to export the collected provenance information of the notebook. Users can also clear the provenance history. However, users are given an alert to export the provenance before removing the provenance history from the notebook. This information is currently available in JSON format. For semantic interoperability, we plan to make this information available in other formats, including JSON-LD, RDF, etc.

[9] https://cytoscape.org/

[10] `https://github.com/otakustay/react-diff-view`

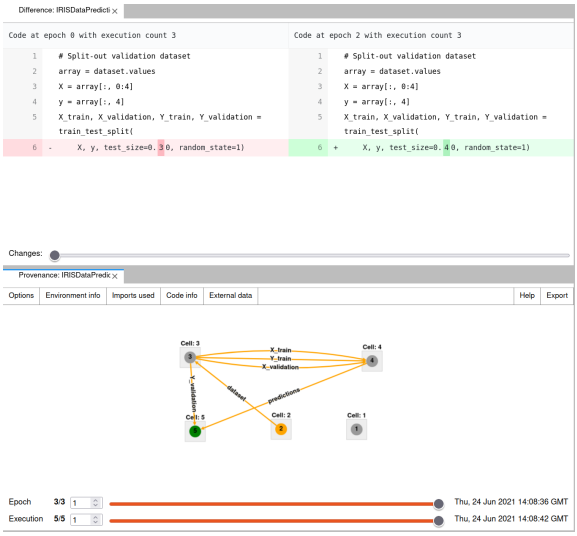

Fig. 5: Code difference between 2 different epochs

4 Evaluation of MLProvLab

The primary goal of our evaluation was to gather information about how the features of MLProvLab assist data scientists in the provenance management of Jupyter notebooks. As a result, we conducted a performance evaluation to test how MLProvLab handles notebooks with different numbers of cells. We also conducted a user evaluation to gather information on the impression of MLProvLab and test how the provenance management helps them understand the notebooks.

4.1 Performance Evaluation

We did a performance test with two different notebooks, one with 25 notebook code cells and the other with 100 code cells. Based on the study of analyzing 1 million computational notebooks on GitHub [RTH18], the typical number of total cells in a notebook range from 25-30. Notebook with more than 100 cells is infrequent. Hence, we selected notebooks with 25 and 100 cells. For the performance test, we used notebooks that calculate the Fibonacci number given an input. We defined ten variables, with each calculating a Fibonacci number for ten. These ten variables were assigned to each other and were repeated in the other cells. This was done to get multiple dependencies between every cell. As a result, each node in the provenance graph had ten outgoing edges to the other node.
Table 1 shows the status of the notebook run scenarios used for the evaluation. We evaluated six notebook run scenarios with different configurations. For example, in Notebook Run

Scenario A, the notebook with 100 cells is executed without enabling and updating the MLProvLab extension and provenance graph. While in Scenario B, the notebook has the same number of cells, with both MLProvLab extension and provenance graph enabled, and in Scenario C, MLProvLab enabled but without updating the provenance graph. These scenarios are also repeated with a notebook with 25 cells. We measured the execution time of the notebook in each scenario. Table 2 shows the results from the performance test for each notebook scenario. Each notebook is tested with different numbers of epochs (1, 2, 3, 4, 5, 10, and 20). The execution time in seconds for each notebook run scenario for each epoch count is shown.

Notebook Run Scenario	Count of code cells	MLProvLab enabled	Provenance Graph Updated
A	100	no	no
B	100	yes	yes
C	100	yes	no
D	25	no	no
E	25	yes	yes
F	25	yes	no

Tab. 1: Definition of the notebook run scenarios

Epoch	Time in seconds					
Notebook Run Scenario	A	B	C	D	E	F
1	2.46	89.06	4.13	1.02	5.83	1.66
2	2.39	90.94	5.58	1.05	6.64	1.75
3	2.50		7.37	1.02	6.24	1.54
4	2.47		8.56	1.01	6.21	1.99
5	2.48		10.05	1.02	6.37	2.21
10			21.02		7.97	2.62
20			24.18		8.11	3.85

Tab. 2: Performance evaluation of MLProvLab

Notebook size	Epochs	Count of notebook code cells
12.0 KB	0	25
2.5 MB	10	25
5.0 MB	20	25
36.0 KB	0	100
9.9 MB	10	100

Tab. 3: Cell count and size of evaluation notebooks

As seen in Table 2, the execution time of the notebooks increases as the number of epochs increases. For notebook scenario B, where there are 100 cells and both MLProvLab and the provenance graph are enabled and updated, we can see the overhead in loading the notebook. If one compares the runs with enabled and disabled extensions, one quickly sees

that a constant overhead is added. It is also noticeable that the execution times increase the more often the notebook is executed. This behavior is expected, as additional computations must be performed in the backend. Users working with notebooks with the average number of cells (around 25-30) are unaffected. However, working with notebooks with around 100 code cells with and without MLProvLab with multiple executions results in overhead in time. The overhead is due to the recomputation of the graph. However, due to this limitation, MLProvLab provides an option to disable the recalculation of the graph after every cell execution. This can further minimize the general overhead.
Table 3 shows the statistics based on the size of the notebooks and the number of execution. As expected, the size of the provenance information increases with the number of executions and code cell count. Currently, the captured provenance information is stored in the metadata of the notebook. This is located in a JSON object that has to be rewritten to disk after each update. Notebook containing the provenance information benefits users to share their intermediate and negative results, their choices and assumptions made during experimentation, etc. Currently, MLProvLab allows users to export the collected provenance data and then remove the provenance information if the notebook size gets too large. We plan to provide users an option to efficiently store the data, e.g., in SQLite database outside of JupyterLab.

4.2 User Evaluation

We present the materials and methods used and the results of the user evaluation conducted to get the impression of MLProvLab.
Participants. We used convenience sampling for the recruitment of participants. Participation in the user evaluation was voluntary. Forty participants responded to the survey, of which 36 agreed to the consent form and filled in their research background. However, only 15 participants finished the user evaluation and submitted their responses. We believe that this was due to the relative long time (around 25 minutes) needed to complete the tasks. Participants who read and agreed to the informed consent form and submitted their full responses were included in the final study. Of 15 participants, 14 have a computer science background, and 1 has physics. Seven undergraduate students, 2 Master students, 4 PhD Students, 1 PostDoc, and 1 Professor participated in the user evaluation.
Materials. We explored many publicly available data science notebooks to create a realistic provenance history for the experimentation. We also selected the evaluation notebook, which is not difficult for the participants to understand in minimum time. We used the Digit Recognizer problem from Kaggle, which uses the MNIST dataset[11]. We adapted the code and the resulting notebook contained 19 code cells with the provenance information collected from 4 epochs and 55 executions.
The questionnaire for the user evaluation was designed and developed using the following resources: (1) interviews conducted with the data scientists [SLK21] in the Werkstatt project

[11] https://www.kaggle.com/c/digit-recognizer

[Sa20] and (2) existing published literature on computational research reproducibility [Pi20]. The interviews and the existing literature provided insights into the challenges and problems faced by scientists and the provenance information required in reproducing published results of others in the context of data science and ML. The questionnaire was developed in English. A group of three researchers from computer science provided feedback on the length of the questionnaire, the priority and clarity of the defined questions, and technical issues in filling out the questionnaire. Based on the feedback, changes were made to the final version of the questionnaire.

The evaluation consisted of 26 questions grouped in 6 sections. The six sections are (1) Informed Consent Form (2) Research context of the participant (3) Testing MLProvLab with evaluation notebook (4) MLProvLab Introduction (5) Questions for Evaluation (6) General Impressions of MLProvLab. In the first section, we asked the consent from the respondents to participate in the evaluation. The informed consent form contained information about the study's background, purpose, procedure, voluntary participation, and contact information. Other than the informed consent form, none of the questions in the evaluation were mandatory.

In the second section, we asked about the research background of the participants. In addition to their current domain and position, we asked the participants whether they use Juypter Notebooks and machine learning in their work. In the third section, we asked the participants to open the evaluation notebook and in the following section, we introduced MLProvLab. We provided a short tour showing its features and how they worked. This help page included screenshots and annotations of each feature provided by the tool. In the fifth section, we provided the questions to answer based on the evaluation notebook using MLProvLab. This section included 13 questions. Some of the questions were either single-choice or multiple-choice questions. Here is a list of the questions:

Q1 Which version of the kernel was used in epoch '1'?

Q2 Which external modules were used in epoch '1'?

Q3 Are there any imported modules that were not used in epoch '3'?

Q4 Which one was the most used module in epoch '3'?

Q5 Which data sources were used in the notebook?

Q6 In which execution and epoch the following figure got printed?

Q7 Which version of seaborn was used?

Q8 Which versions of python were used in the notebook?

Q9 When was the notebook last executed?

Q10 Are there any differences in the python and kernel versions used in the notebook in different executions?

Q11 Which cells of the notebook in epoch '4' are dependent on the variable 'X_train'?

Q12 What is the accuracy score of the experiment in epoch '2' when RandomForestClassifier was used?

Q13 Has the train-test split ratio for the dataset changed during different executions?

In the last section, we asked the participants about their general impression of MLProvLab. In the first question, we asked how important is each MLProvLab module for the provenance management of computational experiments. We used a 5-point Likert scale for the answer options from *Very Easy* to *Very Difficult*. We also asked how easy it is to find provenance information on data science scripts using MLProvLab. In the next question, we asked the users to rate the perceived usefulness of MLProvLab. We asked whether they would like to use MLProvLab in their daily work. In the end, we provided an open-response question to participants to provide comments regarding the new features or changes they would like to see in MLProvLab. The average time taken to answer the evaluation questions in the notebook was 8 minutes. However, the average interview time, including the MLProvLab tour, was 28 min. The extra loading time of the Binder instance and the tour of MLProvLab could be some reasons for the long interview time.

The online evaluation was implemented using LimeSurvey[12]. The evaluation notebook is available in GitHub and was hosted using Binder[13]. Binder allows users to open the notebook with its execution environment, making the code and the extension (in this case, MLProvLab) available to everyone. We used a Jupyter notebook with Python version 3 to analyze the evaluation results. The source code and the results are available in GitHub repository[14].

Methods. We sent invitations for participation to the PhD, Master, and Bachelor students in the Fusion group of the Computer Science Department of the University of Jena, Germany, and the Werkstatt project's collaborating partners.

Results. Of 15 participants, 80% use Jupyter notebooks regularly or sometimes in their work. 66.67% of participants use Machine Learning regularly or sometimes in their work. Analyzing the results from the evaluation tasks of the notebooks, we see that 82% of answers to each question were correct, while 18% of answers were wrong. Three participants answered all the questions correctly. Eleven participants answered more than 60% of the questions correctly. However, the one participant who scored 46% did not attempt four questions and partially answered three correctly. For multiple-choice questions, we mark the answer correct only if the participants select all the right options. We observe that the multiple-choice questions were answered incorrectly, in particularly for Question **Q11**, where seven participants gave the wrong answer. Questions **Q1** and **Q8** were answered correctly by every participants, followed by Questions **Q3**, **Q7**, **Q10** and **Q12**. For Question **Q11**, none of the persons selected the wrong option, but all the correct options were not marked.

[12] https://www.limesurvey.org/

[13] https://mybinder.org/v2/gh/fusion-jena/MLProvLab/HEAD?urlpath=lab%2Ftree%2Fbinder%2Fevaluation_notebook.ipynb

[14] https://github.com/fusion-jena/MLProvLab

The majority of the questions were answered correctly, which matches the impression given

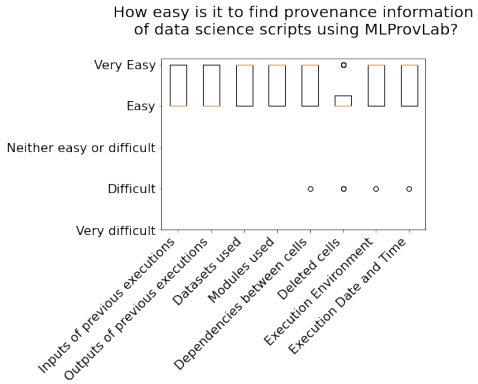

Fig. 6: Ease of finding provenance information of data science scripts using MLProvLab

by the users as shown in Figure 6. Finding the inputs and outputs of previous executions, the datasets and modules used, dependencies between cells, execution environment, and temporal aspects of notebook execution are either very easy or easy to find using MLProvLab (Figure 6). We did not provide any questions/tasks related to deleted cells; hence, we see that some participants were not aware of this feature of MLProvLab and chose the difficult option. Figure 7 shows the perceived usefulness and importance of MLProvLab modules.

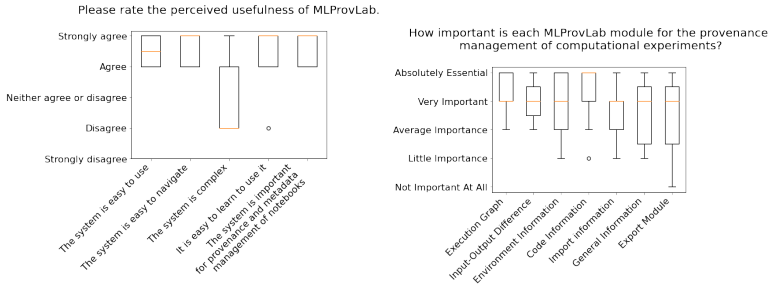

Fig. 7: Perceived usefulness and importance of MLProvLab and its modules for provenance management

Most of them marked that it was easy to use and navigate. Everyone agreed that the system is important for the provenance and metadata management of notebooks. Execution graph, Input-Output Difference, and Code Information are considered very important for users. The export information module was not considered important in the survey. We believe that this is because there were no tasks involving the export module. We received general comments from 11 participants for the open-response questions. The majority of them provided positive feedback. Some of the improvements provided by the participants include renaming the tabs to more meaningful names, rearranging tabs, creating a user option to directly input the epoch and execution number near the slider, and adding more general details in the Help tab as some features are not self-explanatory, and graph visualization not

following temporal order. Some suggested improvements are taken care of and implemented after the evaluation.

Limitations.

This study was exploratory, and the sample needs to be more diverse to generalize the findings. Most of the participants have a computer science background. We expected more participation from other areas of study. Five participants from fields other than computer science did not complete the study. However, only 1 of these five respondents uses Jupyter Notebooks and Machine Learning in their work. This could be one of the reasons for not participating in the user evaluation. Our primary users are data scientists who use and have used Jupyter Notebooks. Most of the participants are students, but also other academic grades are represented. We also see many participants who do not use ML in their work. As a result, we also got opinions from users with and without experience in the domain. However, we have observed that each such participant has answered nine and more questions using MLProvLab.

5 Conclusions and Future Work

We presented MLProvLab for the provenance management of data science notebooks. It is an extension of JupyterLab, to track, manage, compare, and visualize the provenance of notebooks. Through MLProvLab, users can efficiently and automatically track the provenance metadata, including datasets and modules used. We provide users the facility to compare different runs of computational experiments, thereby ensuring a way to help them make their decisions. The tool helps data scientists to collect more information on their experimentation and interact with them. It is designed so that the users do not need to change their scripts or configure them with additional annotations. In our future work, we aim to extend MLProvLab to identify the relationships between data and models for ML automatically. We want to track further the datasets and columns that have been used to derive the features of an ML model. This will help data scientists to get more information on the configurations used, e.g., hyperparameters, ML methods, etc. We also plan to provide interoperability by providing semantic annotations and descriptions of the collected fine-grained provenance information. We plan to use this provenance information to replay and rerun a notebook.

Acknowledgments

The authors thank the Carl Zeiss Foundation for the financial support of the project “A Virtual Werkstatt for Digitization in the Sciences (K3)” within the scope of the program line “Breakthroughs: Exploring Intelligent Systems for Digitization - explore the basics, use applications” and Friedrich Schiller University Jena for the IMPULSE funding: IP 2020-10.

References

[Ca17] Carvalho, L. A. M. C.; Wang, R.; Gil, Y.; Garijo, D.: NiW: Converting Notebooks into Workflows to Capture Dataflow and Provenance. In. SciKnow 2017, Austin, Texas, 2017, 2017.

[Da12] Davison, A.: Automated Capture of Experiment Context for Easier Reproducibility in Computational Research. Computing in Science Engineering 14/4, pp. 48–56, 2012, ISSN: 1521-9615.

[He19] Head, A.; Hohman, F.; Barik, T.; Drucker, S. M.; DeLine, R.: Managing Messes in Computational Notebooks. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019. ACM, p. 270, 2019.

[Ho14] Hoekstra, R.: PROV-O-Matic, https://github.com/Data2Semantics/prov-o-matic, Accessed 10 September 2021, 2014.

[Ke19] Kery, M. B.; John, B. E.; O'Flaherty, P.; Horvath, A.; Myers, B. A.: Towards Effective Foraging by Data Scientists to Find Past Analysis Choices. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems. ACM, Glasgow Scotland Uk, pp. 1–13, May 2019, ISBN: 978-1-4503-5970-2, visited on: 05/16/2021.

[KM18] Kery, M. B.; Myers, B. A.: Interactions for Untangling Messy History in a Computational Notebook. In: 2018 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2018, Lisbon, Portugal, October 1-4, 2018. IEEE Computer Society, pp. 147–155, 2018.

[KP17] Koop, D.; Patel, J.: Dataflow Notebooks: Encoding and Tracking Dependencies of Cells. In: 9th USENIX Workshop on the Theory and Practice of Provenance, TaPP 2017, Seattle, WA, USA, June 23, 2017. USENIX Association, 2017.

[KR+16] Kluyver, T.; Ragan-Kelley, B., et al.: Jupyter Notebooks-a publishing format for reproducible computational workflows. In: ELPUB. Pp. 87–90, 2016.

[KSK21] Kerzel, D.; Samuel, S.; König-Ries, B.: Towards Tracking Provenance from Machine Learning Scripts. In: Proceedings of the 13th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, IC3K 2021, October 25-27, 2021. SCITEPRESS, 2021.

[Ma21] Macke, S.; Parameswaran, A. G.; Gong, H.; Lee, D. J. L.; Xin, D.; Head, A.: Fine-Grained Lineage for Safer Notebook Interactions. Proc. VLDB Endow. 14/6, pp. 1093–1101, 2021, URL: http://www.vldb.org/pvldb/vol14/p1093-macke.pdf.

[Mc15] McPhillips, T. et al.: YesWorkflow: a user-oriented, language-independent tool for recovering workflow information from scripts. arXiv preprint arXiv:1502.02403/, 2015.

[Or20] Ormenisan, A. A.; Ismail, M.; Haridi, S.; Dowling, J.: Implicit provenance for machine learning artifacts. Proceedings of MLSys 20/, 2020.

[PGS18] Petricek, T.; Geddes, J.; Sutton, C.: Wrattler: Reproducible, live and polyglot notebooks. In (Herschel, M., ed.): 10th USENIX Workshop on the Theory and Practice of Provenance, TaPP 2018, London, UK, July 11-12, 2018. USENIX Association, 2018.

[Pi15] Pimentel, J. F. N.; Braganholo, V.; Murta, L.; Freire, J.: Collecting and Analyzing Provenance on Interactive Notebooks: When IPython Meets No Workflow. In: Proceedings of the 7th USENIX Conference on Theory and Practice of Provenance. TaPP'15, USENIX Association, Edinburgh, Scotland, p. 10, 2015.

[Pi20] Pineau, J.; Vincent-Lamarre, P.; Sinha, K.; Larivière, V.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; Larochelle, H.: Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program). arXiv preprint arXiv:2003.12206/, 2020.

[RTH18] Rule, A.; Tabard, A.; Hollan, J. D.: Exploration and Explanation in Computational Notebooks. In: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems. CHI '18, ACM, Montreal QC, Canada, 32:1–32:12, 2018, ISBN: 978-1-4503-5620-6.

[Sa20] Samuel, S.; Shadaydeh, M.; Böcker, S.; Brügmann, B.; Bucher, S. F.; Deckert, V.; Denzler, J.; Dittrich, P.; von Eggeling, F.; Güllmar, D., et al.: A virtual "Werkstatt" for digitization in the sciences. Research Ideas and Outcomes 6/, 2020.

[Sc17] Schelter, S.; Boese, J.-H.; Kirschnick, J.; Klein, T.; Seufert, S.: Automatically tracking metadata and provenance of machine learning experiments. In: Machine Learning Systems Workshop at NIPS. Pp. 27–29, 2017.

[Sc18] Schelter, S.; Biessmann, F.; Januschowski, T.; Salinas, D.; Seufert, S.; Szarvas, G.: On Challenges in Machine Learning Model Management. IEEE Data Eng. Bull. 41/, pp. 5–15, 2018.

[Sh23] Shankar, S.; Macke, S.; Chasins, S.; Head, A.; Parameswaran, A.: Bolt-on, Compact, and Rapid Program Slicing for Notebooks [Technical Report]./, 2023.

[SK18] Samuel, S.; König-Ries, B.: ProvBook: Provenance-based Semantic Enrichment of Interactive Notebooks for Reproducibility. In: International Semantic Web Conference (P&D/Industry/BlueSky). 2018, URL: http://ceur-ws.org/Vol-2180/paper-57.pdf.

[SK21] Samuel, S.; König-Ries, B.: Understanding experiments and research practices for reproducibility: an exploratory study. PeerJ 9/, e11140, Apr. 2021, ISSN: 2167-8359, URL: https://doi.org/10.7717/peerj.11140.

[SLK21] Samuel, S.; Löffler, F.; König-Ries, B.: Machine Learning Pipelines: Provenance, Reproducibility and FAIR Data Principles. In: Provenance and Annotation of Data and Processes - 8th and 9th International Provenance and Annotation Workshop, IPAW 2020 + IPAW 2021, Virtual Event, July 19-22, 2021, Proceedings. Vol. 12839. Lecture Notes in Computer Science, Springer, pp. 226–230, 2021.

[Va16] Vartak, M.; Subramanyam, H.; Lee, W.-E.; Viswanathan, S.; Husnoo, S.; Madden, S.; Zaharia, M.: ModelDB: a system for machine learning model management. In: Proceedings of the Workshop on Human-In-the-Loop Data Analytics. Pp. 1–3, 2016.

[Wa20] Wang, J.; Kuo, T.; Li, L.; Zeller, A.: Assessing and Restoring Reproducibility of Jupyter Notebooks. In: 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020. IEEE, pp. 138–149, 2020.

[We19] Wenskovitch, J.; Zhao, J.; Carter, S.; Cooper, M.; North, C.: Albireo: An Interactive Tool for Visually Summarizing Computational Notebook Structure. In: 2019 IEEE Visualization in Data Science (VDS). IEEE, pp. 1–10, 2019.

[Za18] Zaharia, M. et al.: Accelerating the Machine Learning Lifecycle with MLflow. IEEE Data Eng. Bull. 41/4, pp. 39–45, 2018, URL: http://sites.computer.org/debull/A18dec/p39.pdf.