Splitting, Renaming, Removing: A Study of Common Cleaning Activities in Jupyter Notebooks

Helen Dong
Carnegie Mellon University
hldong@andrew.cmu.edu

Shurui Zhou
University of Toronto
shuruiz@ece.utoronto.ca

Jin L.C. Guo
McGill University
jguo@cs.mcgill.ca

Christian Kästner
Carnegie Mellon University
kaestner@cs.cmu.edu

Abstract

Data scientists commonly use computational notebooks because they provide a good environment for testing multiple models. However, once the scientist completes the code and finds the ideal model, he or she will have to dedicate time to clean up the code in order for others to easily understand it. In this paper, we perform a qualitative study on how scientists clean their code in hopes of being able to suggest a tool to automate this process. Our end goal 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 and tedious cleaning work. By sampling notebooks from GitHub and analyzing changes between subsequent commits, we identified common cleaning activities, such as changes to markdown (e.g., adding headers sections or descriptions) or comments (both deleting dead code and adding descriptions) as well as reordering cells. We also find that common cleaning activities differ depending on the intended purpose of the notebook. Our results provide a valuable foundation for tool builders and notebook users, as many identified cleaning activities could benefit from codification of best practices and dedicated tool support, possibly tailored depending on intended use.

I. Introduction

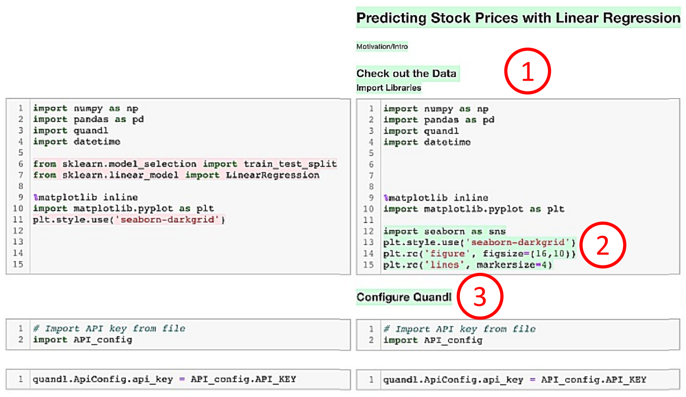
The data scientist role has become increasingly important for developing data-driven software systems. One type of tool frequently adopted by data scientists is computational notebooks, such as Jupyter or R notebooks, for conducting the exploratory programming during the initial phase of model development [1]. Notebooks allow data scientists to run small chunks of code (organized in code cells) quickly and in any order desired. Therefore, it is easy explore and compare multiple options to select a high performing configuration during machine learning model development. As observed from previous work [2], such process tends to incremental, i.e. data scientists may add new cells or small code blocks gradually as they explore alternative methods. When the data scientists need to share their models with other collaborators, they must make explicit effort to clean the notebook to make it easier to understand and reuse.

The meaning of a 'clean' notebook can be inherited from what is generally considered to be 'clean' code, such as having properly named functions, containing descriptive markdown and meaningful comments to explain the code's intention, and having no dead or irrelevant code. In Figure 1, we show an example of an author (developer on Github with public repository) who cleaned code with changes to markdown, including adding titles and refactoring code, both of which made the code visually more appealing, and in essence, aided in readability.

Modern programming IDEs provide various functions and plug-ins that supporting cleaning the source code [3]–[6]. However, current notebook environments provide little explicit support for cleaning, making the cleaning process laborious and tedious. Support is limited to a few academic projects that focus on specific tasks, such as slicing the code used to reproduce a specific figure or result [7], generating documentation [8], and code synthesis [9]. More importantly, we currently do not have a solid understanding of the relevant types of cleaning tasks data scientists most commonly engage in, which would be needed to focus on tooling effort. In this paper, we perform a qualitative analysis of a set of curated public Jupyter notebooks, their version history, and related source code files to identify which cleaning activities data scientists engage in, which will be useful in understanding how to design effective tools to support typical cleaning tasks. Specifically, we are interested in *what are the common cleaning activities, what are the context of those activities, and to what extent could they be automated.*

Based on our observations, the most common activity involved adding or deleting comments, such as adding comments to describe the functions or commenting out sub-optimal code. This is due to the nature of notebooks, in which developers can write several functions or models, one being in each cell of the notebook, and commenting out entire cells that run less optimally. After finding the best one, they can just delete the other commented out, less optimal functions. Another common act of cleaning we saw was reordering cells. Because notebooks can be executed in any order desired, it is possible for the developer to write code sequentially, but then run them or intend to run them in a different order. However, after finalizing the code, the developer will ideally want to reorder the cells so the reader does not have to figure out the intended execution order.

In order to understand the context of the cleaning, we collected the notebooks that are likely to serve different purposes, i.e. personal notebooks (for mostly exploration purposes),



Predicting Stock Prices with Linear Regression

Motivation/Intro

Check out the Data

Import Libraries

```python
import numpy as np
import pandas as pd
import quandl
import datetime

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-darkgrid')
```

```python
import numpy as np
import pandas as pd
import quandl
import datetime




%matplotlib inline
import matplotlib.pyplot as plt

import seaborn as sns
plt.style.use('seaborn-darkgrid')
plt.rc('figure', figsize=(16,10))
plt.rc('lines', markersize=4)
```

Configure Quandl

```python
# Import API key from file
import API_config
```

```python
# Import API key from file
import API_config
```

```python
quandl.ApiConfig.api_key = API_config.API_KEY
```

```python
quandl.ApiConfig.api_key = API_config.API_KEY
```

Fig. 1: Example of a single iteration of cleaning. ① represents changes to markdown (adding descriptions); ② represents an example of code refactoring; and ③ represents the addition of titles to increase readability and to help the reader understand the code.

sharing notebooks (notebooks that are meant to be used or seen by others), and production notebooks (notebooks that will be part of a larger project, such as creating a library). While all notebooks embodied some of the cleaning activities, such as the ones described above, we also discovered different qualities for different types of notebooks. An effective cleaning tool would have to consider what type of notebook the data scientist is working with and clean accordingly. For instance, the cleaning for personal notebooks can focus more at the code level, with comments, and reordering cells whereas the cleaning for sharing notebooks can focus more on the comments, markdown, and overall understanding of how to use the code. For production notebooks, there could be a functionality where it takes in a notebook file and cleans it up by removing commented out code and splits into separate Python files with respective classes.

II. Study Design

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 Jupyter notebooks collected from GitHub public repositories. Through this process, we identified and grouped different kinds of cleaning activities.

A. *Qualitative analysis of notebook version history*

We identified cleaning activities in notebooks by manually analyzing each notebook's history in version control closely. To this end, we cloned the notebook's repository and examined the diffs between all versions. Because notebooks are stored in a format that does not lend itself to analysis with traditional text-based *diff* tools, we used reviewNB [10], a notebook versioning and collaboration tool, which allows us to differentiate the changes on both code and markdown cells.

We proceeded in two steps. In the first step, we analyzed a small sample of notebooks to develop a coding frame of different cleaning activities and in the second step, we used this coding frame to analyze a larger and more representative sample of notebooks.

To establish the coding frame, two authors manually analyzed the changes between each commit from 28 different notebooks (details about our data collection procedure is in Section II-B) through an open coding process [11], including reading the commit messages, analyzing the code and markdown changes, and taking notes about observed cleaning activities. After collecting initial coding as notes, we collectively analyzed the codes and grouped them, resulting a coding frame of different cleaning activities. We analyzed the clarity of the coding frame on a set of 10 different notebooks from our dataset to verify our coding frame. After establishing the coding frame, we then analyzed the history of a larger sample of notebooks and their histories, applying the coding frame to identify instances of cleaning activities through axial coding [11]; After creating our coding frame, we did not observe any additional cleaning activities in the newly sampled notebooks so we did not update our coding frame.

In Table I, we show the coding frame of cleaning qualities that we created from our qualitative analysis.

B. *Data Collection*

For our analysis, we need to collect notebooks with publicly available histories. There are millions of public notebooks on platforms such as GitHub and Kaggle [12]–[15], many of which have at least some history recorded with a version control system. Due to the high manual effort in the qualitative analysis, we curate relatively small samples of notebooks from this large pool.

We fetched GitHub repositories that contained at least one notebook and had at least 10 commits with changes to the notebook(s). To observe the recent practice on notebook cleaning activities and to make sure the notebooks under study are complete (without receiving future cleaning steps), we only considered repositories in which the last edit was conducted between 18 and 30 months before the start of our research. This step yields an initial dataset with more than 20k notebooks.

To get a better understanding of what the developers considered as cleaning, we start with randomly sampling 20 notebooks from our initial dataset of 20k notebooks that had the terms 'cleaning' or 'refactoring' anywhere in their commit messages. After establishing the understanding of the scope of cleaning activities, we then sampled another 28 notebooks with the same filtering strategy from the dataset of 20k notebooks for producing our coding frame as described in the previous section.

While looking for patterns in cleaning activities, we found that notebooks are used for different purposes, and the cleaning activities varied correspondingly. Based on the previous studies [1] and our initial observation of 28 notebooks about their characteristics, we classified notebooks into one of three groups – sharing notebooks, production notebooks, and personal notebooks, which we describe in the next section. Due to this distinction, we determined that a uniformly random sample was not suitable for understanding the cleaning activities for different types of notebooks. Instead, we decided to perform a stratified sampling strategy to collect additional notebooks for analysis using our coding frame.

Thus in our analysis of cleaning activities for different categories of notebooks, we sampled another 30 notebooks from our 20k notebook dataset – 10 from each of the three groups we identified.

C. Categorization of Notebooks

We describe the three different types of notebooks below. We note that these three groups cover a wide variety of public notebooks found on GitHub but we excluded some groups, such as notebooks meant for class projects, which include either a course-number (i.e. CS-100) or keywords such as "project" or "class" in the title.

- **Sharing Notebooks** are notebooks that are created with an intended purpose to be shared with others, including the general public. Examples of sharing notebooks include ones to be submitted to competitions or to be used as a part of software documentation, textbooks, or other the training materials. Such notebooks are characterized by extensive README files that communicate the purpose of the project to others. These notebooks also tend to have less comments but more markdown describing the notebook on a high level.
- **Production notebooks** are those that are used to prepare models that are meant to be integrated in some software system either as library used by other developers or as a product by end users. While in some cases, notebooks may directly be executed in production pipelines [16]–[18], more commonly notebooks are used for prototyping and development before the code is migrated into a code infrastructure for production use [19]. For example, a machine learning model may be served to end users through a web interface or REST API, or may be integrated into a product. We are interested in cleaning steps in such projects, including those extra steps that happen when migrating to production code, as shown in Figure 2.
- **Personal Notebooks** are notebooks that are used for data exploration and modeling but without taking apparent steps to share the notebook or to use it for production settings. Even though they were published publicly, such notebooks are often used as playgrounds and for exploring ideas or learning concepts; it is harder to understand the purpose of the code as an outsider. These notebooks tend to have little documentation in markdown cells, though code comments may still be common.

D. Sampling Each Type of Notebook

Similar with how we created the initial notebook dataset, we first selected all notebook repositories from GitHub with at least 10 commits to a notebook file and no changes in 18 month prior to our research. We then randomly sample notebooks from this set and categorize them to one of the three categories based on the criteria described below. We

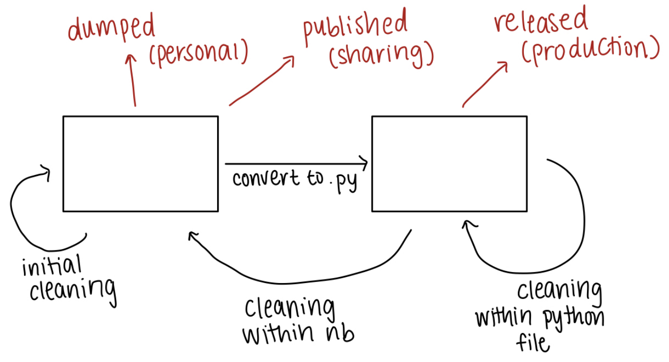


Fig. 2: Process of cleaning notebooks. A box represent a state of a notebook or Python file. An arrow indicates a transition to a new state.

repeat the sampling process until we retrieve ten notebooks for each category.

We categorize a notebook as a **sharing notebook** if the notebook is in a repository (1) that had at least 10 stars on GitHub (i.e., it has received some public recognition), (2) that had an extensive README file, and (3) that did not contain Python files (so that we can exclude libraries and production notebooks).

For **production notebooks**, we used clone detection techniques to identify similar code snippets between notebook and Python files that are included in the same repository. Specifically, we looked for Type-3 clones, which are defined as code where 'one or more statements can be modified, added, or removed.' [20] To find the code clones, we used a tool, namely Nicad [21]. With multiple rounds of testing the best modes on Nicad, we decided to detect the Type-3 clones with at least 75% similarity between the code in a notebook file, which we extracted using nbcovert [22], and the code in a Python file.

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, which we manually filtered out, it was a viable personal notebook.

Using ten notebooks from each group, we carefully looked through the commit history to identify common cleaning qualities following our coding frame. The result will be discussed in Section III.

E. Limitations and Threats to Validity

Our design restricts us to the observations of changes between consecutive commits submitted to GitHub. We cannot study the small and individual changes to notebook cell as they are performed by a data scientist because they cannot be distinguished in commit history. While it would be possible to collect more fine-grained history information in a lab setting [23], we decided to analyze public histories, which allowed us to sample from a much larger pool of data science projects, to observe real-world projects developed over

Cleaning Activity	Description
Deleting commented block	Deletion of comments or commented out code (dead/unused code or comments).
Adding commented block	Adding comments to describe code functionality or commenting out code that may not work.
Reordering cells	Change the order of cells without modifying their content.
Adding markdown cells	Adding descriptions in headings and titles to sections of the notebook as markdown cells.
Renaming variables	Change variables names that are more meaningful and informative.
Splitting the notebook	Splitting up cells of the notebook into different files.
Reorganizing into functions and classes	Wrapping specific functions within a class for easy understanding and putting blocks of code within functions.
Removing cells	Removing entire cells from the code.
Fixing typos in code	The commit fixes minor issues within the code.
Adding temporary values	Adding temporarily values that are to be changed later. For instance, making the "else" case branch do nothing for the commit.

TABLE I: Finalized Coding Frame: Summary of the common cleaning traits we noticed listed in order of most common to least common.

extended periods, and to not influence the data scientists with our observations as they are working on their projects.

A drawback of our study design is that it provides limited insights into *why* certain cleaning steps were performed. Since our observation is during the commit time, the context of the cleaning activities, i.e. the newly developed code before it is committed, has been lost.

While we invested significant effort to stratify our sample, our sample size is relatively small and does not allow for statistical generalization. Additionally, we may have missed other kinds of notebooks not explicitly represented in our strata which may exhibit different cleaning characteristics.

III. Results and Observations

In our stratified sample of 30 notebooks, we found seven different cleaning activities that were frequently used, as summarized in Table I.

The most common cleaning activity, which we saw in twenty-eight out of the thirty notebooks, is **deleting commented blocks**, which includes deleting comments as well as deleting commented-out code. This activity has to do with removing obsolete code and comments that describing the code, where the data scientists may have found a better version of a model so they can remove the sub-optimal one when cleaning. The high frequency of the occurrence of this category is an expected given the exploratory nature of data science work where data scientists often try multiple alternatives before settling on one [24] and the frequent use of copy-and-paste during exploration [1].

The second most common cleaning activity, which we observed in twenty-two out of the thirty notebooks, was **adding commented blocks**. For instance, after writing all the code, the data scientists may want to go back and add textual comments describing the code to aid in readability. This is also an expected behavior for programming in both the traditional IDE and in notebook environment when the developer elaborates on the rationales of their code for the readers, either themselves or others.

The third most common cleaning activity was **reordering cells**, which we saw in fifteen of the thirty notebooks. Since notebooks can be executed in whatever order desired, it is possible for the data scientists to write code sequentially, but to run them in a different order [25]. This leads to the difficulty of reproducing the result from a certain run of the notebook. After finalizing the code, the developer will ideally reorder the cells to explicitly communicate the intended execution order of the cells.

The fourth most common cleaning activity that we observed within twelve out of the thirty notebooks was **adding markdown**. We mostly observed changes to markdown in sharing notebooks (and sometimes personal notebooks). However, the markdown differs for each type of notebook. The markdown in sharing notebooks are visibly directed to an audience – for instance, the comments often describe how to use the code. Whereas in personal notebooks, the comments describe what each part of the code does.

Another cleaning activity we observed was **renaming variables** – in eight out of the thirty notebooks. Renaming variables has to do with naming variables with more meaningful names in later commits, for instance, changing the variable "*x*" to "*numberOfIterations*". This is also an expected cleaning activity because it helps in readability and maintainability of the code.

We observed the notebook being **split into multiple Python files** during cleaning for production notebooks. This occurred in seven out of ten production notebooks in our dataset. In the production notebook repositories, there would typically be a few notebooks in early commits, with a lot of uncleaned code. But in later commits, Python files would emerge that contained parts of the notebook code. It is likely that the developers used the notebook for writing experimental code, and then converted the notebooks to Python files for integration and deployment purposes.

Finally, the last common quality we observed was **reorganizing code into functions and classes**. We observed this in four out of the thirty notebooks, and solely in production notebooks. We show an example of a function being wrapped in a class below. In an earlier commit, the notebook looked like the following:

```
# In[1]:
#image to rotate
image = cv2.cvtColor(image,
    cv2.COLOR_BGR2RGB)
#angleTest = 180
angleTest = 90
#rotating at an angle
image = ndimage.rotate(image, angleTest)
print(image)
return image
```

After cleaning, the notebook code was moved over to a Python file with the dead code deleted and the code is put into functions to improve the readability and modularity.

```
def rotate_image(image):
    """
    :param takes in an image to rotate
    :return: image that has been rotated
    """
    image = cv2.cvtColor(image,
    cv2.COLOR_BGR2RGB)

    angleTest = 90
    image = ndimage.rotate(image, angleTest)

    return image
```

Moreover, there are cases where the entire code from the notebook was placed within a class.

Additionally, we saw instances where the notebook was directly converted to a Python file, but we omit these cases in our study because they don't indicate any explicit cleaning effort. Instead, it was likely that the developer used tools such as nbconvert to convert the notebook to a Python file.

Beyond general observations about cleaning steps across our sample, we also found noticeable differences among different kinds of notebooks. First of all, *personal notebooks* contained substantially fewer cleaning activities. We found that commenting out code to be much more common than full removal in these kind of notebooks. In many of these notebooks, commented out code still persisted until the final commit, suggesting that cleaning actions have not been conducted, possibly because the notebook was not intended to be consumed by others. Although moving cells around was a common cleaning trait amongst all groups of notebooks, we saw it more often in personal notebooks. In contrast, the cleaning steps in *sharing notebooks* mostly related to markdown changes, which persisted much more than in other kinds of notebooks. Also, another major difference we noticed was that in sharing notebooks, the cleaning steps occurred much earlier in the commit history as compared to the personal notebooks. This could be due to the fact that the author had intended audiences early on, so they wanted to ensure the readability of the notebook constantly. In *production notebooks*, splitting code into multiple files was particularly common during the code migration step, as was naturally wrapping notebook code into functions and classes. During migration it is also very common to remove code and obsolete comments, while adding extra documentation as code-level comments and reorganizing code into Python files neatly. Production notebooks also often had a descriptive README, except rather than describing the code on a high level, it describes how to download or use some potential tool (often the tool provided by the notebook code).

IV. Discussion

From our exploration, we saw that depending on the purpose of the notebook, cleaning steps are implemented differently. For instance, if the notebook is meant to be distributed, it is usually nicely formatted, easier to understand, and cleaning happens earlier in the commits. If the notebook is meant to be eventually turned into production code via Python, it is messier but is eventually cleaned up in the Python code. These notebooks are used as rough drafts for the Python files. If the notebook is for personal exploration use, there is less cleaning observed. In these notebooks, we also notice there are less commits. From these observations, we can come up with suggestions for cleanup tools. For instance, ideally, the tool will allow the notebook user to identify the purpose of the notebook, and support cleaning the notebook depending on the type. The cleaning for personal notebooks can focus more at the code level, with comments, and reordering cells whereas the cleaning for sharing notebooks can focus more on the comments, markdown, and overall understanding of the code. Then for production notebooks, the cleaning can focus on how to organize the notebook into a python file or multiple python files.

V. Related work

Researchers have recently explored how developers use notebooks [26]–[28] and what pain points they face [29], emphasising a very iterative and exploratory workflow that is often not well supported by current tools. Many of these studies mention that developers routinely clean code after exploratory steps, but they did not analyze further what activities notebook users engage with for cleaning, which was the goal of this paper.

Several recent studies have analyzed public notebooks on GitHub [14], [15], [28], [30], on Kaggle [13], and within corporations [15], finding that many are not reproducible [14], [31], contain redundant code [28], and use poor coding practices [32]. We are not aware of any longitudinal analyses of individual notebooks.

Tool support for computational notebooks is constantly evolving and recent versions support many more features developers might expect from integrated development environments, such as auto-completion and code navigation. In addition, recent academics have developed tooling for specific tasks, such as slicing the code used to reproduce a specific figure or result [7], generating documentation [8], and code synthesis [9]. We are hopeful that our results help inform future tool support that can help with specific cleaning activities.

VI. Conclusion

From this qualitative study, we concluded that cleaning occurs in most notebooks and is an iterative approach that occurs throughout the project. We used a set of heuristics and tools to collecting notebooks for different purposes and analyzed the occurrence of various cleaning activities. Our study suggested that in personal notebooks, cleaning occurs in later commits, and they mostly involve moving cells around or adding markdown. In notebooks that are meant for sharing, cleaning occurs much earlier, and a lot of it has to do with changing markdown by adding headers or editing the README file. In production notebooks, cleaning occurs mostly between the transition from notebook to a Python file or within the Python file itself. Our work indicates that the cleaning additives taken for the notebooks share certain similarity with coding in traditional IDEs but also have distinct need due to factors that are unique to notebooks, such as the flexibility of cell execution order and the explicit role of a markdown cell. Our work also illustrates the necessity of effective notebook tools that should support cleaning activities of notebooks with different purposes.

REFERENCES

[1] B. Sheil., "Variolite: Supporting exploratory programming by data scientists," in *Environments for exploratory programming. Datamation 29, 7*, 1983, pp. 131–144.

[2] J. Liu, N. Boukhelifa, and J. R. Eagan., "Understanding the role of alternatives in data analysis practices." in *IEEE Transactions on Visualization and Computer Graphics.*, 2019.

[3] M. Fowler, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[4] E. Murphy-Hill, C. Parnin, and A. P. Black, "How we refactor, and how we know it," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 5–18, 2011.

[5] S. Romano, C. Vendome, G. Scanniello, and D. Poshyvanyk, "A multi-study investigation into dead code," *IEEE Transactions on Software Engineering*, vol. 46, no. 1, pp. 71–99, 2018.

[6] D. Hou, "Studying the evolution of the eclipse java editor," in *Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange*, 2007, pp. 65–69.

[7] A. Head, F. Hohman, T. Barik, S. M. Drucker, and R. DeLine, "Managing messes in computational notebooks," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–12.

[8] C. Yang, S. Zhou, J. L. Guo, and C. Kästner, "Subtle bugs everywhere: Generating documentation for data wrangling code," in *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 11 2021.

[9] I. Drosos, T. Barik, P. J. Guo, R. DeLine, and S. Gulwani, "Wrex: A unified programming-by-example interaction for synthesizing readable code for data scientists," in *Proceedings of the 2020 CHI conference on human factors in computing systems*, 2020, pp. 1–12.

[10] A. Rathi, "Reviewnb tool," https://www.reviewnb.com/, 2018.

[11] J. W. Creswell and C. N. Poth, *Qualitative inquiry and research design: Choosing among five approaches*. Sage publications, 2016.

[12] A. Rule, A. Birmingham, C. Zuniga, I. Altintas, S.-C. Huang, R. Knight *et al.*, "Ten simple rules for writing and sharing computational analyses in jupyter notebooks," *PLoS Comput Biol*, vol. 15, no. 7, 2019.

[13] L. Quaranta, F. Calefato, and F. Lanubile, "KGTorrent: A dataset of python Jupyter notebooks from kaggle," in *Proc. of 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 2021, pp. 550–554.

[14] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire, "Understanding and improving the quality and reproducibility of Jupyter notebooks," *Empirical Software Engineering*, vol. 26, no. 4, pp. 1–55, 2021.

[15] F. Psallidas, Y. Zhu, B. Karlas, M. Interlandi, A. Floratou, K. Karanasos, W. Wu, C. Zhang, S. Krishnan, C. Curino *et al.*, "Data science through the looking glass and what we found there," *arXiv preprint arXiv:1912.09536*, 2019.

[16] A. Therriault, "Jupyter notebooks and production data science workflows," Talk at JupyterCon'17, https://www.youtube.com/watch?v=gM1XMu3BOps, Nov. 2017.

[17] Netflix Technology Blog, "Scheduling notebooks at Netflix," Blog post, https://netflixtechblog.com/scheduling-notebooks-348e6c14cfd6, Aug. 2018.

[18] T. Kopp, "How data scientists can tame Jupyter notebooks for use in production systems," Blog post, https://tanzu.vmware.com/content/blog/how-data-scientists-can-tame-jupyter-notebooks-for-use-in-production-systems, Jul. 2018.

[19] A. C. Rule, "Design and use of computational notebooks," Ph.D. dissertation, University of California, San Diego, 2018.

[20] Y. Jia, D. Binkley, M. Harman, J. Krinke, and M. Matsushita, "Kclone: A proposed approach to fast precise code clone detection," in *Proceedings of the 16th IEEE International Conference on Program Com- prehension, ICPC 2008*, 2008, pp. 172–181.

[21] C. Roy and J. Cordy, "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 Comprehension, ICPC 2008*, 2008, pp. 172–181.

[22] J. D. Team, "Nbconvert," https://https://nbconvert.readthedocs.io/en/latest//, 2015.

[23] M. B. Kery, B. E. John, P. O'Flaherty, A. Horvath, and B. A. Myers, "Towards effective foraging by data scientists to find past analysis choices," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–13. [Online]. Available: https://doi.org/10.1145/3290605.3300322

[24] H. Fangohr, M. Beg, M. Bergemann, V. Bondar, S. Brockhauser, C. Carinan, R. Costa, F. Dall'antonia, C. Danilevski, W. Ehsan, S. Esenov, R. Fabbri, S. Fangohr, G. Flucke, C. Fortmann, D. Fulla Marsa, G. Giovanetti, D. Göries, S. Hauf, and J. Kieffer, "Data exploration and analysis with Jupyter notebooks," in *17th Int. Conf. on Acc. and Large Exp. Physics Control Systems*, 01 2019, pp. 799–806.

[25] J. Wang, L. Li, and A. Zeller, "Restoring execution environments of Jupyter notebooks," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 1622–1633.

[26] A. Rule, A. Tabard, and J. D. Hollan, "Exploration and explanation in computational notebooks," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–12.

[27] M. B. Kery, M. Radensky, M. Arya, B. E. John, and B. A. Myers, "The story in the notebook: Exploratory data science using a literate programming tool," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–11.

[28] A. P. Koenzen, N. A. Ernst, and M.-A. D. Storey, "Code duplication and reuse in Jupyter notebooks," in *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 2020, pp. 1–9.

[29] S. Chattopadhyay, I. Prasad, A. Z. Henley, A. Sarma, and T. Barik, "What's wrong with computational notebooks? pain points, needs, and design opportunities," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–12.

[30] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire, "A large-scale study about quality and reproducibility of Jupyter notebooks," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019, pp. 507–517.

[31] J. Wang, K. Tzu-Yang, L. Li, and A. Zeller, "Assessing and restoring reproducibility of Jupyter notebooks," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2020, pp. 138–149.

[32] J. Wang, L. Li, and A. Zeller, "Better code, better sharing: on the need of analyzing Jupyter notebooks," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, 2020, pp. 53–56.