A Qualitative Study of Cleaning in Jupyter Notebooks

Helen Dong
Carnegie Mellon University
USA

ABSTRACT

Data scientists commonly use computational notebooks because they provide a good environment for testing multiple models. However, once the scientist finds the ideal model, he or she will have to dedicate time to clean the code in order for others to understand it. In this paper, we perform a qualitative study on how scientists clean their code. Our end goal is to provide additional aid to data scientists, who can then focus more on their actual work rather than the routine and tedious cleaning duties.

CCS CONCEPTS

• **Software and its engineering** → *Software post-development issues.*

KEYWORDS

Empirical Study, Mining Repositories, Software Engineering for AI

ACM Reference Format:
Helen Dong. 2021. A Qualitative Study of Cleaning in Jupyter Notebooks. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3468264.3473490

1 INTRODUCTION

Data scientists often use computational notebooks, such as Jupyter or R notebooks, to conduct the exploratory programming aspect in a project [6]. From previous research [3], it was discovered that data scientists who use notebooks tend to incrementally add new cells of code to explore alternative methods. If the overall end goal is to share their model with other collaborators, these scientists must perform cleaning steps in order to make it easier for others to understand their code. Although a seemingly simple task, this cleaning step can be tedious and laborious. Additionally, current notebook environments provide little explicit support for cleaning, and the few academic projects in this area focus on specific tasks, such as slicing the code used to reproduce a specific figure or result [1]. More generally, we do not have a solid understanding of the different types of cleaning steps that data scientists commonly engage in, without which we cannot effectively design a more tailored tool.

This issue leads us to ask the following research question: what types of cleaning activities are the most common in computational notebooks? To answer this question, we perform a qualitative analysis of public notebooks, their version history, and related source code files to identify which cleaning activities data scientists engage in to help us understand what type of tools could support typical cleaning tasks. Our long-term vision is for tool builders to address possible gaps and provide additional aid to data scientists, who then can focus more on their actual work rather than the routine cleaning work. Having clean code is important because it not only saves time for the reader, but also for the writer, since the writer ideally will not have to take the time to explain the code to the reader.

2 RESEARCH APPROACH

Given that we have little initial knowledge about different forms of cleaning, we adopt a mostly qualitative research strategy in which we carefully analyze the public history of a sample of Jupyter notebooks. For each notebook, we noted all the changes made between two artifacts (i.e the notebook before and after a specific commit), and used inductive coding to identify common groups and come up with a coding frame. Then we use our coding frame to sample notebooks and identify trends.

To come up with the coding frame, we fetched 2600 Jupyter notebooks on GitHub and randomly selected about 25 of them to analyze their commit history. We identified cleaning activities by manually analyzing changes between commits using reviewNB [4]. While looking for patterns in cleaning activities, we found that notebooks are used for different purposes, and the cleaning activities vary correspondingly. Using literature [1] from previous studies and observing characteristics of the 25 notebooks, we were able to distinguish 3 main groups that most notebooks on GitHub fall under. Due to this separation of groups, we determined that a uniformly random sample was not suitable for our purposes. Thus, we decided to stratify our sample in order to cover a larger range of different notebooks.

3 CORPUS SELECTION & SAMPLING

We describe the three different types of notebooks and how we sample each of them below.

Sharing Notebooks are notebooks that are created with an intended purpose to be shared with others. Such notebooks are characterized by extensive README files that communicate the purpose of the project to an audience. These notebooks tend to have less inline code comments and more markdown (titles in Jupyter Notebooks) describing the notebook on a high level.

Production notebooks are those that are used to prepare models that are intended to be integrated in some software system to be used as a library by other developers or as product by end users. For these notebooks, there exists some sort of production code, and in our case, we looked for python files (which were originally .ipynb files converted to .py files). These notebooks also tend to have an

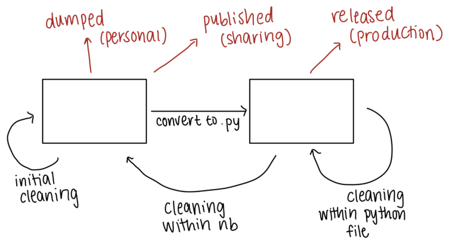

Figure 1: Process of cleaning notebooks. A box represent a state of a notebook or python file. An arrow indicates a transition to a new state..

extensive README, but they generally explain how to use the tool rather than describing the code on a high level.

Personal Notebooks are notebooks that are used for data exploration, and may not become anything more than that. In these notebooks we tend to observe a lot of playground code and testing, and it is harder to understand the purpose of the code. These notebooks may be for the developer's personal use and not intended for anyone else to see.

After identifying these three groups, we were able to perform stratified sampling to continue our study. We fetched repositories from GitHub that were last edited in 2018. The reason for this was to ensure that the project was completed or abandoned, and thus the notebook will likely not have any further cleaning steps. We chose a relatively recent date because we wanted to look at notebooks that represented cleaning in modern-day coding. To retrieve **sharing notebooks**, we searched for notebooks (1) in repositories that had at least 10 stars on GitHub (i.e., received some recognition), (2) that had an extensive README file, and (3) do not contain python files to exclude libraries and production notebooks. To fetch **production notebooks**, we used clone detection techniques to identify similar code snippets between notebook and python files. Specifically, we looked for type 3 clones, which are defined as code where 'one or more statements can be modified, added, or removed.' [2] To find the code clones, we used a tool, namely Nicad [5]. With multiple rounds of testing the best modes on Nicad, we decided to detect the type 3 clones with at least 75% similarity. To find **personal notebooks**, we searched for notebooks that did not fall under the sharing or production notebook category. As long as it was not used for a class project, it was a viable personal notebook. Within each group of notebooks, we randomly selected 10 notebooks and carefully looked through the commit history to identify common cleaning qualities.

4 RESULTS

Figure 1 displays, on a high level, the cleaning steps and different categories of notebooks that implement these cleaning steps.

Based on our observations, the most common activity involved adding or deleting comments, or making changes by commenting out code. This is because in notebooks, developers may write many functions for testing purposes and when they do not work, they can just comment them out and run other cells. Upon finding the best one, they can just delete the other commented out, less optimal functions. Another common act of cleaning we saw was the act of reordering cells. Since notebooks can be executed in any order desired, it is possible the developer wrote the code sequentially, but did not run them in that order. After finalizing the code, the developer has reorder the cells in order for the reader to understand the intended order of the cells. While all notebooks embodied similar qualities, different types of notebooks also had specific traits.

Personal Notebooks. We noted minimal cleaning in these types of notebooks; the extent of cleaning was merely making changes to comments or markdown. Since these notebooks were mostly meant for exploration, we noticed a lot of code being commented out throughout each commit. We also saw a lot of inline changes to the code (for example, adjusting parameters of models). There were instances where the notebook on the final version was not completely clean (i.e. commented out code remained was an obvious sign). This could be because the author did not intend to clean the notebook as it may not have been meant to seen by others.

Sharing Notebooks. The cleaning traits in sharing notebooks mostly had to do with markdown changes (i.e adding headers or descriptive paragraphs for a group of cells). The markdown changes in these notebooks seemed to be directed towards an audience, and often would provide a high level explanation of the code. One difference we noticed between sharing notebooks and personal notebooks was that in sharing notebooks, the cleaning steps occured much earlier in the commit history as compared to the personal notebooks. This is likely because authors of sharing notebooks originally intended to share the notebook with others, so keeping their code clean was on their priority list.

Production Notebooks. For production notebooks, the most cleaning occured in the transition between notebook to python file. When we compared the notebook to the python files, we noticed that there was often one notebook that was split into multiple python files. Additionally, the subset of the code was often put into a class and any commented out code in the notebook was removed. The notebook was likely used for testing and perfecting code and after, it was split into to several python files. When we compared the notebooks and the python files, the notebooks were always messier and less readable. The notebook also included commented out code that was never cleaned up within the notebook but was later removed when converted to python files. We also saw instances where the notebook was directly converted to a python file, which occured quite often, but we did not look into these cases because it was likely that the developer just used a tool such as nbconvert to convert the notebook to a python file.

5 CONCLUSION

From this qualitative study, we saw that cleaning practices vary depending on the purpose of the notebook. An ideal tool would allow the user to identify the purpose of the notebook, and clean the notebook accordingly. For instance, the cleaning for personal notebooks can focus more at the code level whereas the cleaning for sharing notebooks can focus more on markdown to allow a reader to understand the code. For production notebooks, there could be a functionality where it takes in a notebook file and splits into separate python files with respective classes.

Acknowledgements. The work was supported as part of an NSF-funded REU project (award 1813598).

REFERENCES

[1] Andrew Head, Fred Hohman, Titus Barik, Steven M Drucker, and Robert DeLine. 2019. Managing messes in computational notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–12.

[2] Yue Jia, David Binkley, Mark Harman, Jens Krinke, and Makoto Matsushita. 2008. KClone: A Proposed Approach to Fast Precise Code Clone Detection. In *Proceedings of the 16th IEEE International Conference on Program Com- prehension, ICPC 2008*. 172–181.

[3] Liu, Jiali, Nadia Boukhelifa, and James R. Eagan. 2019. Understanding the role of alternatives in data analysis practices.. In *IEEE Transactions on Visualization and Computer Graphics*.

[4] Amit Rathi. 2018. ReviewNB Tool. https://www.reviewnb.com/.

[5] C.K. Roy and J.R. Cordy. 2008. NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty- Printing and Code Normalization. In *Proceedings of the 16th IEEE International Conference on Program Com- prehension, ICPC 2008*. 172–181.

[6] Beau Sheil. 1983. Variolite: Supporting Exploratory Programming by Data Scientists. In *Environments for exploratory programming. Datamation 29, 7*. 131–144.