Better Code, Better Sharing: On the Need of Analyzing Jupyter Notebooks

Jiawei Wang  
Faculty of Information Technology, Monash University, Australia

Li Li  
Faculty of Information Technology, Monash University, Australia

Andreas Zeller  
CISPA Helmholtz Center for Information Security, Germany

ABSTRACT

By bringing together code, text, and examples, Jupyter notebooks have become one of the most popular means to produce scientific results in a productive and reproducible way. As many of the notebook authors are experts in their scientific fields, but laymen with respect to software engineering, one may ask questions on the quality of notebooks and their code. In a preliminary study, we experimentally demonstrate that Jupyter notebooks are inundated with poor quality code, e.g., not respecting recommended coding practices, or containing unused variables and deprecated functions. Considering the education nature of Jupyter notebooks, these poor coding practices, as well as the lacks of quality control, might be propagated into the next generation of developers. Hence, we argue that there is a strong need to programmatically analyze Jupyter notebooks, calling on our community to pay more attention to the reliability of Jupyter notebooks.

CCS CONCEPTS

• Software and its engineering → Software verification and validation.

KEYWORDS

Python, Jupyter Notebooks, Quality Assurance, API Deprecation

ACM Reference Format:

1 INTRODUCTION

Jupyter, a free open-source web application allowing users to write documents containing explanatory text, equations, and visualizations, as well as live codes and their execution results, has become tremendously popular nowadays. It provides a way for faculties to make a living workbook for sharing computational information (e.g., code) along with explanations, meanwhile, students can benefit from the live code to better understand the concepts introduced in the notebook. In addition to tutoring purposes, Jupyter notebook has also become the data scientists’ computational notebook of choice. Indeed, Jupyter has emerged as a de-facto standard for data scientists [3]. As argued by Helen Shen on Nature, Jupyter notebook makes data analysis easier to record, understand and reproduce [2].

The popularity of Jupyter notebook is also reflected by the expansion of public Jupyter notebook projects on Github. As of September 2018, there are over 2.5 million Jupyter projects on Github, which is 10 times more than that of 2015. One main reason leading to this popularity of Jupyter could be that Jupyter excels in literate programming [7], which allows users to formulate and depict their thoughts with text, supplemented by links, figures, and mathematical equations, as they prepare to write code cells. These code cells can then be executed along with the preparation of the notebook and the results can be permanently recorded, which can further be shared with other users as replicable computational documentation.

Despite the aforementioned benefits, the usage of Jupyter has also come with some drawbacks. As argued by Joel Grus [3], because of inadvertently running code cells out of order, developers may encounter the problem that notebooks do not behave as expected. Moreover, Jupyter notebooks might also encourage poor coding practices, e.g., it is difficult to logically organize the code into a reusable manner. Considering Jupyter notebooks are often used as tutorials or documentation for inexperienced programmers to learn practical programming skills, this poor coding practices may further be propagated into the next generation of developers.

This calls for programmatically analyzing Jupyter notebooks—to ensure the quality of the notebooks and the correctness of the code, the consistency between the code and its explanatory text, and more.

By and large, the software engineering community has not yet proposed promising approaches to automatically analyze Jupyter notebooks. To this end, we conduct in this work a preliminary study of a large set of Jupyter notebooks, aiming at checking if the code presented in the notebooks is implemented with good qualities. In a sample of 1982 “high-quality” Jupyter notebooks, our experimental results reveal that the publicly released Jupyter notebooks contain code with poorly respect to the Python style conventions and code qualities (e.g., including unused variables that are defined but never referenced and accessing deprecated features of certain Python libraries).

This preliminary study empirically shows that there is indeed a strong need to analyze Jupyter notebooks. Therefore, based on the experimental results, we further present our vision towards programmatically and systematically analyzing Jupyter notebooks. We argue that our community needs to propose promising approaches to (1) enforce good coding styles, (2) improve the quality and reliability of the code, (3) apply best practices for software quality, and (4) ensure a good balance between text and code in Jupyter notebooks.
2 PRELIMINARY STUDY

Jupyter notebooks are frequently used to present tutorials and developer documentation, from which inexperienced programmers can learn for certain programming skills. Therefore the quality of the notebooks is extremely important. As our initial attempt towards checking the quality of the code written in Jupyter notebooks, we present in this work a preliminary study. Aiming for motivating the need for automated analysis of Jupyter notebooks, we are interested in the following two research questions:

RQ1 (Quality in Programming Style): To what extent does the code in Jupyter notebooks respect recommended Python programming conventions? Programming conventions are a set of guidelines or programming styles that are recommended for programmers to follow to improve the readability and maintainability of their source code. Although those conventions are not enforced by compilers, we argue that developers (especially Jupyter notebook writers) should try their best to follow them. Compared to independent Python code, which may not be substantially read by other developers, Jupyter notebook code is expected to be read by developers (especially the newbie ones). As a result, the existence of poor coding styles in Jupyter notebooks would have a bigger impact on that of independent Python code.

RQ2 (Quality in Code Content): How does the quality of actual code in Jupyter notebooks compare with that of independent Python code? Ideally, compared with the code quality of independent Python code, the code in notebooks should come with higher quality because of the education purpose of notebooks. However, to the best of our knowledge, it is unknown to the community whether this is true or not. The code scripts written in Jupyter notebooks are scattered at different locations due to the literate programming feature, fostering unused variables that should not be included in high-quality code samples. Therefore, we investigate in this research question the appearance of unused variables and the usage of deprecated APIs in Jupyter notebooks, as our first initiative towards characterizing the code quality of Jupyter notebooks.

These two research questions are just our first initiative towards motivating the necessity of programmatically analyzing Jupyter notebooks, which, to the best of our knowledge, has not yet been investigated by the software engineering community. Our community should hence take action to propose promising (code analysis) approaches to ensure the quality of Jupyter notebooks.

2.1 Experimental Setup

Dataset. To answer the aforementioned research questions, we resort to Github to harvest a dataset (i.e., Jupyter notebooks) to support our empirical investigations. Instead of randomly cloning Jupyter notebooks, for which their qualities are unknown, we focus on a set of notable projects that are curated by the Jupyter team. Specifically, the Jupyter team has maintained a gallery of interesting Jupyter notebooks [1]. In this preliminary study, we restrict ourselves to analyzing Python-based notebooks only. After removing dead links and duplicated links, we automatically collected 1,982 notable Python-based notebooks covering various topics such as mathematics, signal processing, natural language processing, etc., as our research subject data.

2.2 RQ1: Quality in Programming Style

As the first research question, we are interested in checking if the Python code written in Jupyter notebooks respects the Python coding style. Ideally, because most of the Jupyter notebooks are provided for education, the code should be well aligned with the recommended coding conventions so that the learners will not be misled to write Python code with poor coding practices. To this end, we resort to the PEP8 checker1 to evaluate the code written in Jupyter notebooks. This check aims at checking Python code

1https://pypi.org/project/pep8/
<table>
<thead>
<tr>
<th>Python</th>
<th>Remark</th>
<th>Notebook</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>E301</td>
<td>line too long</td>
<td>E231</td>
<td>missing whitespace after <code>;</code>, or <code>:</code></td>
</tr>
<tr>
<td>E231</td>
<td>missing whitespace after <code>;</code>, or <code>:</code></td>
<td>E301</td>
<td>line too long</td>
</tr>
<tr>
<td>E111</td>
<td>indentation is not a multiple of four</td>
<td>E225</td>
<td>missing whitespace around operator</td>
</tr>
<tr>
<td>E201</td>
<td>whitespace after `('</td>
<td>E251</td>
<td>unexpected spaces around keyword / parameter equals</td>
</tr>
<tr>
<td>E265</td>
<td>block comment should start with <code>#</code></td>
<td>E703</td>
<td>statement ends with a semicolon</td>
</tr>
<tr>
<td>E302</td>
<td>expected 2 blank lines, found 0</td>
<td>E261</td>
<td>at least two spaces before inline comment</td>
</tr>
<tr>
<td>E225</td>
<td>missing whitespace around operator</td>
<td>E265</td>
<td>block comment should start with <code>#</code> comment</td>
</tr>
<tr>
<td>E251</td>
<td>unexpected spaces around keyword / parameter equals</td>
<td>E128</td>
<td>continuation line under-indented for visual indent</td>
</tr>
<tr>
<td>E701</td>
<td>Multiple statements on one line (colon)</td>
<td>E301</td>
<td>whitespace after <code>(</code></td>
</tr>
<tr>
<td>E202</td>
<td>whitespace before `('</td>
<td>E302</td>
<td>expected 2 blank lines, found 0</td>
</tr>
</tbody>
</table>

### 2.3 RQ2: Quality in Code Content

As another experiment towards the verification of the quality of the Python code in Jupyter notebooks, we check in this research question if unused variables are presented by the providers of Jupyter notebooks. Unused variables are such variables that are defined in a code cell but are never used in that cell and its subsequent cells. Fig. 3 illustrates the working process of our methodology. Given a piece of Python code, we first build an Abstract syntax tree (AST) for the involving code (as shown in step (1)). Specifically, as shown in the AST, each variable is associated with a special node indicating it is introduced into the context (i.e., Store) or referenced by the context (i.e., Load). In the second step, we perform an in-order traversal over the AST and separate all the variables that are associated with the “Store” or “Load” context into a variable access table. As shown in (2), the table contains a list of variables following their appearing order in the code. Following this table, if a variable is stored but not loaded subsequently, we will consider it as an unused variable and will flag it as such. If a variable is the result of a cell, e.g., \( x = f() \); \( x \), the variable (i.e., \( x \)) will be considered as used (i.e., “Store” in the AST tree). Following this straightforward approach, we experimentally find that 118 and 66 (out of 147) repositories (or 2150 and 1613 unused variables) whose Jupyter notebooks and independent python scripts contain unused variables, respectively. The fact that notebooks have introduced significantly more unused variables than that of independent Python scripts suggests that notebook contributors should be more careful when writing code on notebooks.

Furthermore, we take this opportunity to also look at if notable Jupyter notebooks contain code employing deprecated functions of Python libraries. Again, due to the educational purpose, we argue that deprecated functions should be avoided by notebook contributors or fixed under maintenance. Just as a representative example, we investigate the usage of deprecated functions via the Scikit-Learn library, a machine learning library for Python. Being imported by 201 notebooks, Scikit-Learn is one of the most popular libraries in our dataset. We manually go through the release notes of the Scikit-Learn library published within the past three years (since 2016). Table 2 enumerates the top 5 most used deprecated APIs. Among the 215 notebooks, 76 of them (around 35.35%) have somehow leveraged deprecated APIs (as shown in Table 2, one notebook can access multiple deprecated APIs), illustrating that deprecated APIs are recurrently used by Jupyter notebook contributors. As for independent python scripts, 15 scripts make use of deprecated APIs among 66 ones Scikit-learn libraries are called.

The aforementioned evidence experimentally shows that Jupyter notebooks, even notable ones, are inundated with low-quality code. It also suggests that long-term maintenance of Jupyter notebooks...
We argue that a good balance between the explanatory text and the code is preferred. The flow of the code and text should be also kept consistent. To achieve this, we believe that an interdisciplinary approach, which involves code analysis and comprehension, natural language processing, and artificial intelligence, could be highly useful.

4 RELATED WORK

To the best of our knowledge, our work is the first investigation motivating the necessity of deep static/dynamic analysis of Jupyter notebooks—a requirement widely overlooked by the software research community. Indeed, the only work targeting the analysis of Jupyter notebooks, as we are aware of, is the one recently conducted by Pimentel et al. [8], who empirically look at the reproducibility of Jupyter notebooks. Their experimental results show that the success rate of reproducing Python notebooks are quite low (less than 25%). This evidence further supplements our initiative towards calling on our community to propose advanced approaches for analyzing Jupyter notebooks.

Jupyter notebooks have been popularly investigated by our fellow researchers of other communities [4–6]. For example, Rule et al. [9] look at computational notebooks from the human factors point of view. Based on a large scale empirical study of computational notebooks on Github, the authors show that not all computational notebooks contain explanatory text and only a small set of notebooks have discussed the reasoning or results of the methods described. Through an interview with 15 academic data analysts, they argue that computational notebooks are considered to be messy. These results complement our work and demonstrate the necessity of analyzing Jupyter notebooks.

5 CONCLUSION

In this work, we conducted a preliminary study on a set of notable Jupyter notebooks. Our experimental results reveal that Jupyter notebooks are indeed inundated with poor coding practices. Motivated by these empirical results, we presented our vision on the need of analyzing Jupyter notebooks, appealing for the software engineering community to pay more attention to the quality and reliability of Jupyter notebooks.

REFERENCES


Table 2: Top five most frequently deprecated API uses in Scikit-Learn involved notebooks

<table>
<thead>
<tr>
<th>Deprecated API</th>
<th># Notebooks</th>
<th>Sample Notebook</th>
</tr>
</thead>
<tbody>
<tr>
<td>sklearn.cross_validation</td>
<td>56</td>
<td>justmarkham/DA14/hrrtts/dataanalysis</td>
</tr>
<tr>
<td>sklearn.grid_search</td>
<td>22</td>
<td>rashed/pattern_classification</td>
</tr>
<tr>
<td>sklearn.datasets.fetch_mldata</td>
<td>6</td>
<td>jakevdp/PythonDataScienceHandbook</td>
</tr>
<tr>
<td>sklearn.preprocessing.Imputer</td>
<td>3</td>
<td>ogrisel/parallel_ml_tutorial</td>
</tr>
<tr>
<td>sklearn.mixture.GMM</td>
<td>3</td>
<td>jakevdp/PythonDataScienceHandbook</td>
</tr>
</tbody>
</table>

is also demanded by the community, in order to deliver reliable notebooks to inexperienced learners.

3 VISION

The aforementioned preliminary study experimentally reveals that Jupyter notebooks, even for notable ones, are inundated with poor coding practices and code smells. Considering the education nature of Jupyter notebooks, the current situation, if not changed, in the long run, would certainly harm the community. The new generations of programmers are educated with poor coding styles that may lead to technical debts, and even with wrong examples that may introduce errors into critical software systems. Therefore, we argue that there is a strong need to properly analyze Jupyter notebooks before releasing them to the public.

We now enumerate some of the future directions that are needed to be addressed by the community.

- **Enforcing good coding styles.** The fact that notable Jupyter notebooks have their code written without fair respect to the recommended coding conventions suggests that there is no attempt yet for regulating Jupyter users to write source code with good programming styles. However, poor coding styles can be learned and hence propagated into thousands of programmers who might write more code with poor coding practices. Therefore, we argue that our community should implement effective approaches to enforce good coding styles in Jupyter notebooks.
- **Improving code quality and reliability.** Automated tools are expected to locate poor quality code (or code smells) and subsequently to recommend fixes to improve the code quality, so as to improve the overall quality of Jupyter notebooks. In addition to the occurrences of unused variables, the uses of deprecated functions, many other topics (such as the usage of duplicated code and inefficient algorithms, etc.) are also worthwhile to explore.
- **Apply best practices for software quality.** The Software Engineering community has produced a wealth of best practices to ensure software quality. Like other software, Jupyter Notebooks can be tested, verified, reviewed, and assessed. Users of Jupyter Notebooks should be encouraged to apply static checkers and bug finders; use tests and assertions for systematic checks; provide specifications on result properties; and use and apply domain-specific consistency checks for Notebook results. This also calls for better tools that analyze and check Notebook code—including static analysis for Python, Julia, or R code.
- **Ensure a good balance between text and code.** Jupyter notebooks embrace an innovative way of sharing knowledge, where the intricacies are not only explained but also complemented with live coding examples. However, too much water can drown the miller. We argue that a good balance between the expository text and the code is preferred. The flow of the code and text should be also...