Bug Analysis in Jupyter Notebook Projects: An Empirical Study

TAIJARA LOIOLA DE SANTANA, Federal University of Bahia, Institute of Computing (IC-UFBA), Salvador, Brazil
PAULO ANSELMO DA MOTA SILVEIRA NETO, Federal University Rural of Pernambuco (UFRPE), Recife, Brazil
EDUARDO SANTANA DE ALMEIDA, Federal University of Bahia, Institute of Computing (IC-UFBA), Salvador, Brazil
IFTEKHAR AHMED, University of California, Irvine, CA, USA

Computational notebooks, such as Jupyter, have been widely adopted by data scientists to write code for analyzing and visualizing data. Despite their growing adoption and popularity, few studies have been found to understand Jupyter development challenges from the practitioners’ point of view. This article presents a systematic study of bugs and challenges that Jupyter practitioners face through a large-scale empirical investigation. We mined 14,740 commits from 105 GitHub open source projects with Jupyter Notebook code. Next, we analyzed 30,416 StackOverflow posts, which gave us insights into bugs that practitioners face when developing Jupyter Notebook projects. Next, we conducted 19 interviews with data scientists to uncover more details about Jupyter bugs and to gain insight into Jupyter developers’ challenges. Finally, to validate the study results and proposed taxonomy, we conducted a survey with 91 data scientists. We highlight bug categories, their root causes, and the challenges that Jupyter practitioners face.

CCS Concepts: • Software and its engineering → Software testing and debugging;

ACM Reference Format:
Taijara Loiola de Santana, Paulo Anselmo da Mota Silveira Neto, Eduardo Santana de Almeida, and Iftekhar Ahmed. 2024. Bug Analysis in Jupyter Notebook Projects: An Empirical Study. ACM Trans. Softw. Eng. Methodol. 33, 4, Article 101 (April 2024), 34 pages. https://doi.org/10.1145/3641539

This material was partially based upon work supported by the INES, CNPq grant 465614/2014-0, CAPES grant 88887.136410/2017-00, and FACEPE grants APQ-0399-1.03/17 and PRONEX APQ/0388-1.03/14, and FAPESB INCITE PIE0002/2022.

Authors’ addresses: T. Loiola de Santana and E. Santana de Almeida, Federal University of Bahia, Institute of Computing (IC-UFBA), Rua Augusto Viana, s/n - Palácio da Reitoria, Canela, Salvador - CEP: 40110-909, Salvador, Bahia, Brazil; e-mails: Taijara@gmail.com, esa@rise.com.br; P. A. da Mota Silveira Neto, Federal Rural University of Pernambuco (UFRPE), Rua Dom Manuel de Medeiros, s/n, Recife, Pernambuco, Brazil, 52171-900; e-mail: paulo.motant@ufrpe.br; I. Ahmed, University of California, Irvine, CA 92697 949-824-5011; e-mail: iftekha@uci.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM 1049-331X/2024/04-ART101
https://doi.org/10.1145/3641539

ACM Trans. Softw. Eng. Methodol., Vol. 33, No. 4, Article 101. Publication date: April 2024.
1 INTRODUCTION

Due to the increased availability of data and computing resources over the past few years, data science and analytics have become important areas of investigation [3]. Data science is an emerging field that combines mathematics, statistics, computer science, and domain knowledge to derive insights from data [6, 35]. Data analytics is the multidisciplinary science of quantitatively and qualitatively examining data to draw new conclusions or insights (exploratory or predictive) or for extracting and proving (confirmatory or fact-based) hypotheses about that information for decision making and action [4].

Jupyter is a free, open source web application for creating and sharing computational documents. It offers a simple, streamlined, document-centric experience. In addition, it allows users to write documents composed of text, equations, visualizations, and code snippets and their execution results [40]. It has become the most widely used system for exploring and analyzing data [17, 23]. Data analysts use computational notebooks to write and refine code to understand unfamiliar data, test hypotheses, and build models [9].

Even with the benefits and growing popularity of Jupyter Notebooks, it has presented some problems. Wang et al. [40] analyzed a sample of 1,982 Jupyter Notebooks and found that they contain poorly written code concerning the Python style conventions, with unused variables which are defined but never referenced, and accessing deprecated functions. Pimentel et al. [23] investigated the reproducibility aspects of real notebooks using a corpus consisting of 1,159,166 unique notebooks collected from 264,023 GitHub repositories. Out of 863,878 attempted executions of valid notebooks (i.e., notebooks with defined Python version and execution order), only 24.11% executed without errors.

Other recent studies [5, 9] have identified additional problems related to name-value inconsistency, where the name and the value of a variable do not match [22], and dependencies, of which around 94% of notebooks do not formally state or document dependencies [42]. Data analysts also have called their code ad hoc, experimental, and throw-away code [12], besides describing notebooks as messy [13, 28], containing ugly code and dirty tricks in need of cleaning and polishing [28].

In addition, with the popularity of data science (Glassdoor ranks data science as the #3 job in America for 20221) and the serious consequences that a bug can bring in a data science project (UK lost nearly 16,000 COVID-19 cases by exceeding spreadsheet data limit2), analyzing and improving Jupyter Notebook projects have potentially relevant impact.

This article presents the first comprehensive study of bugs in Jupyter Notebook projects and the challenges that data scientists face in practice. Analyzing historical bugs that occurred in a system is an important step to reduce bugs [36]. It can provide relevant knowledge to develop new tools for bug detection, triage bug reports, locate likely bug spots, suggest possible fixes, and help monitor and improve quality during the development process.

The software engineering community has conducted a number of studies that investigate bugs [8, 10, 20, 26, 36, 38, 44], belonging to diverse domains, including machine learning systems. Notably, prior investigations have primarily concentrated on bug categories distinct from those encountered in Jupyter Notebook projects. These distinctions arise not only from the disparity in domains but also from the unique programming language and paradigm inherent to Jupyter

---

150 Best Jobs in America for 2022: https://www.glassdoor.com/List/Best-Jobs-in-America-LST_KQ0,20.htm
2Thousands of coronavirus cases were not reported for days in the UK because officials exceeded the data limit on their Excel spreadsheet: https://www.businessinsider.com/uk-missed-16000-coronavirus-cases-due-to-spreadsheet-failure-2020-10

ACM Trans. Softw. Eng. Methodol., Vol. 33, No. 4, Article 101. Publication date: April 2024.
Bug Analysis in Jupyter Notebook Projects: An Empirical Study

Notebook. Furthermore, the community has underscored the imperative nature of scrutinizing the quality of notebooks to improve the quality and reliability of the code [5, 40, 42].

To better understand bugs that appear in Jupyter Notebook projects, we followed three steps. First, we mined 14,740 commits from 105 GitHub open source repositories with Jupyter Notebook code. Next, we analyzed 30,416 StackOverflow posts, which gave us insights into bugs that software developers face when developing Jupyter Notebook projects. Finally, we conducted semi-structured interviews with 19 data scientists to validate the findings identified in the previous steps and understand how these bugs impact the daily life of data scientists working with Jupyter Notebook projects.

Overall, the article makes the following contributions:

— We provide a comprehensive understanding of bug classes and their underlying root causes within the context of Jupyter Notebook projects.
— We propose a comprehensive taxonomy comprising eight distinct bug categories specifically tailored for Jupyter Notebook projects.
— Drawing upon our analysis of data collected from the mining software repository study, encompassing observations from GitHub and StackOverflow, coupled with insights garnered through interviews, we provide a set of recommendations tailored to benefit both researchers and practitioners within the field. It is important to mention that to validate the study results and proposed taxonomy, we conducted a survey with 91 data scientists.
— For replication and reproducible research, we make our materials available on our project website. These include a dataset of Jupyter Notebook bugs collected from GitHub and StackOverflow, and all interview data (prompt, summary of professional and demographic information from participants, and codebook). Our artifacts can be found at the accompanying website.  

2 METHOD

This section describes the methodology used in our study to characterize Jupyter Notebook bugs, which involve GitHub repository mining, StackOverflow posts analysis, and semi-structured interviews with data scientists. To fulfill this purpose, our study aims to answer the following research questions:

— RQ1: What types of bugs are more frequent? Motivation: The types of bug identification and the comprehension of how often they appear are the first step toward better understanding and building a taxonomy of bugs in Jupyter Notebook projects. It is an important step for researchers and practitioners who use this tool daily.
— RQ2: What are the root causes of bugs? Motivation: The root cause of bugs provides additional information to understand the bugs better. Comprehending these causes can help understand what is needed to work around, fix, or improve the Jupyter environment.
— RQ3: What are the frequent impacts of bugs? Motivation: Understanding and quantifying the impact of a bug can help prioritize and scale how severe it is.
— RQ4: What challenges do data scientists face in practice on Jupyter projects? Motivation: The Jupyter Notebook is commonly adopted by data scientists from different domains, from finance systems to the car industry. Despite their growing adoption and popularity, there has been no study to understand Jupyter Notebook usage challenges from practitioners’ points of view. The current environmental limitations can also be analyzed.

3https://github.com/bugsjupyterempiricalstudy/BugJupyterPaper
To answer these questions, the following steps were performed. First, (i) a GitHub repository mining analysis was performed to characterize bugs in the context of Jupyter Notebook projects. In this analysis, only commits related to bug fixing were considered by inspecting the commit message [8, 10, 20, 38]. Second, the StackOverflow posts analysis was performed to characterize data science difficulties/issues/questions when using Jupyter Notebooks. Third, manual labeling and classification were performed in both datasets (GitHub and StackOverflow) to identify the main bug types, root causes, and impact. The coding process applied during the labeling used a set of first-cycle and second-cycle coding methods for data analysis [31]. Fourth, semi-structured interviews were conducted with data scientists to obtain and validate insights on the main issues when developing using Jupyter Notebook projects. Fifth, a survey was carried out with 91 developers to validate our findings and proposed taxonomy. Figure 1 shows the overall research methodology used in our study. All quantitative and qualitative data is available online at the accompanying website.

2.1 Repositories and Posts Selection and Mining

We chose GitHub repositories predominantly written in the "Jupyter Notebook" language in the initial step since we are interested in data science projects using this environment. The projects were sorted by star. Next, to filter out the most relevant and active projects from the 11,010 projects collected, some inclusion and exclusion criteria were applied as recommended by Munaiah et al. [21]:

- **Inclusion/Exclusion criteria:**
  - Projects written in the "Jupyter Notebook" language sorted by the number of stars in descending order. The following information was retrieved: id, name, description, URL, commits, forks, star, subscribes, issues, watch, releases, contributors, languages, create at, and last modified. This resulted in 11,010 projects.
  - Projects without description ("None") were removed. This resulted in 9,866 projects.
  - Projects written in Chinese, Japanese, and any language other than English were removed. This resulted in 8,612 projects.
  - Projects related to courses, tutorials, and books were removed. The following keywords were used to identify the projects: handbook, book, cookbook, tutorial, course, training, tracking, 2nd edition, bootCamp, workshop, hackathons, and presentations. This resulted in 6,910 projects.
  - Projects without any update (commit) in 2020 were removed. This resulted in 6,885 projects.
  - The repository must have at least 24 commits in 2020 (corresponding to two commits per month in 2020). This criterion was used to filter out inactive repositories. This resulted in 3,714 projects.
  - The repository must have at least 10 contributors in 2020. This criterion was used to eliminate irrelevant repositories (c.f. [1, 18, 25]). This resulted in 115 projects.
  - Finally, the repository must have commits in Jupyter file format (.ipynb) with the following keywords in the commit message: fix, fixes, fixed, fixing, defect, defects, error, errors, bug, bug fix, bugfixing, bugfix, bugs, issue, issues, mistake, mistakes, mistaken, incorrect, incorrect.
fault, faults, flaws, flaw, failure, correction, and corrections. This criterion was used to filter out commits not related to bug fixing. [8, 20] and resulted in 105 projects.

After filtering, we selected the top 105 Jupyter repositories, resulting in 14,740 valid commits in our GitHub raw dataset. Next, the StackOverflow posts were retrieved using the query (select Id, PostTypeId, AcceptedAnswerId, ParentId, CreationDate, DeletionDate, Score, ViewCount, OwnerUserId, OwnerDisplayName, LastEditorUserId, LastEditorDisplayName, LastEditDate, LastActivityDate, Tags, AnswerCount, CommentCount, FavoriteCount, ClosedDate, CommunityOwnedDate, ContentLicense, Title from Posts where Tags LIKE jupyter-notebook) applied to the StackOverflow API, resulting in 30,416 posts. Incomplete and inaccessible posts were removed, leaving 29,654 posts in our StackOverflow raw dataset. Finally, the .csv generated was analyzed.

8https://data.stackexchange.com/stackoverflow/query/1541588/jupyternotebookbugs
Table 1. Quantity and Percentage Considering Different StackOverflow Metrics

| Type of Bug                | Occurrences | Voting Score | Views   | Answers | Comments | Favorites |
|---------------------------|-------------|--------------|---------|---------|----------|-----------|
|                           | Qtd %       | Qtd %        | Qtd %   | Qtd %   | Qtd %    | Qtd %     |
| Environments and Settings | 1,118 43.2% | 2,077 49.9%  | 3,563,941 54.3% | 1,374 50.8% | 1,822 42.1% | 410 48.6% |
| Implementation            | 569 22.0%   | 333 8.0%     | 799,413 12.2% | 516 19.1% | 1,086 25.1% | 57 6.8%   |
| Kernel                    | 278 10.8%   | 406 9.8%     | 625,782 9.5%  | 242 8.9%  | 532 12.3%  | 81 9.6%   |
| Conversion                | 172 6.7%    | 401 9.6%     | 424,601 6.5%  | 159 5.9%  | 184 4.3%   | 68 8.1%   |
| Connection                | 160 6.2%    | 317 7.6%     | 416,104 6.3%  | 172 6.4%  | 238 5.5%   | 66 7.8%   |
| Processing                | 126 4.9%    | 333 8.0%     | 573,660 8.7%  | 124 4.6%  | 249 5.8%   | 88 10.4%  |
| Cell Defect               | 93 3.6%     | 174 4.2%     | 91,801 1.4%   | 73 2.7%   | 135 3.1%   | 57 4.4%   |
| Portability               | 69 2.7%     | 119 2.9%     | 65,904 1.0%   | 46 1.7%   | 82 1.9%    | 36 4.3%   |

2.2 Classifying and Labeling Bugs

We created a spreadsheet with all GitHub commits and filtered StackOverflow posts containing the bug type, root cause, and impact. The bug type refers to errors found in Jupyter Notebook projects and grouped into categories. The grouping process and labeling were iteratively performed from scratch by classifying and validating the commits and posts. The coding process applied used a set of first-cycle and second-cycle coding methods for data analysis [31]. The first methods are those processes during the initial coding of data. The second method, if needed, is an advanced way of reorganizing and reanalyzing data coded by first-cycle methods. We used the codes created in the first cycle to cluster into categories in the second. These clusters (bug types) then served as the source of our results. To analyze them, we investigated the title, body, pull requests, and other information that can assist us in gaining a comprehensive understanding of issues on GitHub commits. Regarding StackOverflow, we analyzed the title, body, the comments of the selected posts, and also the accepted answers [10, 20, 26, 36]. To explore the root cause, we analyzed the reason that triggered the error by analyzing the changes made in the bug-fixing commits and the answers that provide a solution in the StackOverflow [8, 20, 43, 44]. We applied root cause analysis [20] using the five whys technique [33]. Finally, regarding impact, we analyzed major effects of bugs by reading the commit message, pull request messages, and associated issues. In StackOverflow, the question description was important to understand the impact [8, 10, 36].

Table 1 shows different metrics extracted from the StackOverflow database. The metrics are as follows. The Occurrence is the number of questions from each type of bug, Voting Score is the sum of the up and down votes on a post, Views is the number of post visualizations, Answers is the number of answers of each post, Comments is the number of comments per posts, and Favorites is what we used to refer to an “interesting” post. A correlation analysis was performed to evaluate the relationship between the number of occurrences and all metrics. The Pearson correlation was 0.92 between Occurrences and Voting Score, 0.95 between Occurrences and Views, 0.99 between Occurrences and Answers, 0.99 between Occurrences and Comments, and 0.90 between Occurrences and Favorites. This indicates a high correlation in all comparisons, showing that the results will be the same as whatever metric is used in the analysis. We used the number of occurrences in both commits and posts to answer the research questions based on bug frequency.

Once the 14,740 commits and 30,416 posts were collected, the first and second authors applied the coding process using a set of first-cycle and second-cycle coding methods [31] to identify the bug types. In addition, both authors used root cause analysis [20], using the five whys technique [33] to identify the root cause and impact. The label (Bug type, root cause, and impact) identification was performed iteratively as commits and posts were parsed. It was performed until reaching a saturation state where no new categories appeared [7]. This saturation was achieved when analyzing 855 of 14,740 commits, giving us a margin of error of 3% at a 95% confidence level. In addition, analyzing 2,585 of 30,416 posts gave us a margin of error of 2% at a 95% confidence level.
Finally, having established the reliability of judgment, new commits and posts were classified by a single author. This reliability and saturation of judgment were achieved with 855 commits and 2,585 posts. During this step, some commits were discarded since they were unrelated to bugs (270 commits) or reported "typo" errors (129 commits) and improvements unrelated to bug fixing (839 commits). Some StackOverflow posts were also discarded since they mentioned some hacking and not bugs (14,914 posts) or only questions related to Jupyter Notebook usage (1,950 posts not related to Jupyter). Next, three authors (the second, third, and fourth) independently classified 145 commits randomly, and 137 posts were selected to validate the first author classification. When a conflict happens, all authors vote to solve the conflict. We measured the inter-rater agreement among the authors using Cohen’s kappa coefficient [34]. A training session was performed among the authors to clarify the labeling and what they mean. After that, Cohen’s kappa coefficient was more than 81% for bug type, 95% for the root cause, and 95% for impact, according to Landis and Koch [19], which is 'substantial agreement.'

2.3 Data Scientists Interviews

We conducted semi-structured interviews with data scientists to validate the findings identified in the previous steps and understand how these bugs impact the daily lives of data scientists working with Jupyter Notebook projects. We contacted professionals from companies and researchers who work with data science by inviting them to participate in our interview through e-mail. Once we received the acceptance reply, we started the interviews. The list of interview questions is available at the website.9

Protocol. We designed the interview prompt to understand and validate previous findings on the data scientists’ usage of Jupyter Notebook projects. It was composed of 18 open questions. The participants were informed that they could omit to answer a question to avoid arbitrary answers. The interviews started with some demographic questions and participants’ expertise. The technical section comprises questions about the Jupyter Notebook environment and the tool’s problems and challenges.

The interview pilot was performed using one data scientist. After that, the second and third authors also supported interview improvement, solving questions and difficulties based on pilot feedback. Some questions were added, updated, and removed to make the interview easier to understand and answer. The pilot interview responses were only used to calibrate the instrument, and these responses were not included in the final results. The interview instrument can be seen in the supplementary material of the article. All interviews were conducted remotely, and we recorded the audio for further analysis with the participants’ consent. The interviews took about 43 minutes on average. We transcribed the recorded interviews using QDA Miner.10

Participants. After conducting a pilot interview with one data scientist (not included in the study) as a pretest [32], 19 data scientists were interviewed. The participants in this study were chosen through convenience sampling, with a criterion stipulating a minimum of 1 year of practical experience in the field of data science across various companies and domains. Details of their diverse backgrounds are presented in Table 2. The data scientists came from 12 different companies, working in domains such as mobile games, finance, car, petrochemical, and mining. A total of 40% of participants hold a doctoral degree, 25% hold a master’s degree, 25% hold a bachelor’s degree, and 10% conducted post-doctoral studies.

9https://github.com/bugsjupyterempiricalstudy/BugJupyterPaper
10https://provalisresearch.com/products/qualitative-data-analysis-software/freeware/
Table 2. Interview participants background

| Id   | Role                  | Company Area         | Exp. (Years) |
|------|-----------------------|----------------------|--------------|
| DS1  | Data Engineer         | Petrochemical Industry| 5            |
| DS2  | Feature Owner         | Car Industry         | 8            |
| DS3  | Data Scientist        | Finance              | 13           |
| DS4  | Coordinator           | Mining Company       | 10           |
| DS5  | Data Scientist        | Finance              | 8            |
| DS6  | Software Engineer     | Engineering Solutions| 10           |
| DS7  | Data Scientist        | Mobile Games         | 11           |
| DS8  | IA Researcher         | University           | 20           |
| DS9  | Data Scientist        | IT Services          | 18           |
| DS10 | Teacher               | University           | 15           |
| DS11 | ML Engineer           | Mobile Games         | 13           |
| DS12 | Data Scientist        | Finance              | 12           |
| DS13 | Data Scientist        | Finance              | 25           |
| DS14 | Business Manager      | Finance              | 17           |
| DS15 | Data Scientist        | Finance              | 15           |
| DS16 | Data Scientist        | Finance              | 14           |
| DS17 | Data Scientist        | Finance              | 11           |
| DS18 | Data Scientist        | Finance              | 9            |
| DS19 | DS Researcher         | University           | 9            |

Analysis. The audio transcription was the first step (14 hours and 33 minutes). The first author was responsible for conducting the transcription process using the OTranscribe tool. We also performed a minor review to validate the transcriptions and clarify some answers.

Next, the first author started the coding process using the QDA Miner Lite tool. The first author and two experts iteratively worked in the coding step to reduce the subjective bias during the open coding process. We used a set of first-cycle and second-cycle coding methods for data analysis. The first-cycle methods are those processes during the initial coding of data. Second-cycle methods, if needed, are ways of reorganizing and reanalyzing data coded through first-cycle methods. All codes created in our study were later clustered into categories. Analyzing our data, we could define categories to understand the answers from interview participants. Next, the authors resolved the potential conflicts in the labels and categories. This resulted in 52 codes, seven categories, and five challenges.

3 SURVEY

To validate the study results and proposed taxonomy, we conducted a survey with 91 data scientists. We contacted companies and LinkedIn profiles and then sent the survey link.

Protocol. We created a 10-minute survey to validate our findings and proposed bug taxonomy. It was composed of 11 open questions and 14 closed questions. Nine of the 14 questions used a Likert scale (Strongly Disagree, Disagree, Neutral, Agree, and Strongly Agree). The survey also collected demographic information from respondents. The survey design followed the guidelines for personal opinion surveys by Kitchenham and Pfleeger.

We piloted out a survey with two researchers (both with doctoral degrees) with experience in the area to get feedback on the questions and their corresponding answers, difficulties faced in answering the survey, and time to finish it. We rephrased some questions and removed others to make the survey easier to understand and answer. We used an anonymous survey. However, in the end, the respondents could provide their e-mail to receive a summary of the results. The survey

11https://otranscribe.com/
instrument is available on the website. The respondents are spread out over 14 countries around four continents. The top 3 countries where the respondents come from are Brazil, India, and the United States. The professional experience of these 90 respondents varies from less than a year to 8 to 10 years.

Regarding the experience area, 86% of the respondents already work with data processing, 84% of the respondents work with data cleaning, 82% of the respondents work with exploratory data analysis, 63% of the respondents work with modeling and algorithms, 56% of the respondents work with data collection, 31% of the respondents work with data requirements, and 30% of the respondents work with communication (BI).

Data Analysis. We collected the ratings that our respondents provided for each question. Next, we converted these ratings to Likert scores from Strongly Disagree to Strongly Agree. We computed the Likert score of each question related to the taxonomy validation and plotted a Likert scale graph. This bar chart shows the number of responses corresponding to Strongly Disagree, Disagree, Neutral, Agree, and Strongly Agree.

Next, we applied the coding process [30] to analyze the answers from the survey open questions and the respondent’s perception of our taxonomy by analyzing each bug classification. We used a set of first-cycle and second-cycle coding methods for data analysis [30]. The first-cycle methods are the steps applied in the initial data coding. Second-cycle methods, if needed, are advanced ways to reorganize and reanalyze data coded in the first cycle. The codes created were clustered in categories and then analyzed to understand the answers from the respondents. To reduce the bias during the coding process, the open questions were analyzed by two authors with previous experience in this type of study. Each author analyzed the answers independently and conducted the coding process. In the end, both authors met to analyze the differences in coding. After that, we defined an “agreement level” of 0.80 measured using Cohen’s kappa [19].

4 RESULTS
This section reports the answers to our targeted research questions and findings from the data collected from GitHub, StackOverflow, and interview responses.

4.1 Types of Bugs in Jupyter Projects (RQ1)
Data scientists face different types of bugs when using Jupyter Notebooks. To understand these bugs, we classified them into different types and created an initial taxonomy. Next, we used the interviews to validate and improve the proposed taxonomy. Figure 2 shows the taxonomy, then we describe the types of bugs with examples and their occurrence percentage (in parentheses) in StackOverflow and GitHub.

Kernel Bugs | KN (StackOverflow - 10.8% | GitHub - 2.9%). This type covers the bug problems in the Kernel operation when using Jupyter Notebooks. The most common occurrences of Kernel Bugs are crashing, booting, installation, and unresponsive problems:

- **Kernel Crash**: A common bug that happens during notebook usage is when the kernel breaks. Sometimes the crashing is followed by a warning message, and the kernel is unusable in other cases. According to participants in our interviewees, it is a common bug fixed by kernel restarting.
- **Kernel Not Found**: This happens when the user starts the Jupyter Notebook, but it is not linked to a Kernel. This way, the Kernel Not Found message is displayed. Some StackOverflow posts relate this bug to installation issues.

[12]https://github.com/bugsjupyterempiricalstudy/BugJupyterPaper
Initialization Bugs: This happens during Kernel initialization, usually caused by wrong installations or conflict with the installed Kernel.

Kernel Restart: The Kernel unexpectedly restarts during its usage.

Example: Kernel bugs can cause many problems, such as data and information loss, and delays in project time, repeating the lost analyses. In bug #107937815 (GitHub) or #35673530 (StackOverflow), the Python updating generated incompatibility among packages used in the notebook and as reported by DS13:

✓ DS13: “The Kernel bugs are the most frequent ones . . . It impacts project execution time since it interrupts the data analysis.”

Conversion | CV (StackOverflow - 6.7% | GitHub - 10.6%). This comprehends bugs related to errors during notebook conversion from the .ipynb file type to other formats. Data scientists commonly use the conversion function to distribute their analyses and results to different audiences. This type refers to bugs during conversion, resulting in poorly rendered conversions or corrupted files:

Conversion Interrupted: This occurs when there is an attempt to convert a notebook to another format, but this conversion is interrupted.

Conversion with Defects: This happens when a conversion task finishes successfully, but its result contains unintended defects, such as PDFs generated without images.

Nbconvert Bugs: This is related to bugs from the nbconvert module, responsible for conversions using command line commands. In these cases, the conversion does not even start.

Example: Conversion is one of the essential Jupyter functionalities. Bugs #99244384 (GitHub) and #46415269 (StackOverflow) are examples involving the nbconvert module. It impacts the user experience, mainly for new users, as reported by DS12:

✓ DS12: “It happens with new users, which spend considerable time performing the export procedure.”

Portability | PB (StackOverflow - 2.7% | GitHub - 1.3%). This involves bugs that are related to Jupyter Notebook execution in different environments. Although this feature is one of the pillars in the Jupyter project [11], we found different bug occurrences, such as compatibility, rendering,
and environment configuration problems. Thus, this bug refers to errors obtained when rendering the notebook in environments and platforms other than the original one:

- **GitHub Bugs**: This is related to bugs when executing .ipynb files in the GitHub environment. It happens when the notebook is not rendered or shows rendering defects.
- **Nbviewer Bugs**: Similar to the previous entry, this happens when the user tries to execute the .ipynb file in the nbviewer platform.
- **Different Platforms**: This bug is related to the attempt to run the notebook on a different platform from its origin, which can happen in situations of different operating systems, machines, and browsers (even situations of execution of a .ipynb in a Google Colab, Jupyter-Lab, or any platform other than the original). This bug is generally related to the difference in configurations between the platform it was originally developed on and the platform it was ported to.

**Example**: Problems at this stage make it difficult to disseminate the analysis. Bugs #200722670 (GitHub) and #47868625 (StackOverflow) describe the need for modifications to correctly display the notebook on the GitHub environment. Participant DS11 reported a similar problem:

✓ **DS11**: “GitHub has a tool to view Jupyter Notebooks, right, but it’s kind of random, it opens whenever it wants. It doesn’t always work to open Jupyter Notebook in the browser.”

**Environments and Settings** | **ES (StackOverflow - 43.2% | GitHub - 35.6%)**. This is related to bugs in the development environment and configuration issues. It can happen due to several aspects, such as missing libraries, issues during libraries installation, compatibility between components and libraries, incompatibility with operational systems, problems with a package manager (e.g., Anaconda, PIP), and problems with installation and configuration of extensions:

- **Update and Downgrade Version**: This happens due to incompatibility with the currently installed version of a library or extension, and this library or extension needs to be updated or downgraded to work correctly.
- **Installation Bugs**: Wrong installations may cause this bug or lack of dependencies during installation.
- **Incompatible Component**: The components used in notebooks can be different, and some of them or versions of some of them generate incompatibilities for use in the same notebook. When installed or used, reports of extensions generate incompatibilities with various components.

**Example**: The environment setup is a time-consuming task. Bugs #200722670 (GitHub) and #35561126 (StackOverflow) show the cost of solving a problem due to a wrong Python version. Participant DS13 suggested that the notebook could aid the user with this setup checking:

✓ **DS13**: “Depending on the project you’re working on and the dependencies you need to install, the setup environment is a laborious task. Maybe it could be managed by Jupyter Notebooks. It is hard to say how it would be possible, but the environment creation could help to avoid configuration problems and everything else.”

**Connection Bugs** | **CN (StackOverflow - 6.2% | GitHub - 0.9%)**. This happens when connecting the notebook with external resources, such as databases, hardware, and repositories. It can occur in two ways:

- **External Resource Access Bugs**: This happens when the notebook disconnects or is no longer available to external resources.
— **Disconnection and Connection Establishment Bugs**: In this bug, the notebook itself loses connection to its server.

**Example:** Bugs #107937815 (GitHub) and #63863571 (StackOverflow) report problems related to URL and external image connection. Another connection problem reported happens when receiving data through a serial port, as highlighted by Participant DS2:

✓ DS2: “[D]uring Arduino usage some problems are difficult to know the root cause. It this situation, we looked at the Arduino board, try to disconnect and connect again, turn it on and off, replace the Arduino board to see if one of the work around solve our problem. After all tries, for some reason we get the Arduino board connected to Jupyter Notebook.”

**Processing | PC (StackOverflow - 4.9% | GitHub - 1.9%).** Data analysis often requires high processing power. Thus, memory availability and concurrency are valuable resources. Bugs of this type are related to timeout, memory errors, and longer processing tasks:

— **Memory Leak**: This occurs when a large memory allocation is incompatible with the process that is being executed. In general, the user identifies this bug when there is a delay in the execution.

— **RAM and GPU Bugs**: All bugs related to memory overflow and slow processing fall into this category.

**Example:** This bug may affect data scientists by increasing analysis time, interruptions, and data loss. Bugs #86884600 (GitHub) and #643288550 (StackOverflow) report a bug related to high-resolution images, in which a workaround should be performed to get the notebook processed. Chattopadhyay et al. [5] also reported Jupyter lack of support for handling large volumes of data, and one of our participants also reported this:

✓ DS10: “It has happened several times with me, and it happened when I was manipulating large datasets. I spent some time understanding, debugging, and identifying the root cause of this bug.”

**Cell Defect (CD) (StackOverflow - 3.6% | GitHub - 2.6%).** This involves bugs related to notebook cell rendering, such as code cells, markdown, or outputs, and it usually happens when using interactive components, latex, markdown, or cells. Next, we present some groups of this bug:

— **Layout Bugs**: This refers to cell rendering problems, such as results beyond the margin, unexpected formulas, testing formatting, blank cells, and graphics visualization problems. It can happen in any Jupyter Notebook cell.

— **Interactive Components Bugs**: This happens with components that allow the users to interact directly with the rendered cells.

**Example:** Bugs #237890763 (GitHub) and #69695030 (StackOverflow) are examples in which the user faces problems with “input()” or notebook scrollbar. Participant DS14 highlighted this as follows:

✓ DS14: “It was a very annoying error, and it frequently occurs on a personal computer as a Mac. For some reason, the cell size reduced and ended up cutting the text in half. I don’t know, I could not identify what caused it . . . and it happens a lot.”

**Implementation | IP (StackOverflow - 22% | GitHub - 44.2%).** Bugs related to implementation in general, syntax, logical, non-instantiated variables, algorithms, and semantics are examples of this type of bug. Analyzing all posts and commits, we identified the following implementation bugs:
— **Semantic Error**: Bugs related to logic misunderstandings. In this bug, the code executes correctly, but its execution generates a different output than expected, either due to poorly defined parameters or wrong algorithms.

— **Syntax Error**: Programming bugs include incorrect variable or function declaration and calls, missing or incorrectly assigned parameters, missing or misplaced parentheses, warnings or errors generated by nonstandard Python (PEP8) coding, and other general programming errors.

— **Data Science Lib Wrong Usage**: Bugs related to the inappropriate use of functions from typical data science libraries, such as Pandas, Scikit-learn, and TensorFlow.

— **Data Science Algorithm Error**: Bugs in the logic of statistical analysis or machine learning models.

**Example**: The implementation bugs are common for developers, but in Jupyter Notebooks, the chance of bugs occurring may be higher by the possibility of creating duplicated cells and cells out of order. Bugs #222507066 (GitHub) and #45946060 (StackOverflow) are examples where changes were made to fix errors of duplicate code and out-of-order cell execution. Participant DS14 reported this bug as follows:

✓ **DS14**: “When you’re writing in your notebook, you can write your code along with your text and it’s easy to lose context at some point. For example, if you write in a cell at the top of the notebook, keeping the context of the cells running part bottom of the notebook, when you run your code nothing will make sense.”

**Frequent Bug Types.** To understand the frequency of each bug type discussed previously, we statistically analyzed the labeled data. Figure 3 shows the distribution of bug types in GitHub and StackOverflow. Bugs were caught on different platforms that have different dynamics and functionality. On GitHub, bugs are reported during the development of a project, and their resolution directly impacts that project, whereas StackOverflow is a more diversified environment where bugs may be related to a specific question from a user or a project. Thus, some differences in the number of occurrences of bugs between the two databases may be related to these differences, such as Kernel bugs, which in general have the “Restart” of the Kernel as a workaround and appear with more difficulty on GitHub.
Looking at GitHub and StackOverflow datasets, the most frequent bug type was the “Environments and Settings” with 35.6% and 43.2%, respectively. It was also reinforced by the interviewers, which highlighted problems with version control, component incompatibility, wrong or missing installations, and problems with extensions. The second most frequent bug type was “Implementation” with 44.2% (GitHub) and 22% (StackOverflow).

We calculated the average annual growth (2014–2021) in the StackOverflow dataset for a more in-depth analysis of the bug types and their occurrences. We calculate the annual average growth by first calculating the annual growth per year, then calculating the annual average growth.

Four types of bugs are growing above the general annual average (Figure 4).

The “Implementation” and “Environments and Settings” bugs grow at a rate of 48% and 38%, respectively, which is reflected in the total percentage of the number of occurrences. The two bugs correspond to more than 60% of the total bug occurrences in the two analyzed databases (see Figure 3).

However, the “Portability” and “Cell Defect” bug types show an annual growth rate higher than the total average, at 39% and 37%, respectively, despite a low overall occurrence rate.

Previous research [23, 24, 42] analyzed exceptions related to reproducibility errors to get insights about the errors found. In the same way, we correlate the exceptions found with the bug types to understand them better.

The exceptions reported in StackOverflow were also collected and analyzed to understand better the most frequent type of bugs (Table 3). The “ImportError,” “ModuleNotFoundError,” and “AttributeError” frequently occur in Environments and Settings bugs, whereas the “TypeError,” “AttributeError,” and “NameError” appear in Implementation bugs. Table 3 summarizes the main problems found in each type of bug.

Finally, we used the Apriori algorithm [2] for association rules to understand the association between the bug type and the exceptions. It enables researchers to identify interesting patterns and insights from data. Table 4 shows the top 10 associations between bug types and exceptions.

It is possible to observe in Table 4 that some relationships highlighted only by the volume of occurrence (see Table 3) are validated. “ImportError” and “ModuleNotFoundError” actually have a very strong relationship with the bug “Environments and Configurations” since the confidence level is more significant than 99%. In addition, both have a “lift” above 1, reinforcing the actual
Table 3. Python exceptions per Type of Bugs

| Exception              | ES | IP | KN | CN | CV | PC | PB | CD | Total |
|------------------------|----|----|----|----|----|----|----|----|-------|
| ImportError            | 297| 3  | 11 | 8  | 4  | 0  | 0  | 0  | 323   |
| ModuleNotFoundError    | 266| 5  | 5  | 3  | 0  | 0  | 2  | 0  | 281   |
| TypeError              | 31 | 222| 4  | 0  | 1  | 0  | 0  | 0  | 258   |
| AttributeError          | 116| 101| 2  | 8  | 1  | 1  | 0  | 1  | 230   |
| NameError              | 14 | 76 | 1  | 0  | 0  | 0  | 0  | 0  | 91    |
| FileNotFoundError      | 14 | 9  | 12 | 2  | 2  | 0  | 0  | 0  | 39    |
| ValueError             | 12 | 15 | 3  | 1  | 4  | 0  | 0  | 0  | 35    |
| OSError                | 17 | 8  | 2  | 2  | 3  | 1  | 0  | 0  | 33    |
| RuntimeError           | 16 | 2  | 2  | 0  | 1  | 7  | 0  | 0  | 28    |
| SyntaxError            | 2  | 1  | 0  | 1  | 0  | 0  | 1  | 0  | 5     |
| **Total**              | 785| 442| 42 | 25 | 16 | 9  | 3  | 1  | 1015  |

(KN) Kernel, (CV) Conversion, (PB) Portability, (ES) Environments and Settings, (CN) Connection, (PC) Processing, (CD) Cell Defect, (IP) Implementation.

Table 4. Apriori analysis Python exceptions per Type of Bugs

| Antecedents               | Consequents            | Support  | Confidence | Lift  |
|---------------------------|------------------------|----------|------------|-------|
| ImportError               | Environments and Settings | 0.2188  | 0.9928     | 1.8040|
| ModuleNotFoundError      | Environments and Settings | 0.1661  | 0.9905     | 1.7998|
| NameError                | implementation         | 0.0647   | 0.9205     | 2.4730|
| TypeError                | implementation         | 0.1837   | 0.8949     | 2.4044|
| AttributeError            | Environments and Settings | 0.0871  | 0.5240     | 0.9522|
| implementation            | TypeError              | 0.1837   | 0.4936     | 2.4044|
| AttributeError            | implementation         | 0.0783   | 0.4712     | 1.2658|
| Environments and Settings | ImportError            | 0.2188   | 0.3977     | 1.8040|
| Environments and Settings | ModuleNotFoundError    | 0.1661   | 0.3019     | 1.7998|
| implementation            | AttributeError         | 0.0783   | 0.2103     | 1.2658|

existence of the occurrence relationship between exceptions and the type of bug. However, the confidence level of “AttributeError” against “Environments and Settings” is 52%. With a “Lift” less than 1, there is unlikely to be any relationship between these occurrences.

This happens similarly to “TypeError,” “AttributeError,” and “NameError” concerning the “implementation” bugs highlighted in Table 3. The result of the algorithm also reinforces the relationship between “TypeError” and “NameError” with “implementation,” as they have a confidence level of 92% and 89% consecutively and a high “lift.” However, the “AttributeError” for the “implementation” bug does not have a confidence level as high as the others, at 47%, but the “lift” greater than 1 indicates that there is a relationship between the occurrence of the exception and the bug and also the exception bug, as the inverse relationship appears in the results and with a “lift” greater than 1 as well.

**Finding 1**

The most frequent bugs in the Jupyter Notebook are those related to Environments and Settings (StackOverflow - 43.2% | GitHub - 35.6%) and Implementation (StackOverflow - 22% | GitHub - 44.2%), which also show an annual growth rate above the average, at 38% and 48%, respectively. Although the Portability and Cell Defect bugs have had fewer occurrences, they have had above-average growth over the years.
In our survey (Figure 5), we defined each bug type category and then asked the respondents if they agreed/disagreed with each bug type. Among the 90 survey respondents, 67.8% agreed, whereas 11.2% disagreed that “Kernel Bugs” should be considered as bug type. The average Likert score for this statement is 3.9 (i.e., between Agree and Neutral). Even for the “Implementation Bugs” which had more disagree answers, the average Likert score was 3.5 (i.e., between Agree and Neutral). It is important to reinforce our findings.

4.2 Root Causes of Bugs (RQ2)

The root cause of bugs helps us understand their origin and how we can correct them. Table 5 shows the distribution of bug types according to their root causes. Next, we describe each one and show its percentage of occurrence.

**Install and Configuration Problems (StackOverflow - 32.1% | GitHub - 16.3%).** Bugs with this type of root cause are common for programmers, who generally spend time configuring their development environment as completely as possible, even before starting development. However, due to the exploratory nature of the data analysis activity, there is a recurring need to use other tools and configurations to correct, improve, or extract new insights. Thus, when bugs with this type of root cause occur, the analysis process is interrupted, causing a loss of time and productivity.

*Example:* In the dataset, the bug with the highest number of views on StackOverflow was #15514593, whose root cause is a system path configuration problem when using the notebook:

> DS013: “Depending on the project you are going to develop and the dependencies you need to install, you may have problems with delays . . . The Jupyter Notebook could improve by creating environments to avoid configuration problems.”

**Version Problems (StackOverflow - 19.0% | GitHub - 22.5%).** Bugs with root causes related to incompatible versions require an update or downgrade on the notebook. In addition, these root causes may be related to components with conflicting versions. Unlike other classic IDEs (VSCode or PyCharm), Jupyter Notebook does not have alerts and intelligent version controls. The user manages the versions, and conflicts are challenging to solve.

*Example:* An example of this root cause is StackOverflow post #54966280, which highlights a cross-environment conflict issue that forced the user to downgrade the library being used. It was described too by the following interviewee’s speech:
✓ DS01: “Maybe this version problem cost me the most Man-Hour. Although I did not find the solution very complex, it was quite annoying. I even had to create a TXT with a list of these issues and how I got around them because they happen frequently.”

**Coding Error (StackOverflow - 17.6% | GitHub - 31.5%).** Implementation bugs are caused by code errors, such as incorrectly assigning variables, developing repeating structures, errors in using libraries and functions, and wrong plotting configuration. This root cause has the main characteristic of causing a Runtime Error during code execution.

**Example:** In interviews, many users report common mistakes during development and claim that this is part of the data analysis process, reported in post #62103298 (StackOverflow) and highlighted in the speech:

✓ DS03: “Bugs are part of the nature of the work, every developer will deal with bugs, mainly Jupyter Notebook bugs . . . what exists is inherent to usability, which is very common. As it is done in cells, and you can execute these cells regardless of the order, it often happens to execute in different orders and lose the correct value of the variables . . . So this, non-linearity of the execution causes many problems with execution usability.”

**Hardware and Software Limitation (StackOverflow - 6.7% | GitHub - 6.1%).** Limitations found in the software and hardware during the creation and execution of the notebook.

**Example:** Post #44719592 (StackOverflow) highlights a slowdown in data processing performance caused by operating system changes. Note the following similar report from an interviewee:

✓ DS10: “I turned everything off, and I just tried to run it (Jupyter), I had other windows open, and it went wrong, so I closed everything to test if it resolved, and it did, it took a while and I had to turn off everything running in the background.”

**Memory Error (StackOverflow - 5.6% | GitHub - 1.5%).** The root cause related to the memory overflow in an active process.

**Example:** Post #54823185 (StackOverflow) comments on a Crash in the notebook possibly related to a memory overflow. Likewise, the interviewee reports a similar problem in his speech:

✓ DS12: “This happens often, especially when the memory runs out, then the notebook crashes, and you have to run everything again.”

**Deprecation (StackOverflow - 0.9% | GitHub - 0.1%).** This happens when a component was or will be suspended.

**Example:** Posts #63279999 and #67344009 (StackOverflow) raised a Warning related to a matplotlib.

**Permission Denied (StackOverflow - 0.9% | GitHub - 0.1%).** This happens when the notebook has its permission to access an external resource blocked, such as when trying to access external databases.

**Example:** Post #63339420(StackOverflow) describes a permission error when trying to access a database external to .ipynb.

**TimeOut (StackOverflow - 1.9% | GitHub - 0.1%).** The root cause related to the timeout of an active process.
Example: Post #49611472 (StackOverflow) highlights a failure in the notebook that without a specific reason crashes the Kernel, requiring a restart. A similar situation is reported by an interviewee:

✓ DS03: “The Kernel Crash happens often; you are doing data loading, and after hours of processing, it times out, the Kernel dies, and you lose 3 hours of processing. It’s challenging.”

Logic Error (StackOverflow - 2.0% | GitHub - 13.7%). Errors in the logic of the developed code cause it. Unlike the root cause “Coding Error” mentioned earlier, this root cause usually does not generate a “Runtime Error.” This happens due to an error in the code logic. The Bugs with this root cause do not interrupt the code execution; instead, they impact code quality or expected results.

Example: Post #69041030 (StackOverflow) highlights an implementation flaw caused by incorrect handling of ‘nan’ values. As with the Root Cause Coding Error, respondents also identify logical errors as part of the data analysis process, as highlighted in the following speech:

✓ DS02: “I’ve certainly had something like this, but I think it’s more a human error than a tool error, we always type something wrong, forget something, think it’s a method and it’s a function . . . happens.”

Unknown (StackOverflow - 13.3% | GitHub - 8.1%). The root cause was not evident during the analysis. These bugs usually have no solution or alternative solutions that do not effectively solve the problem generated.

Example: Post #44634070 (StackOverflow) highlights a Kernel failure caused by an unknown situation, and similarly the respondent reports the following:

✓ DS07: “I already had a problem with the Kernel crashing for no apparent reason that I had to reset, sometimes you lose things, and it has happened a few times . . . normally you restart and it works again and all is well.”

Table 5 shows that the three most frequent root causes are Install and Configuration Problems (StackOverflow - 32.1% | GitHub - 16.3%), Version Problems (StackOverflow - 19.0% | GitHub - 22.5%), and Coding Error (StackOverflow - 17.6% | GitHub - 31.5%). These are bugs whose causes are related to component installation or configuration problems, wrong component versions, and coding problems such as semantic, logical, or syntax errors.

We could not identify all of the root causes for some bugs in our dataset. Thus, some bugs were classified using the Unknown category (StackOverflow - 13.3% | GitHub - 8.1%). This category is present in all bug types, especially those related to Kernel. It can reinforce the user’s difficulty in understanding that bug type.

The root cause of Hardware and Software Limitations (StackOverflow - 6.7% | GitHub - 6.1%) occurs when there are limitations in the software or hardware where the notebook is running. This happens in all types of bugs; however, the Memory Error (StackOverflow - 5.6% | GitHub - 1.5%) frequently appears as a root cause of the processing bugs.

The other root causes occur occasionally, such as Logic Error (StackOverflow - 2.0% | GitHub - 13.7%) in the developed code; TimeOut (StackOverflow - 1.9% | GitHub - 0.1%), when an active process achieves the time limit; Deprecation (StackOverflow - 0.9% | GitHub - 0.1%), where a component or functionality is outdated; and Permission Denied (StackOverflow - 0.9% | GitHub - 0.1%), when the permission to access an external resource is denied.

The Apriori algorithm [2] was also applied to understand the association between the bug type and root causes. Table 6 shows the top 10 associations between bug types and root causes. The
Bug Analysis in Jupyter Notebook Projects: An Empirical Study

Table 5. Frequency of Bug Type vs Root Cause

| Root Causes                  | ES | IP | KN | CV | CN | PC | CD | PB | Total |
|------------------------------|----|----|----|----|----|----|----|----|-------|
| Install and Configuration Problems | 635| 117| 36 | 0  | 56 | 12 | 44 | 0  | 42    | 0     | 0     | 0     | 6     | 7     | 12    | 3     | 970   |
| Coding Error                 | 3  | 0  | 20 | 0  | 34 | 0  | 2  | 0  | 7    | 0     | 78    | 12    | 0     | 1     | 1     | 0     | 724   |
| Version Problems             | 17 | 18 | 397| 185| 4  | 3  | 12 | 49 | 4    | 4     | 0     | 3     | 17    | 7     | 4     | 0     | 682   |
| Unknown                      | 16 | 0  | 3  | 0  | 2  | 0  | 1  | 0  | 1    | 1     | 1     | 1     | 0     | 0     | 0     | 0     | 413   |
| Hardware and Software Limitations | 21 | 1  | 1  | 0  | 1  | 0  | 0  | 1  | 0    | 0     | 0     | 0     | 0     | 0     | 0     | 0     | 224   |
| Logic Error                  | 4  | 0  | 7  | 0  | 5  | 0  | 9  | 0  | 18   | 0     | 6     | 1     | 0     | 0     | 0     | 0     | 169   |
| Memory Error                 | 2  | 1  | 32 | 114| 9  | 1  | 0  | 0  | 0    | 0     | 4     | 0     | 4     | 2     | 0     | 0     | 158   |
| TimeOut                      | 12 | 0  | 15 | 0  | 5  | 0  | 20 | 39 | 1    | 9     | 1     | 36    | 0     | 33    | 4     | 32    | 8     | 50    |
| Deprecation Denied           | 374| 158| 0  | 25 | 45 | 7  | 24 | 2  | 30   | 0     | 0     | 0     | 11    | 0     | 6     | 0     | 25    |
| Permission Denied            | 34 | 9  | 58 | 54 | 117| 2  | 60 | 1  | 38   | 2     | 1     | 0     | 22    | 1     | 14    | 0     | 25    |
| Total                        | 1,118| 304| 569| 378| 278| 25 | 172| 91 | 160  | 8     | 126   | 16    | 93    | 22    | 69    | 11    |

(KN) Kernel, (CV) Conversion, (PB) Portability, (ES) Environments and Settings, (CN) Connection, (PC) Processing, (CD) Cell Defect, (IP) Implementation.

Table 6. Apriori analysis Bug Type and Root Cause

| Antecedents                | Consequents          | Support  | Confidence | Lift  |
|----------------------------|----------------------|----------|------------|-------|
| Coding error               | implementation       | 0.1710   | 0.8255     | 3.0087|
| Version Problems           | Environments and Settings | 0.1560  | 0.7843     | 1.9057|
| Install and Configuration Problems | Environments and Settings | 0.2203  | 0.7748     | 1.8825|
| implementation              | Coding error         | 0.1710   | 0.6231     | 3.0087|
| Environments and Settings   | Install and Configuration Problems | 0.2203  | 0.5353     | 1.8825|
| Environments and Settings   | Version Problems     | 0.1560   | 0.3790     | 1.9057|

results of the algorithm reinforce the most frequent relationships between bug types and root causes, where “Installation and configuration issues” and “Version issues” are for “Environments and configurations” bugs and “Coding error” with “implementation” both have a “Lift” above 1 highlight the bidirectional relationship found in the results for all of these combinations.

Finding 2

The most frequent root causes in Jupyter Notebook projects are Configuration issues (StackOverflow - 32.1% | GitHub - 16.3%), Version issues (StackOverflow - 19.0% | GitHub - 22.5%), and Coding Error (StackOverflow - 17.6% | GitHub - 31.5%). They are the cause of most Implementation and Environments and Settings bugs. The root cause Unknown (StackOverflow - 13.3% | GitHub - 8.1%) appears more related to Kernel Bugs, suggesting a difficulty in identifying its cause.

Figure 6 shows an in-depth view of each root cause for each bug type. We defined each root cause and asked the respondents if they agreed/disagreed with each root cause. All root causes

ACM Trans. Softw. Eng. Methodol., Vol. 33, No. 4, Article 101. Publication date: April 2024.
had agreement levels higher than disagreement levels. These results reinforce our findings and validate the proposed taxonomy.

4.3 Impacts of Bugs (RQ3)

The impact caused by a bug can help increase its severity and serve as a prioritization model and alert for users. Table 7 shows the distribution of bug types according to their impact. All of them are described next.

**Crash (StackOverflow - 24.3% | GitHub - 3.3%).** Bugs whose occurrence generates a break in one or more components of the platform where the notebook is running, interrupting the operation or initialization of the entire platform. This error can occur without generating a specific warning.

*Example:* A common situation of this impact is highlighted in post #53179558 (StackOverflow) where the Kernel dies and the UI sends a message: "Dead Kernel - The Kernel has died, and the automatic restart has failed. It is possible the Kernel cannot be restarted. If you are not able to restart the Kernel, you will still be able to save the notebook, but running code will no longer work until the notebook is reopened."

**Bad Performance (StackOverflow - 3.0% | GitHub - 7.5%).** The occurrence of these bugs does not prevent the correct execution, but they impact on the decrease of its quality or performance.

*Example:* In post #67356512 (StackOverflow), the user highlights the execution of a code where, despite executing as expected, each round of execution consumes more of your RAM memory, reducing the performance of your application.

**Incorrect Functionality (StackOverflow - 13.5% | GitHub - 57.3%).** The result of these bugs generate unwanted/unexpected or incorrect outputs.
Table 7. Frequency of Impact vs Root Cause

| Impacts          | ES | IP | KN | CV | CN | PC | CD | PB | Total |
|------------------|----|----|----|----|----|----|----|----|--------|
|                  | SO | GH | SO | GH | SO | GH | SO | GH | SO | GH | SO | GH | SO | GH | 1.753 |
| Runtime Error    | 900| 197| 1  | 0  | 275| 11 | 49 | 0  | 142| 0  | 8  | 1  | 4  | 0  | 8  | 8  | 1753  |
| Incorrect Functionality | 41 | 90 | 18 | 46 | 1  | 0  | 2  | 0  | 146| 14 | 1  | 3  | 0  | 0  | 840  |
| Crash            | 142| 16 | 64 | 261| 0  | 10 | 96 | 91 | 7  | 8  | 1  | 0  | 86 | 19 | 55 | 11  | 657  |
| Bad Performance  | 7  | 1  | 472| 65 | 2  | 4  | 24 | 0  | 9  | 0  | 71 | 1  | 2  | 0  | 6  | 141  |
| Warning          | 28 | 0  | 14 | 6  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 49   |
| Total            | 1,118| 304| 569| 378| 278| 25 | 172| 91 | 160| 8  | 126| 16 | 93 | 22 | 69 | 11   |

(KN) Kernel, (CV) Conversion, (PB) Portability, (ES) Environments and Settings, (CN) Connection, (PC) Processing, (CD) Cell Defect, (IP) Implementation.

Example: Post #69351179 (StackOverflow) shows a conversion error when using nbconvert.

**Runtime Error (StackOverflow - 57.5% | GitHub - 31.2%).** Execution failure, usually accompanied by an error message.

Example: In post #70058518 (StackOverflow), the user reported an error when executing his code, which stopped mid-execution.

**Warning (StackOverflow - 1.7% | GitHub - 0.7%).** Bug with apparent correct functioning but with alert triggering to the user.

Example: In post #65666665 (StackOverflow), the user reports discomfort with the “Warning” alerts issued when executing the code and tries to find a solution to hide these alerts, as they have no impact on the final result.

The most frequent impacts are Runtime Errors, Incorrect Functionality, and Crashes. The Runtime Error was the most frequent impact in the StackOverflow (57.5%) dataset and the second most frequent in GitHub (31.2%). It is characterized by execution failures followed by an error message. It is commonly found in the Environments and Settings and Implementations types of bugs.

The impact Incorrect Functionality, which is characterized by bugs that the code can be executed but the result is not what was expected, had the highest occurrence on GitHub (57.3%) and appeared on StackOverflow as the third largest impact (13.5%).

The crash, as mentioned before, happens when an interruption in the normal operation or startup of the notebook occurs without any error message, exception, or warning. Considering the GitHub dataset, it happens (3.3%) in a smaller amount than on StackOverflow (24.3%). A possible explanation for this is that Crash is an impact that happens more with Kernel Bugs, and one of the solutions to solve Kernel Crashes is restarting Kernel, which is not showing up in fix commits.

Kernel Crash was the only bug/impact mentioned by all participants in our interview session. According to users, even being an annoying bug, it is easy to get around it by just restarting the Kernel:

✓ **DS12:** “The Kernel Crash happens a lot, especially when the memory runs out and the notebook crashes, we need to run it all over again.”

✓ **DS7:** “The Kernel Crash is usually solved by restarting and returning back to work and that’s ok.”

The other impacts had a smaller volume of occurrences. Bad Performance bugs (StackOverflow - 3.0% | GitHub - 7.5%) do not prevent the correct execution but decrease the quality or performance,
Table 8. Apriori analysis Bug Type and Impact

| Antecedents            | Consequents            | Support  | Confidence | Lift   |
|------------------------|------------------------|----------|------------|--------|
| Kernel                 | Crash                  | 0.0811   | 0.9753     | 5.0530 |
| Environments and Settings conversion | Runtime Error         | 0.3199   | 0.7773     | 1.5172 |
| Runtime Error implementation | Environments and Settings | 0.3199 | 0.6244     | 1.5172 |
| Crash                  | Kernel                 | 0.0811   | 0.4201     | 5.0530 |
| Incorrect Functionality implementation | Runtime Error | 0.0917   | 0.3833     | 1.3969 |
| implementation Incorrect Functionality | implementation | 0.0917   | 0.3340     | 1.3969 |
| Runtime Error implementation | Crash | 0.1578   | 0.5749     | 1.1222 |
| Incorrect Functionality conversion | Incorrect Functionality | 0.0549   | 0.2297     | 2.9734 |

Finding 3

The most frequent impacts from bugs in Jupyter Notebooks are Runtime Error (StackOverflow - 57.5% | GitHub - 31.2%), Incorrect Functionality (StackOverflow - 13.5% | GitHub - 57.3%), and Crash (StackOverflow - 24.3% | GitHub - 3.3%). They are the effects related to bug types Environments and Settings, Implementation, and Kernel Bugs. The Kernel Crash is a common bug/impact in the daily activities of Jupyter users and has as the main workaround solution the restart of the Kernel.

4.4 Challenges in Jupyter Notebook Projects (RQ4)

Data science is a multidisciplinary area involving physicists, mathematicians, statisticians, IT professionals, and others. This diversity is also observed in computational notebooks usage. Thus, we interviewed professional Jupyter users from the industry to understand the dynamics of bugs in Jupyter Notebook projects. We used the interviews to validate our results from mining and collect insights, impressions, and challenges about environmental usage. Next, the main challenges identified by professionals are discussed.

**Backgrounds and Requirements.** The way in which the users realize the bug can influence how to fix it. Kim et al. [15] highlighted the existence of a diversity of profiles of data science professionals. This diversity makes it possible for many Jupyter users not to come from the computing...
field or even not have enough experience to feel the need to follow patterns and strategies that help to reduce or identify errors.

Users with less experience or knowledge tend to produce messy, dirty notebooks. It can eventually generate errors, having the challenge of using a tool with a simplistic layout (compared to a traditional IDE). This simple layout can also induce the users to avoid development standards that bring gain in code quality and consequently reduction of errors.

Analyzing the interview responses with demographic data (see Table 2), we realized that, for example, software engineering knowledge is important for identifying the root cause and fixing the bug as highlighted by a professional:

✓ DS3: "Another very common thing is the knowledge that the person has. Data science is kind of a combination of statistics and computing and within that world you see people from physics, engineering and so on. The concern with having a structured, readable, documented code usually comes from the computing area, as the guy studied software engineering. So you take these people, they have an organized code."

**Lack of Support for the Software Quality.** Due to user diversity, some lack software engineering practices. Jupyter Notebooks potentialize this problem since the environment allows users to duplicate cells, drag and drop cells to different locations, and so on. In addition, notebooks do not provide any mechanism to control and support the users, so these possibilities do not harm good software engineering practices. Although Jupyter provides flexibility and allows users with different backgrounds to use it, no support is provided regarding code quality. This aspect was also mentioned by a professional:

✓ DS14: “The lack of some functionality can be a problem, it can discourage the data scientist writing better code, using good software engineering practices. I see this a lot, my codes when I’m writing in VSCode, for example, are much better than when I’m writing in Jupyter, I feel this also happens in RStudio . . . I can write code better in an IDE than in the Jupyter.”

**Testing and Debugging.** Some interviewees pointed out the lack of basic testing tools as a challenge to be addressed. They also detailed the process of fixing a bug using a trial and error approach. Among the interviewees, especially those with a software engineering background, they pointed out specific functions that could provide important support to this task:

✓ DS11: “I really miss writing unit testing and being able to lint code. The Jupyter Notebook does not have linting, everyone writes the code they want, and today we have tools, such as Black, Isort, Pylint, Flake8, Bandit, and it is very difficult for you to use them in Jupyter Notebooks. I think this lack of lint, this lack of testing is crucial for me.”

Furthermore, software debugging stands out as a crucial undertaking to enhance code quality throughout the development process. The challenge intensifies with the intricacies involved in test development. According to the insights gleaned from interviews, code debugging often shares a common hindrance—the complexity associated with code inspection. This complexity can significantly impact a data scientist’s proficiency in detecting and rectifying bugs, thereby influencing key metrics such as the count of accepted answers (representing successful problem resolutions), the tally of unanswered posts, and the time taken for an answer to be accepted on platforms like StackOverflow.

Figure 7 shows the number of questions reported with the accepted answer, considering each bug type in our StackOverflow dataset. All of the bug types have a similar average (27.9%) of accepted answers. Figure 8 shows the average time to get an acceptable answer. The average time to obtain an acceptable answer in the Jupyter Notebooks domain is 21 days, and at least four out of eight bug types are above average.
Data scientists perceive bugs differently. Their hands-on experience with software engineering techniques can change how they identify bugs. In addition, the lack of basic features in Jupyter to promote code testing and debugging can generate difficulties in fixing bugs.

**Data Analysis Deployment.** Jupyter Notebooks are used in two distinct scenarios: first, the notebook itself is a product and it is further used to replicate or perform new analyses; second, it can be encapsulated and added to another system to use it [5]. Some interviewees (DS3, DS5, DS7, DS8, DS9, DS11, DS12, DS14, DS15, DS16, DS17, and DS18) reported that Jupyter Notebook is a good tool for exploratory analysis and prototyping, but it has some limitations, such as the lack of basic features that could help convert notebook code or facilitate this process when generating a final product to be deployed:
Table 9. Jupyter history in StackOverflow: General posts related to Jupyter vs posts related to bugs in Jupyter

| Year | General Posts | Posts Related to Bugs |
|------|---------------|-----------------------|
| 2014 | 37            | 9                     |
| 2015 | 381           | 121                   |
| 2016 | 1688          | 663                   |
| 2017 | 2559          | 1054                  |
| 2018 | 3976          | 1712                  |
| 2019 | 5490          | 2482                  |
| 2020 | 7107          | 3378                  |
| 2021 | 8607          | 3879                  |
| 2022 | 531           | 251                   |

✓ **DS1**: “It has some issues, especially if you want to generate a deliverable of what you are doing inside a Jupyter Notebook.”

✓ **DS7**: “You can opt for a workaround, but it’s not trivial when you are dealing with libraries you build, functions you build, or classes . . . How do you matter, how do you build this environment, where you have solutions that use a library you created, for example. Maybe if Jupiter itself helped the user to already build the entire class structure and the entire code structure, or even if it offers tools to facilitate things like encapsulating a library, it could be something interesting too.”

Many interviewees use Jupyter Notebook in industrial and robust projects and deploy it inside company systems to perform the analysis:

✓ **DS7**: “When you intend to deploy the Jupyter Notebook code inside the system code, it is not a trivial task since some changes need to be performed to be properly deployed in production environment. It would be interesting if the Jupyter Notebook provide tools to support this deployment.”

**Bad Programming Practices.** Except for DS1, all other interviewees whose academic background was not computer science started in the field of data science and/or programming in Python through Jupyter, which raises the concern about what culture of code quality in computational notebooks is being propagated.

The Jupyter Notebook has appeared in the StackOverflow annual survey since 2017, and as shown in Table 9, the posts and bugs on StackOverflow have only increased. This reinforces the importance of evolving the tool with features to mitigate bugs and help data scientists do exploratory analysis, prototyping, and computational narratives and generate products without losing quality. These aspects were also highlighted by a professional during our interviews:

✓ **DS11**: “Another mistake is also . . . Generally when we write a Jupyter Notebook we do not care much about the quality of the code, we write code in almost any way, we do not care about a Lint, things like that, right. People do not bother to test too, so I think it is one of Jupyter biggest problems. We do not appreciate our code, we do not care much about code quality, we do not care much about unit tests.”

Finding 5

Transforming an analysis developed in Jupyter into a product can be one of the most important features for data scientists in the industry. However, there is still a lack of resources to improve the code quality and this transition process. Some users have been looking for alternative solutions that combine the benefits of a Jupyter Notebook and an IDE. The lack of resources focused on code quality can also lead new data scientists to have bad programming habits.

13https://insights.StackOverflow.com/survey
5 DISCUSSION

In this section, we discuss the implications of our study’s results, particularly the implication for tool builders, researchers, and data scientists.

As highlighted in Finding 1, the most frequent bugs in the Jupyter Notebook are related to Environments and Settings, consisting of 43.2% of analyzed posts from StackOverflow and 35.5% of the issues analyzed in GitHub. The majority of the root causes of this bug category were configuration issues, version issues, and deprecation-related issues, suggesting that a significant amount of effort is spent by data scientists dealing with these issues. If software engineering research can aid data scientists, this would potentially save a substantial amount of time.

Incorrect algorithm implementations cause many bugs (44.2% of GitHub issues and 22% of StackOverflow posts). Most of them are related to coding and logical errors resulting in “Incorrect Functionality” (see Table 4). We posit that this is happening because data scientists are not familiar with the existing software quality assurance techniques such as unit testing, bug localization, and repair. Our intuition is corroborated by the findings reported in Finding 4. This calls for action from software engineering community researchers and practitioners alike to increase awareness about the existing techniques and make such tools available for data scientists. In addition, researchers need to develop tools that can seamlessly integrate with the Jupyter Notebook, making it easy for data scientists to adopt the techniques.

Our study highlights the lack of functionalities that are standard practice in software engineering. For instance, version control systems (i.e., Git) are standard tools used in software development. However, in Jupyter Notebook development, this is not standard practice yet, as mentioned by interviewees DS7 and by DS11 previously. Since existing version control systems do not compare differences in the generated Graphical User Interface (GUI) components, it is difficult to identify the differences between GUI components across different versions of a given notebook. While it is possible to use version control with Jupyter files, as they can be treated as text files, it is not possible to apply the same version control in a way that allows visualizing the differences in a graphical manner through the graphical interface. So, a tool helping developers to compare GUI changes instead of only textual changes can help Jupyter Notebook developers significantly. Interviewees also highlighted the lack of functionality to preview, explore, and interact with the raw dataset before starting analysis and modeling, which can be a better alternative than the notebook cell visualization. Another common feature requested by interviewees was advanced debugging capabilities, such as a viewer of the variables defined in the notebook and the values assigned in each cell. We posit that such easy-to-use debugging capabilities will help reduce the significant time it takes to identify, analyze, and fix implementation errors in Jupyter Notebooks.

In our study, we noticed that Jupyter Notebook is a very useful solution when it comes to analyzing, investigating, and exploring data. A total of 95% of our respondents reported understanding that the main (or only) usefulness is in these steps; in contrast to other traditional IDEs like R-Studio or VS-Code, its simple layout facilitates and highlights the analysis performed. Although almost all respondents reported this tool’s potential in data exploration, 79% of them reported difficulty in transforming the analysis done in the Jupyter Notebook into code to be put into production and the lack of features that facilitate cleaning and adaptation of the code for transposition.

Finally, with the analysis of bugs and interviews, we brought a non-exhaustive list (Table 10) of features desired by users. Some features already have ready-made extensions, but in our analysis, the use of some extensions is not trivial, in addition to generating compatibility, version, and configuration errors. That is why it is important to have an extension with a unified package of solutions for Jupyter Notebook or that some of these solutions are in the standard version of the tool.
Table 10. Features Mentioned by Respondents

| Feature                      | Description                                                                                                                                                                                                 | Occurrences |
|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| Indentation corrector       | For Python, indentation is important. Jupyter allows indentations to be the developer’s responsibility while writing. The indentation corrector identifies and corrects wrong indentations at development time.               | 31.6%       |
| Syntax highlighting         | Function that inspects the code indicating syntax errors, structure errors, and so forth.                                                                                                                  | 15.8%       |
| Data Preview                | Functionality to preview and explore the raw dataset before starting analysis and modeling; a better alternative than the notebook cell visualization.                                                        | 15.8%       |
| Graphic Interaction         | Functionality to manually interact with the graphs generated during data analysis.                                                                                                                        | 15.8%       |
| Multi-Languages per Notebook| Possibility to use other programming languages in the same notebook.                                                                                                                                       | 10.5%       |
| Version control             | Notebook change manager.                                                                                                                                                                                   | 15.8%       |
| Text Editor                 | More advanced code editing features.                                                                                                                                                                       | 26.3%       |
| Development Framework       | Framework that provides a base architecture adapted for the notebook.                                                                                                                                     | 10.5%       |
| Real-Time Collaboration     | Functionality to support people working together at the same time, even if they are in different places.                                                                                                  | 15.8%       |
| Variable Manager            | Viewer of the variables defined in the notebook and the values assigned in each cell.                                                                                                                    | 15.8%       |
| Connection between Notebooks| Functionality for the user to visualize his set of notebooks and make calls to notebooks and cells external to the current notebook.                                                                     | 5.3%        |

6 LESSONS LEARNED

When it comes to the execution of surveys, interviews, and data mining, there are several valuable lessons for researchers and practitioners. Here are some important lessons learned related to each of these methods:

- **Data mining:**
  - Define clear research questions: Clearly define the research questions or objectives before engaging in data mining. This will help focus the analysis and guide the selection of appropriate data mining techniques.
  - Gather high-quality data: The quality of data used in data mining greatly impacts the results. Ensure that the data collected is accurate, complete, and relevant to the research questions at hand. Preprocess and clean the data as necessary to remove noise or inconsistencies.

14https://github.com/kenkoooo/jupyter-autopep8
15https://github.com/jupyterlab-contrib/jupyterlab-variableInspector
— **Validate and interpret results**: Validate the results of data mining models or algorithms to ensure their reliability and accuracy. Interpret the findings in the context of the research objectives and consider potential limitations or biases in the data.

- **Interview**:
  - **Prepare thoroughly**: Prior to conducting interviews, prepare a well-defined interview guide or set of questions. Familiarize yourself with the topic and execute a pilot to refine your questions.
  - **Ensure confidentiality and anonymity**: Assure interviewees that their responses will be kept confidential and, if desired, provide options for anonymity. This helps create a safe space for participants to share their thoughts and experiences more openly.
  - **Use coding techniques**: Utilize coding techniques to explore interviewees’ responses in more depth by identifying key points and insights from the transcribed interview.

- **Survey**:
  - **Clearly define research objectives**: Before conducting a survey, it is crucial to clearly define the research objectives and identify the specific information needed. This helps in designing relevant survey questions and collecting meaningful data. It will also help correlate the answers from the set of questions.
  - **Use a mix of question types**: A combination of different question types (e.g., multiple choice, Likert scale, open ended) can provide a comprehensive understanding of the topic. However, keep the survey length reasonable to avoid respondent fatigue.

### 7 THREATS TO VALIDITY

In this section, we discuss several threats to validity for our study.

- **Projects Selection**: We have not analyzed proprietary repositories, and our findings are limited to open source projects, which may not be representative and comprehensive. We mitigate this limitation by mining a large number of (105) open source projects from GitHub selected based on a well-defined set of criteria.

- **Bug Selection**: We only collected the issues with a set of keywords in the commit message (see Section 2.1). Even with a pre-defined list also used in previous research [8, 20], it is possible to miss some real bugs that do not have these keywords.

- **Manual Analysis of Bugs**: Our study involved manual inspection of bugs, which is a potentially error-prone process. To mitigate this threat, three authors (the second, third, and fourth) analyzed the bugs separately. Next, all divergence in the process was discussed with the whole team until a consensus was reached. Our results are also online for public scrutiny.

- **Quality of Posts**: The trustworthiness of the posts collected from StackOverflow can be a threat to our study. To mitigate this threat, we used an approach similar to that of Islam et al. [10], which collected the posts based on a score of at least 5 and the reputation of users asking the questions. This score can be used as a good indicator to trust the post as a good discussion topic among the developers’ community that cannot merely be solved using an internet search. In addition, the reputation of the users asking questions on StackOverflow can be a threat to the quality of the posts. We only investigated top-scored posts which are from users with different ranges of reputations ranging from novices to experts.

- **Taxonomy**: The final taxonomy depends on the collected commits, posts on StackOverflow, and authors’ judgment. We mitigated this threat by investigating 105 Jupyter Notebook repositories, 14,740 valid commits, and 30,416 StackOverflow posts and performing a cross-validation survey.

- **Interviews**: The interviews were conducted with open-ended questions, where the participants were asked to express their perceptions and opinions. The interviews were conducted at 12 different companies, and when these happened in the same company, the participants were informed
not to talk to each other about it to avoid bias. In addition, we did our best to select experienced professionals at each company to avoid our sample from not being mature enough to have expressive knowledge about our area of investigation. Another aspect that is critical for validity is the quality of the material used in the study. Thus, to ensure that the interview prompt had high quality, a pilot interview was conducted with a professional data scientist. Finally, to avoid the threat of concluding false conclusions based on the interview data, we carefully validated our interviews and findings with the participants as we performed our analysis, sometimes asking for clarifications.

8 RELATED WORK

In this section, we discuss the main work related to our study.

Jupyter Notebooks: Extensions. The Jupyter Notebook project aims to provide the data science community with a simple graphic interface to promote the computational narrative based on usability, collaboration, and portability [11]. Some studies have been proposing different ways to improve these aspects.

Rule et al. [27] investigated how cell folding can contribute to notebook navigation and reading. They developed an extension for it, but in some cases, folded sections were ignored or increased the time of notebook revisions. This shows how the analysis process in a notebook can be confusing and hard to understand, especially in large documents. Head et al. [9] developed a solution to collect and organize code versions, helping the analyst to study, review, and recover old codes and analysis.

Computer notebooks unify text, code, and visual outputs, and being able to interact with the graphical outputs increases the data analysis power of scientists. Kery et al. [14] developed an API for this.

With respect to supporting reproducibility, Wang et al. carried out two studies. The first one is to recover the notebook’s reproducibility with a tool that generates possible execution schemes [39], and the second one is to retrieve and install the notebook’s experimental dependencies [42].

Our study is not focused on producing new features for Jupyter Notebooks. We analyze, identify, and classify bugs in the Jupyter Notebook to provide a systematic overview of bugs and developer challenges and an initial body of knowledge for future work on gaps and limitations in the daily use of the Jupyter Notebook.

Our work understands the importance of notebooks for the data science community. It intends to study their weaknesses and bring important information to evolve the tool and improve the user experience. However, although this study can inspire new features, we intend to produce something other than new features for the Jupyter Notebook. Instead, we review, identify, and classify bugs in the Jupyter Notebook to provide a systematic overview of bugs and developer challenges. We also provide initial knowledge for future work on gaps and limitations in everyday Jupyter Notebook use.

Jupyter Notebooks: How Data Scientists Use Them. Some studies explore how the data scientists use the notebooks in their daily usage. Code duplication, for example, is a common practice from data scientists. Koenzen et al. [17] studied how these duplications happen and identified that although there is an approximately 8% rate of duplicate code in GitHub databases, users prefer not to duplicate their own code.

Data analysis processes provide insights that need to be demonstrated, shared, and disseminated. Wang et al. [37] studied the real-time collaboration and identified that working on synchronous notebooks encourages exploration and reduces communication costs, but the resources currently available for this imply the need for greater team coordination.

ACM Trans. Softw. Eng. Methodol., Vol. 33, No. 4, Article 101. Publication date: April 2024.
Our study also intends to understand more about notebook usage, but we focus on something other than a specific type of usage, as highlighted by the studies. We focus on mapping and quantifying the bugs in the user’s daily life to understand its dynamics and how their specificities can bring more difficulty, deficiency, or problems with the tool.

**Jupyter Notebook: Notebook Quality.** Chattopadhyay et al. [5] conducted a study that involved observing 5 data scientists at their work with computational notebooks. They interviewed 15 data scientists, and next surveyed 156 data scientists. They cataloged nine main problems and difficulties faced by data scientists using computer notebooks. Unlike this research, our study highlights the challenges faced by users from the perspective of real bugs that data scientists encounter in their daily work.

Rule et al. [29] analyzed the structure of 1 million notebooks to assess whether they were being in such a way that the development was reflected in well-structured computational narratives. They identified that most of the notebooks are built without proper cleaning or documentation, making readability, replication, code reuse, and consequently reproducibility a difficult task. Pimentel et al. [23] conducted a large-scale study on notebook reproducibility problems. Their results show that only 24.11% of notebooks run without errors, and out of that percentage, only 4.03% are able to produce the original results. Later, they conducted another study that conducted a more detailed analysis [24]. While the authors are interested in analyzing notebooks regarding their structure, our study aims to understand the notebook code quality throughout the existing bugs.

Investigating the coding quality of Jupyter Notebooks, Wang et al. [41] developed a preliminary study where the results revealed a high amount of bad coding practices in Jupyter Notebooks. However, unlike the previous study, Patra and Pradel [22] decided to focus on a single type of coding inconsistency that appears in Jupyter Notebooks—Name-Value—and its implications for understanding and maintaining code. Unlike the previous studies that cite specific bugs, our work categorizes and quantifies types of bugs and root causes in the domain of Jupyter Notebooks.

All of these studies evaluate the quality of notebooks and how the pain points pointed out by users are related to this quality. Similarly, we aim to cross-reference bugs users report with their complaints and how this impacts their use. Empirically, we observe a relationship between the quality of the notebook developed and the occurrence of some bugs, but we do not intend to deepen this relationship between them. However, some of these studies cite or highlight some bugs, but they need to go into depth, as we are doing here in this work.

**Empirical Studies on Bugs.** Some related works are not directly related to data science. For example, Zhang et al. [44] mined bugs in deep learning applications based on TensorFlow. They analyzed GitHub commits, pull requests and issues, and StackOverflow questions. Using similar mining strategies and same data sources, Islam et al. [10] extended the search for other popular deep learning libraries: Caffe, Keras, TensorFlow, Theano, and Torch. In addition, Thung et al. [36] analyzed bugs in machine learning systems, but their research used the issues reported on the Jira database as a data source. However, to the best of our knowledge, this is the first empirical study of bugs in Jupyter Notebook projects.

Other studies focused on bugs by only analyzing the GitHub projects in different domains, such as bugs in autopilot software in unmanned aerial vehicles [38], bugs in IoT systems [20], bugs in autonomous vehicles [8], and bugs involving Infrastructure as Code Scripts [26].

All previous research focused on analyzing specific aspects of bugs, such as symptoms, commonality, bug evolution, bug-prone stages, and bug detection. Our work is a preliminary study that focuses on providing the characterization of bugs in Jupyter Notebook projects, such as the types of bugs, the potential root causes, their frequency, and the impact and challenges for data scientists.
Similar to previous articles, we also studied bugs, their main characteristics, and how this impacts the user’s challenges in their daily lives. However, we focused only on specific characteristics: root cause, bug type, and impact. Furthermore, although two of these articles are in the domain of data science, these articles focus on something other than computational and/or Jupyter Notebooks.

Studies on Bugs with Taxonomies. Although different domains, four studies had a similar approach to our study: research into bugs in autopilot software in unmanned aerial vehicles [38], bugs in IoT systems [20], bugs in autonomous vehicles [8], and bugs involving Infrastructure as Code Scripts [26].

Considering their particularities, all of them aimed to identify and characterize the bugs in the researched domain, contributing to a deeper understanding of the problems. Thus, processes such as database mining (e.g., GitHub), manual classification, and labeling were common in all studies for the empirical construction of a first bug taxonomy.

Interviews and surveys were used in some of these studies as a way of validating and understanding how bugs can create challenges in the developer’s daily life.

Moreover, it is crucial to underscore that in the creation of a new taxonomy for bugs through empirical means, such as the one conducted on autonomous vehicles [8], have observed and adapted taxonomies from prior research to align with their specific research domain. Still, the need for manual analysis and recursive cross validations between experts were crucial in all research to achieve the domain-specific taxonomy.

Strategies such as the “five whys technique” [20] or “Cohen’s kappa” [26] were used as a way of establishing well-defined rules so that manual interaction processes were possible identify the saturation point of classifications and labeling.

In our study, we propose to carry out all of the questions, analyses, and validations of previous research that would add value to a better understanding of bugs and challenges for data scientists in the field of Jupyter computing notebooks.

9 CONCLUSION

In this work, we conducted a large-scale empirical study to characterize bugs in Jupyter Notebook projects. First, we analyzed 855 commits from 105 GitHub open source repositories. Next, we analyzed 2,585 StackOverflow posts, which gave us insights into bugs that data scientists face when developing Jupyter Notebook projects. Finally, we conducted semi-structured interviews with 19 data scientists from 12 companies to validate the findings. We proposed a taxonomy of Jupyter Notebook specific bugs by analyzing these bugs. In particular, we identify eight classes of bugs, 10 types of root causes, and the impact of bugs.

The most frequent bugs in the Jupyter Notebook are those related to Environments and Settings and Implementation. Regarding the root causes, the most frequent were Configuration issues, Version issues, and Coding Errors. They are the cause of most Implementation and Environments and Settings bugs. The most frequent bug impact was Runtime Error, followed by Incorrect Functionality. In addition, we found that the data scientist’s background determines how the bugs are identified, highlighting the importance of testing and debugging tools. Finally, we identified the Jupyter Notebook deployment as a challenging and poorly supported task.

We believe that this study can facilitate practitioners’ understanding of the nature of bugs and define possible strategies to mitigate them. Our findings can guide future research in related areas, such as developing tools for detecting and recommending bug fixes in the Jupyter Notebook and an empirical study to understand the issues in private projects.
ACKNOWLEDGMENTS
We thank all the software professionals who participated in our interviews and survey.

REFERENCES
[1] Amritanshu Agrawal, Akond Rahman, Rahul Krishna, Alexander Sobran, and Tim Menzies. 2018. We don’t need another hero? The impact of “heroes” on software development. In Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE’18). ACM, 245–253. https://doi.org/10.1145/3183519.3183549
[2] Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast algorithms for mining association rules in large databases. In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB’94). 487–499.
[3] Andrew Begel and Thomas Zimmermann. 2014. Analyze this! 145 questions for data scientists in software engineering. In Proceedings of the 36th International Conference on Software Engineering (ICSE’14). ACM, 12–23. https://doi.org/10.1145/2568225.2568233
[4] Longbing Cao. 2017. Data science: A comprehensive overview. ACM Comput. Surv. 50, 3 (June 2017), 1–42.
[5] Souti Chattopadhyay, Ishita Prasad, Austin Z. Henley, Anita Sarma, and Titus Barik. 2020. What’s wrong with computational notebooks? Pain points, needs, and design opportunities. In Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI’20). ACM, 1–12.
[6] Vasant Dhar. 2013. Data science and prediction. Commun. ACM 56, 12 (Dec. 2013), 64–73.
[7] P. Fusch and L. Ness. 2015. Are we there yet? Data saturation in qualitative research. In Qualitative Report. Nova Southeastern University, Minneapolis, MN, 1408–1416.
[8] Joshua Garcia, Yang Feng, Junjie Shen, Sumaya Almanee, Yuan Xia, and Qi Alfred Chen. 2020. A comprehensive study of autonomous vehicle bugs. In Proceedings of the 42nd International Conference on Software Engineering (ICSE’20). ACM, 385–396.
[9] Andrew Head, Fred Hohman, Titus Barik, Steven Mark Drucker, and Robert DeLine. 2019. Managing messes in computational notebooks. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI’19). ACM, 270. https://doi.org/10.1145/3290605.3300500
[10] Md. Johirul Islam, Giang Nguyen, Rangeet Pan, and Hridesh Rajan. 2019. A comprehensive study on deep learning bug characteristics. In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE’19). ACM, 510–520.
[11] Project Jupyter. 2015. Project Jupyter: Computational Narratives as the Engine of Collaborative Data Science. Retrieved January 26, 2024 from https://blog.jupyter.org/project-jupyter-computational-narratives-as-the-engine-of-collaborative-data-science-2b5fb94c3c58
[12] Sean Kandel, Andreas Paepcke, Joseph M. Hellerstein, and Jeffrey Heer. 2012. Enterprise data analysis and visualization: An interview study. IEEE Trans. Vis. Comput. Graph. 18, 12 (2012), 2917–2926.
[13] Mary Beth Kery, Marissa Radensky, Mahima Arya, Bonnie E. John, and Brad A. Myers. 2018. 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 (CHI’18). ACM, 174. https://doi.org/10.1145/3173574.3173748
[14] Mary Beth Kery, Donghao Ren, Fred Hohman, Dominik Moritz, Kanit Wongsuphasawat, and Kayur Patel. 2020. Mage: Fluid moves between code and graphical work in computational notebooks. In Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology (UIST’20). ACM, 140–151. https://doi.org/10.1145/3379337.3415842
[15] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. 2016. The emerging role of data scientists on software development teams. In Proceedings of the 38th International Conference on Software Engineering (ICSE’16). ACM, 96–107. https://doi.org/10.1145/2884781.2884783
[16] Barbara A. Kitchenham and Shari L. Pfeeger. 2008. Personal opinion surveys. In Guide to Advanced Empirical Software Engineering. Forrest Shull, Janice Singer, and Dag I. K. Sjøberg (Eds.). Springer London, 63–92.
[17] Andreas P. Koenzen, Neil A. Ernst, and Margaret-Anne D. Storey. 2020. Code duplication and reuse in Jupyter Notebooks. In Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC’20). IEEE, 1–9.
[18] Rahul Krishna, Amritanshu Agrawal, Akond Rahman, Alexander Sobran, and Tim Menzies. 2018. What is the connection between issues, bugs, and enhancements? Lessons learned from 800+ software projects. In Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE’18). ACM, 306–315. https://doi.org/10.1145/3183519.3183548
[19] J. Richard Landis and Gary G. Koch. 1977. The measurement of observer agreement for categorical data. Biometrics 33, 1 (1977), 159–174.
Bug Analysis in Jupyter Notebook Projects: An Empirical Study

[20] Amir Makhshari and Ali Mesbah. 2021. IoT bugs and development challenges. In Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering (ICSE’21). IEEE, 460–472.

[21] Nuthan Muniaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2016. Curating GitHub for engineered software projects. PeerJ Prepr. 4 (2016), e2617. https://doi.org/10.7287/peerj.preprints.2617v1

[22] Jibesh Patra and Michael Pradel. 2021. Nalin: Learning from runtime behavior to find name-value inconsistencies in Jupyter Notebooks. In Proceedings of the 44rd IEEE/ACM International Conference on Software Engineering (ICSE’22). ACM.

[23] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. 2019. A large-scale study about quality and reproducibility of Jupyter Notebooks. In Proceedings of the 16th International Conference on Mining Software Repositories (MSR’19). IEEE, 507–517.

[24] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. 2021. Understanding and improving the quality and reproducibility of Jupyter Notebooks. Empir. Softw. Eng. 26, 4 (2021), 65. https://doi.org/10.1007/s10664-021-09961-9

[25] Akond Rahman, Amritanshu Agrawal, Rahul Krishna, and Alexander Sobran. 2018. Characterizing the influence of continuous integration: Empirical results from 250+ open source and proprietary projects. In Proceedings of the 4th ACM SIGSOFT International Workshop on Software Analytics (SWAN@ESEC/SIGSOFT FSE’18). ACM, 8–14. https://doi.org/10.1145/3278142.3278149

[26] Akond Rahman, Effat Farhana, Chris Parnin, and Laurie Williams. 2020. Gang of eight: A defect taxonomy for infrastructure as code scripts. In Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE’20). ACM, 752–764.

[27] Adam Rule, Ian Drosos, Aurélien Tabard, and James D. Hollan. 2018. Aiding collaborative reuse of computational notebooks with annotated cell folding. Proc. ACM Hum.-Comput. Interact. 2, CSCW (Nov. 2018), Article 150, 12 pages. https://doi.org/10.1145/3274419

[28] Adam Rule, Aurélien Tabard, and James D. Hollan. 2018. Exploration and explanation in computational notebooks. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI’18). ACM, 32.

[29] Olivier Serrat. 2017. The five whys technique. In Knowledge Solutions. Springer Singapore, 307–310. https://doi.org/10.1007/978-981-10-0983-9_32

[30] Seyyed Ehsan Salamati Taba, Foutse Khomh, Ying Zou, Ahmed E. Hassan, and Meiyappan Nagappan. 2013. Predicting bugs using antipatterns. In Proceedings of the 2013 IEEE International Conference on Software Maintenance. 270–279. https://doi.org/10.1109/ICSM.2013.38

[31] Jiawei Wang, Li Li, and Andreas Zeller. 2020. Assessing and restoring reproducibility of Jupyter Notebooks. In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE’20). ACM, 138–149.

[32] Jiawei Wang, Li Li, and Andreas Zeller. 2020. Better code, better sharing: On the need of analyzing Jupyter Notebooks. In Proceedings of the 23rd IEEE International Symposium on Software Reliability Engineering (ISSRE’12). IEEE, 271–280.

[33] April Yi Wang, Anant Mittal, Christopher Brooks, and Steve Oney. 2019. How data scientists use computational notebooks for real-time collaboration. Proc. ACM Hum.-Comput. Interact. 3, CSCW (Nov. 2019), Article 39, 30 pages. https://doi.org/10.1145/3359141

[34] Dinghua Wang, Shuqing Li, Guanping Xiao, Yepang Liu, and Yulei Sui. 2021. An exploratory study of autopilot software bugs in unmanned aerial vehicles. In Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE’21). ACM, 20–31.

[35] Jiawei Wang, Tzu-Yang Kuo, Li Li, and Andreas Zeller. 2020. Assessing and restoring reproducibility of Jupyter Notebooks. In Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering (ASE’20). ACM, 138–149.

[36] Jiawei Wang, Li Li, and Andreas Zeller. 2020. Better code, better sharing: On the need of analyzing Jupyter Notebooks. In Proceedings of the 42nd International Conference on Software Engineering, New Ideas, and Emerging Results (ICSE-NIER’20). ACM, 53–56.
[41] Jiawei Wang, Li Li, and Andreas Zeller. 2020. 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 (ICSE/NIER’20). ACM, 53–56. https://doi.org/10.1145/3377816.3381724

[42] Jiawei Wang, Li Li, and Andreas Zeller. 2021. Restoring execution environments of Jupyter Notebooks. In Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering (ICSE’21). IEEE, 1622–1633.

[43] Chenyang Yang, Shurui Zhou, Jin L. C. Guo, and Christian Kästner. 2021. Subtle bugs everywhere: Generating documentation for data wrangling code. In Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE’21). IEEE, 304–316. https://doi.org/10.1109/ASE51524.2021.9678520

[44] Yuhao Zhang, Yifan Chen, Shing-Chi Cheung, Yingfei Xiong, and Lu Zhang. 2018. An empirical study on TensorFlow program bugs. In Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA’18). ACM, 129–140.

Received 11 October 2022; revised 20 December 2023; accepted 3 January 2024