

PGCOMP - Programa de Pós-Graduação em Ciência da Computação  
Universidade Federal da Bahia (UFBA)  
Av. Milton Santos, s/n - Ondina  
Salvador, BA, Brasil, 40170-110

<https://pgcomp.ufba.br>  
[pgcomp@ufba.br](mailto:pgcomp@ufba.br)

One of the new technologies driving data science projects is Computational Notebooks, which allow users to build data-oriented codes, emphasizing the data collected and the analysis performed. Although Computational Notebooks have gained visibility, some problems and solutions already discussed and studied by the software engineering community must be addressed, impacting the quality of the developed software and, consequently, data analysis. In addition, neglecting these aspects can lead to the spread of bad programming practices. 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 are available to understand Jupyter development challenges from the practitioners' point of view. This dissertation systematically studies 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 Stack Overflow posts, which gave us insights into bugs that practitioners face when developing Jupyter Notebook projects. We conducted nineteen interviews with data scientists to uncover more details about Jupyter bugs and to gain insight into Jupyter developers' challenges and finally, to validate all the information obtained, we carried out a survey with several data scientists and an analysis with association rules using the Apriori algorithm. We propose a bug taxonomy for Jupyter projects based on our results. We also highlight bug categories, their root causes, and Jupyter practitioners' challenges.

Palavras-chave: Jupyter Notebooks, Bugs, Interviews, Mining Software Repositories (MSR), Stack Overflow, Empirical Study, Computational Notebooks, Data-driven Development

MSC | 179 | 2024

Bug Analysis in Jupyter Notebook Projects: An Empirical Study

Taijara Loiola de Santana

# Bug Analysis in Jupyter Notebook Projects: An Empirical Study

Taijara Loiola de Santana

Dissertação de Mestrado

Universidade Federal da Bahia

Programa de Pós-Graduação em  
Ciência da Computação

March | 2024

UFBA





Universidade Federal da Bahia  
Instituto de Computação

Programa de Pós-Graduação em Ciência da Computação

**BUG ANALYSIS IN JUPYTER NOTEBOOK  
PROJECTS: AN EMPIRICAL STUDY**

Taijara Loiola de Santana

DISSERTAÇÃO DE MESTRADO

Salvador  
March 01st, 2024



TAIJARA LOIOLA DE SANTANA

**BUG ANALYSIS IN JUPYTER NOTEBOOK PROJECTS: AN  
EMPIRICAL STUDY**

Esta Dissertação de Mestrado foi apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Advisor: Eduardo Santana de Almeida  
Co-Advisor: Paulo Anselmo da Mota Silveira Neto

Salvador  
March 01st, 2024

Ficha catalográfica elaborada pela Biblioteca Universitária de Ciências e Tecnologias Prof. Omar Catunda, SIBI - UFBA.

**C871** Santana, Taijara

Bug Analysis in Jupyter Notebook Projects: An Empirical Study/ Taijara Loiola de Santana. – Salvador, 2024.

179 f.

Orientadora: Prof. Dr Eduardo Santana de Almeida

Dissertação (Mestrado) – Universidade Federal da Bahia. Programa de Pós-Graduação em Ciência da Computação, 2024.

1. Jupyter Notebooks. 2. Bugs. 3. Interviews. 4. Mining Software Repositories (MSR). 5. Stack Overflow. 6. Empirical Study. 7. Computational Notebooks. 8. Data-driven Development. I. ALMEIDA, Eduardo. II. ANSELMO, Paulo. III. Universidade Federal da Bahia. Instituto de Computação. IV. Título.

**CDU:616-083:173.4**

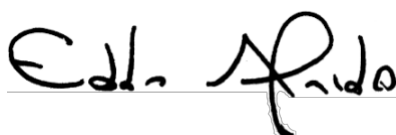
# Termo de Aprovação

**Taijara Loiola de Santana**

## **Bug Analysis in Jupyter Notebook Projects: An Empirical Study**

Esta Dissertação foi julgada adequada à obtenção do título de Mestre em Ciência da Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação da UFBA.

Salvador, 01 de março de 2024



---

Prof. Dr. Eduardo Santana de Almeida  
(Orientador -UFBA)



---

Prof. Dr. Rodrigo Rocha Gomes e Souza  
(UFBA)



---

Prof.<sup>a</sup>. Dr.<sup>a</sup>. Tayana Uchôa Conte  
(UFAM)

*I dedicate this dissertation to God, to my cherished parents, whose love and sacrifices have been my bedrock. To my beloved wife, whose unwavering support and encouragement have fueled my journey. To my brothers and friends, whose constant presence and belief in me have been a source of strength and inspiration.*

## ACKNOWLEDGEMENTS

To my beloved family, whose unwavering support has been my rock throughout this journey. To my parents and brothers, your love, guidance, and encouragement have been the driving force behind my achievements.

Special gratitude to my loving wife, Patricia Oliveira, who has illuminated my path with patience, affection, and unwavering love. Your presence has been my greatest source of strength. I am also deeply grateful for the blessing of our newborn son, Benício, whose arrival fills our lives with even more joy and purpose.

I extend my heartfelt thanks to the esteemed professors at the Federal University of Bahia, particularly to my advisor, Dr. Eduardo Almeida, and my co-advisor, Dr. Paulo Anselmo. Your belief in me and guidance have been invaluable as I navigated this academic endeavor.

I am deeply grateful to all the practitioners, reviewers, and interviewees who contributed to this dissertation, as well as the RiSE research group/LES for their countless direct and indirect contributions through suggestions, criticisms, discussions, and especially friendship.

As Sir Isaac Newton said, "If I have seen further, it is by standing on the shoulders of giants.", thank you all.



"The mind that opens to a new idea never returns to its  
original size."

—ALBERT EINSTEIN

## RESUMO

Uma das novas tecnologias que vêm impulsionando a ciência de dados são os Notebooks Computacionais, que permitem aos usuários construir códigos orientados a dados, enfatizando a análise realizada e os dados obtidos. Apesar de os Notebooks computacionais ganharem visibilidade, problemas e soluções já discutidos e estudados pela engenharia de software precisam ser abordados, impactando a qualidade do software desenvolvido e, conseqüentemente, a análise de dados. Isso também pode levar à disseminação de práticas de programação inadequadas. Notebooks computacionais, como o Jupyter, têm sido amplamente adotados por cientistas de dados para escrever código para análise e visualização de dados. Apesar de sua crescente adoção e popularidade, poucos estudos foram encontrados para compreender os desafios de desenvolvimento do Jupyter do ponto de vista dos praticantes. Este estudo apresenta uma investigação sistemática de bugs e desafios que os praticantes do Jupyter enfrentam por meio de uma investigação empírica em larga escala. Mineramos 14.740 commits de 105 projetos de código aberto do GitHub com código de Notebooks Jupyter. Em seguida, analisamos 30.416 postagens no Stack Overflow, que nos deram insights sobre bugs que os praticantes enfrentam ao desenvolver projetos de Notebooks Jupyter. Conduzimos dezenove entrevistas com cientistas de dados para descobrir mais detalhes sobre os bugs do Jupyter e obter insights sobre os desafios dos desenvolvedores do Jupyter e por fim, para validar todas as informações obtidas, realizamos um survey com diversos cientistas de dados e uma análise com regras de associação utilizando o algoritmo Apriori. Propomos uma taxonomia de bugs para projetos Jupyter com base em nossos resultados. Também destacamos categorias de bugs, suas causas raiz e os desafios que os praticantes do Jupyter enfrentam.

**Palavras-chave:** Notebooks Jupyter, Bugs, Entrevistas, Repositórios de Software de Mineração (MSR), Stack Overflow, Estudo Empírico, Notebooks Computacionais, Desenvolvimento Orientado a Dados

## ABSTRACT

One of the new technologies driving data science projects is Computational Notebooks, which allow users to build data-oriented codes, emphasizing the data collected and the analysis performed. Although Computational Notebooks have gained visibility, some problems and solutions already discussed and studied by the software engineering community must be addressed, impacting the quality of the developed software and, consequently, data analysis. In addition, neglecting these aspects can lead to the spread of bad programming practices. 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 are available to understand Jupyter development challenges from the practitioners' point of view. This dissertation systematically studies 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 Stack Overflow posts, which gave us insights into bugs that practitioners face when developing Jupyter Notebook projects. We conducted nineteen interviews with data scientists to uncover more details about Jupyter bugs and to gain insight into Jupyter developers' challenges and finally, to validate all the information obtained, we carried out a survey with several data scientists and an analysis with association rules using the Apriori algorithm. We propose a bug taxonomy for Jupyter projects based on our results. We also highlight bug categories, their root causes, and Jupyter practitioners' challenges.

**Keywords:** Jupyter Notebooks, Bugs, Interviews, Mining Software Repositories (MSR), Stack Overflow, Empirical Study, Computational Notebooks, Data-driven Development

# CONTENTS

<b>Chapter 1—Introduction</b>	1
1.1 Motivation . . . . .	1
1.2 Problem Statement . . . . .	2
1.3 Goal . . . . .	2
1.4 Statement of the Contributions . . . . .	3
1.5 Out of Scope . . . . .	3
1.6 Document Structure . . . . .	4
<b>Chapter 2—Background</b>	5
2.1 Computational Narratives . . . . .	5
2.2 Computational Notebooks . . . . .	5
2.3 Jupyter Notebooks . . . . .	7
2.4 Jupyter Problems . . . . .	8
2.5 Software Development IDEs Supporting Notebooks . . . . .	8
2.6 Chapter Summary . . . . .	10
<b>Chapter 3—Related Work</b>	11
3.1 Jupyter Notebooks - Extensions . . . . .	11
3.2 Jupyter Notebooks - How Data Scientists Use it . . . . .	12
3.3 Jupyter Notebook - Notebook Quality . . . . .	12
3.4 Empirical Studies on Bugs . . . . .	13
3.5 Studies on Bugs With Taxonomies . . . . .	14
3.6 Chapter Summary . . . . .	14
<b>Chapter 4—Methodology</b>	17
4.1 Research Design . . . . .	17
4.2 Repositories and Posts Selection and Mining . . . . .	19
4.3 Classifying and Labeling Bugs . . . . .	20
4.4 Data Scientists Interviews . . . . .	23
4.5 Survey . . . . .	25
4.6 Chapter Summary . . . . .	26
<b>Chapter 5—Results</b>	29
5.1 Types of Bugs in Jupyter Projects (RQ1) . . . . .	29

5.2	Root Causes of Bugs (RQ2)	38
5.3	Impacts of Bugs (RQ3)	45
5.4	Challenges in Jupyter Notebook Projects (RQ4)	49
5.5	Discussion	53
5.6	Lessons Learned	56
5.7	Threats to Validity	57
5.8	Chapter Summary	58
<b>Chapter 6—Conclusions</b>		<b>59</b>
6.1	Summary of Research Contributions	59
6.2	Research Products	60
6.3	Future work	60
6.4	Concluding Remarks	61
<b>Appendix A—List of GitHub Projects</b>		<b>69</b>
<b>Appendix B—List of Other Notebooks</b>		<b>73</b>

## LIST OF FIGURES

2.1	Notebook layout. . . . .	6
2.2	Jupyter notebook interface. . . . .	8
4.1	Research Methodology. . . . .	19
5.1	Taxonomy of Jupyter Notebook bugs. . . . .	29
5.2	Frequency of bug types (a) StackOverflow and (b) Github. . . . .	34
5.3	Average annual growth of bugs extracted from StackOverflow. . . . .	35
5.4	Bug Type Survey Validation. . . . .	38
5.5	Root Cause Survey Validation. (KN) Kernel, (CV) Conversion, (PB) Portability, (ES) Environment and Settings, (CN) Connection, (PC) Processing, (CD) Cell Defect, (IP) Implementation. . . . .	45
5.6	Bug reports with accepted answers (StackOverflow). . . . .	51
5.7	Average accepted response time (in days). . . . .	51

## LIST OF TABLES

4.1	Quantity and Percentage considering different StackOverflow Metrics. . .	22
4.2	Interview participants background. . . . .	25
5.1	Python Exceptions per Type of Bugs. . . . .	36
5.2	Apriori Analysis Python Exceptions per Type of Bugs. . . . .	37
5.3	Frequency of Bug Type vs Root Cause . . . . .	42
5.4	Apriori Analysis Bug Type and Root Cause. . . . .	44
5.5	Frequency of Impact vs Bug type . . . . .	47
5.6	Apriori Analysis Bug Type and Impact. . . . .	49
5.7	Jupyter History in StackOverflow - General Posts related to Jupyter vs Posts related to Bugs in Jupyter . . . . .	53
5.8	Features mentioned by respondents. . . . .	55
A.1	List of GitHub projects . . . . .	71
B.1	Other Notebooks List . . . . .	73

# 1

## INTRODUCTION

Data science and data analysis are emerging areas with professionals from different backgrounds, such as mathematics, statistics, and computer science. They combine this diverse knowledge with domain knowledge to obtain strategic insights through exploration, quantification, qualification, and data prediction (DHAR, 2013; TAO et al., 2020; CAO, 2017).

Over the years, special attention has been given to the data analysis and processing areas due to the large amount of data available (BEGEL; ZIMMERMANN, 2014). With this growth in data science and analysis, Jupyter Notebook has become one of the most used tools (PIMENTEL et al., 2019; KOENZEN; ERNST; STOREY, 2020). Jupyter is a computational notebook, a web application that allows data scientists to write text and code in a documentation structure that describes the data analysis process and that is commonly used to refine code, explore unknown data, test hypotheses, and build models (WANG; LI; ZELLER, 2020a; HEAD et al., 2019). Although Jupyter offers a new type of development, different from the traditional model found in traditional IDEs, it can bring benefits and problems to the data analysis development process, such as the ease of exploration and documentation, weighed against the potential for promoting undesirable development practices.

### 1.1 MOTIVATION

The growing use of computer notebooks, especially the Jupyter Notebook, presented benefits to the data scientist, but several problems also accompanied it. Wang et al. (WANG; LI; ZELLER, 2020a) analyzed 1982 Jupyter notebooks and found their code incompatible with Python development standards, having unused variables and obsolete functions. Pimentel et al. (PIMENTEL et al., 2019) analyzed 1,159,166 notebooks and found that 24.11% of them could be reproduced without errors. Some studies focused on highlighting the discomfort that the user has when using Jupyter (HEAD et al., 2019; CHATTOPADHYAY et al., 2020), Others identified specific and important problems such as *name-value inconsistency* found in numerous notebooks (PATRA; PRADEL, 2021) or the lack of formal declaration of the notebook's *dependencies*, a gap present in about 94%



of the notebooks studied, making it difficult or impossible to reproduce them (WANG; LI; ZELLER, 2021).

In addition, Jupyter's popularity grows with the popularity of data science (Glassdoor ranks data science as the #3 job in America for 2022<sup>1</sup>), and these errors can have serious consequences. For example, because of these problems, there is a tendency for data scientists and analysts to see notebooks as an *ad-hoc, experimental, and throw-away code* (KANDEL et al., 2012) tool, besides describing them as messy (KERY et al., 2018; RULE; TABARD; HOLLAN, 2018a) and containing *ugly code* and *dirty tricks* in need of *cleaning* and *polishing* (RULE; TABARD; HOLLAN, 2018a). Understanding these issues more deeply can help the data scientist community identify points of improvement and care when using the tool.

## 1.2 PROBLEM STATEMENT

Previous studies by the Software Engineering community investigated bugs in different domains (THUNG et al., 2012; ZHANG et al., 2018; ISLAM et al., 2019; GARCIA et al., 2020; RAHMAN et al., 2020; MAKHSHARI; MESBAH, 2021; WANG et al., 2021). These studies demonstrate how important historical bug analysis is for bug reduction (THUNG et al., 2012), and provide important information for improving Jupyter and similar tools. The community has also stressed "the strong need to analyze the quality of the notebooks" (WANG; LI; ZELLER, 2020a; CHATTOPADHYAY et al., 2020; WANG; LI; ZELLER, 2021) to improve the quality and reliability of the code. However, bugs and their characteristics in Jupyter projects have not yet been studied.

As evidenced in previous research (WANG; LI; ZELLER, 2020a; PIMENTEL et al., 2019; HEAD et al., 2019; CHATTOPADHYAY et al., 2020; PATRA; PRADEL, 2021; WANG; LI; ZELLER, 2021), Jupyter notebook has a series of limitations, problems and challenges faced by its users, especially related to its layout, and the bugs that may be appearing in this scenario can cause serious damage to a data science project, (like what happened in the UK when nearly 16,000 COVID-19 cases were lost from exceeding the spreadsheet data limit <sup>2</sup>). Thus, analyzing and improving Jupyter Notebook projects have a potentially relevant impact.

## 1.3 GOAL

Motivated by the problems and challenges presented in the previous sections, the objective of this work can be stated as follows:

*Investigate the bugs in Jupyter projects, including their characteristics, such as root causes, the impact of their occurrence, the challenges data scientists face, and how it*

---

<sup>1</sup>50 Best Jobs in America for 2022 - [https://www.glassdoor.com/List/Best-Jobs-in-America-LST\\_-KQ0,20.htm](https://www.glassdoor.com/List/Best-Jobs-in-America-LST_-KQ0,20.htm)

<sup>2</sup>Thousands 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>

*affects them.*

We follow four steps to understand bugs in Jupyter projects better. First, we extracted commits from open source GitHub repositories, referring to the Jupyter Notebooks code (Appendix A.1). We then did a similar process with the Stack Overflow platform, using the bug-related posts on Jupyter and their responses. This way, we had a distinct raw dataset with information about the errors data scientists face when developing projects on the Jupiter notebook. Next, we carried out recursive processes of sampling, classifying, and categorizing the databases to identify our bug taxonomy. We conducted semi-structured interviews with data scientists to validate the findings identified in the previous steps and understand how these bugs affect the scientists' daily lives, providing findings on the challenges data scientists face at Jupyter. Finally, to validate all the information obtained, we carried out a survey with several data scientists and an analysis with association rules using the Apriori algorithm (AGRAWAL; SRIKANT, 1994).

## 1.4 STATEMENT OF THE CONTRIBUTIONS

As a result of the work presented in this dissertation, a list of the main contributions may be enumerated:

- 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 Stack Overflow, coupled with insights garnered through interviews, we provide a set of recommendations tailored to benefit both researchers and practitioners within the field.
- 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 Stack Overflow, and all interview data (prompt, summary of professional and demographic information from Participants, and codebook). Our artifacts can be found at the accompanying website<sup>3</sup>.

## 1.5 OUT OF SCOPE

As Jupyter notebooks are part of a broader context, a set of related aspects will be outside their scope. Therefore, the following questions are not directly addressed by this work:

- Other computing notebooks. Jupyter is the pioneer and one of the best-known computing notebooks in the community. There are other notebooks that may have similar problems and challenges or provide solutions to some of the problems highlighted in this work; however, we only focused on Jupyter in our study.

---

<sup>3</sup><https://github.com/bugsjupyterempiricalstudy/BugJupyterPaper>

- Minor bugs. This work took random samples from a raw database for analysis and focused on classifying, labeling, and characterizing the most common bugs. Minor bugs may have been left out of the analysis.
- Characteristics of bugs. This work characterizes bugs by type, root cause, and impact; however, it was not the focus of this study to deepen this analysis with details of how each type of bug can be avoided, how each type of bug behaves, or what damage is caused by each type.

## **1.6 DOCUMENT STRUCTURE**

The remainder of this document is organized as follows: Chapter 2 summarizes the most important topics to understand Computational Narrative, Computational Notebooks, and Bugs. Chapter 3 discusses related work. Chapter 4 presents the research questions and the proposed methodology. Chapter 5 shows the results found in this research. Finally, the chapter 6 summarizes and discusses the main findings.

# 2

## BACKGROUND

This chapter describes the fundamental concepts relevant to this work: Computational Narratives, Computational Notebooks, Jupyter Notebooks, and other Notebooks, such as Google Collaboratory and RMarkdown. Section 2.1 defines computational narratives and their importance. Section 2.2 correlates the importance of computational narratives with computational notebooks and definitions. Section 2.3 presents the Jupyter notebook and section 2.4 some of its issues. Finally, section 2.5 describes the IDE with notebooks and brings a non-exhaustive list of the main computing notebooks used, such as Google Collaboratory and RMarkdown. The challenges described in this chapter highlight the importance of computational notebooks for data analysis and data scientists. In particular, it motivates our work to understand more deeply the problems related to computing notebooks, what can arise from this scenario related to bugs, and how this can impact the daily lives of computer scientists.

### 2.1 COMPUTATIONAL NARRATIVES

While computers consume, produce and process data, humans need context to process information. Thus, a narrative needs to order and connect events and facts in a chronology that makes sense to the reader, contextualizing and facilitating the understanding of the human being (RULE; TABARD; HOLLAN, 2018a).

Exploring data and extracting insights is made up of numerous blocks of code reviewed, duplicated, and reordered countless times. It is challenging to organize, document, review, and replicate the analysis performed, making each code unique and situational.

The computational narrative can be a possible solution for promoting the construction of development narratives in an orderly and documented way in a single story.

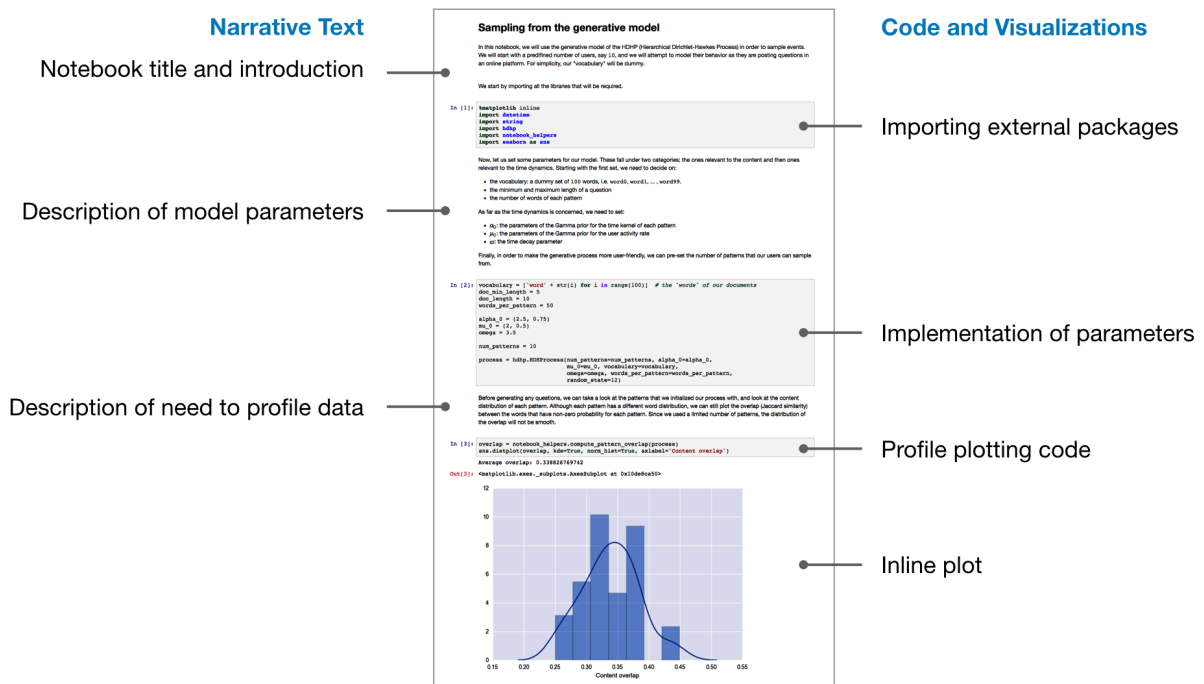
### 2.2 COMPUTATIONAL NOTEBOOKS

To develop in a narrative way, it is necessary to document the code and the analysis process. How each step was performed, the logic of the general analysis, analysis changes, and their intermediate and final results. In addition, for this narrative to be helpful to

the data scientist and community, it needs to meet three fundamental aspects<sup>1</sup>: First, allow the analysis to be developed and detailed in different ways, texts, images, graphics, codes, etc., to compose a story that the different public can read. Secondly, data analysis must be reproducible for future corrections, validations, and improvements. Moreover, finally, it must be possible to work collaboratively.

In this context, computer notebooks were designed to support the construction of computational narratives, combining text, graphics, images, code, and output into a single document.

The structure of a notebook consists of a single page, composed of blocks divided into types, Markdown blocks, which can contain texts, images, and links; code blocks and output blocks, which are the outputs resulting from the execution of the code blocks (see figure 2.1).



**Figure 2.1** Notebook layout.  
(RULE; TABARD; HOLLAN, 2018b)

The blocks are sequentially organized and are executed and re-executed in any order. It enables greater freedom of exploration. However, it also makes the development messy and confusing (HEAD et al., 2019; WANG et al., 2020). Rule et al. suggest ten rules on how to write a good notebook, such as modularizing code, registering dependencies,

<sup>1</sup><https://blog.jupyter.org/project-jupyter-computational-narratives-as-the-engine-of-collaborative-data-science-2b5fb94c3c58>

using version control, and planning notebooks thinking they will be read, executed, and explored by others (RULE et al., 2019).

## 2.3 JUPYTER NOTEBOOKS

Jupyter is the computational notebook chosen by data scientists and has been gaining popularity since 2016 (PERKEL, 2018). It is a free tool that combines code, text, computational outputs, and multimedia resources in a single document. It results in a computational narrative from data analysis mixed with models to test hypotheses and conjectures.

The Jupyter notebook format is ".ipynb" (JSON schema <sup>2</sup>), which can be ported and run on any platform with the same kernel. There are some kernels in Jupyter Project <sup>3</sup> that supports other languages besides Julia, Python, and R.

In addition to the classic web solution, there is also the Jupyter Lab, which was designed to be an extensible environment, which adds the Jupyter notebook experience to a user interface.

The Jupyter Lab mix text editors, a more flexible view of the development and mainly the possibility to use and develop Jupyter extensions<sup>4</sup>.

The notebook interface works as follows <sup>5</sup>, the notebook runs the code, stores the output, and text blocks into a single editable document. When this document is saved, it is sent from the browser to the server, which maintains it in the ".ipynb" format. The server is responsible for saving and loading notebooks, making it possible to edit the document without linking a kernel (see figure 2.2). In addition, its architecture allows the Jupyter notebook also run outside the machine where the kernel is.

The Jupyter project developed this flexible interface of the notebook architecture as a solution to the following problem:

*"the collaborative creation of reproducible computational narratives that can be used across a wide range of audiences and contexts."* <sup>6</sup>.

---

<sup>2</sup><https://github.com/jupyter/nbformat/blob/main/nbformat/v4/nbformat.v4.schema.json>

<sup>3</sup><https://jupyter.org/>

<sup>4</sup><https://jupyterlab.readthes.io/en/stable/index.html>

<sup>5</sup><https://docs.jupyter.org/en/latest/projects/architecture/content-architecture.html>

<sup>6</sup><https://blog.jupyter.org/project-jupyter-computational-narratives-as-the-engine-of-collaborative-data-science-2b5fb94c3c58>

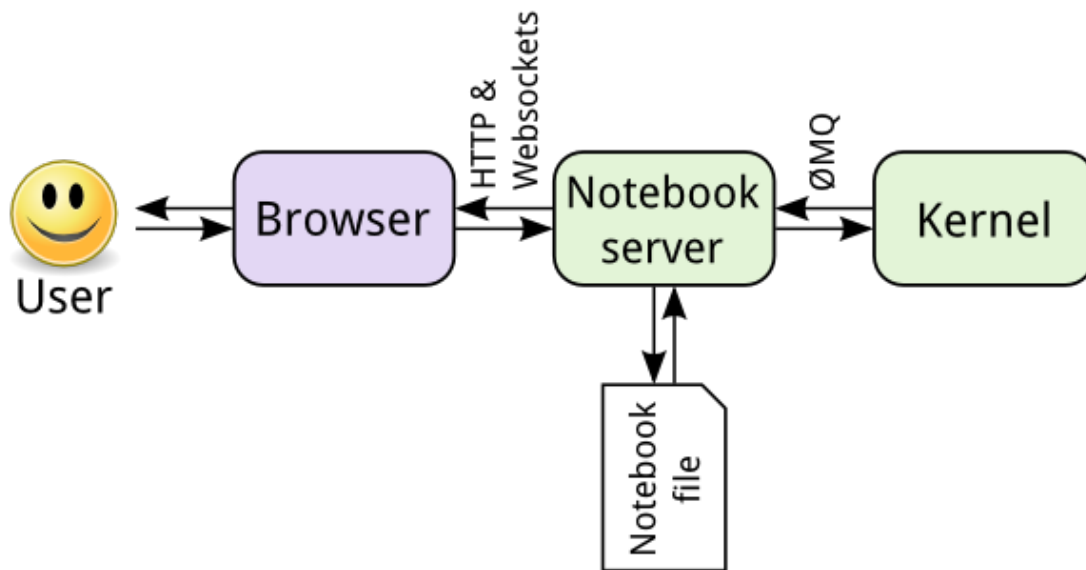


Figure 2.2 Jupyter notebook interface.

## 2.4 JUPYTER PROBLEMS

The Jupyter Notebook has emerged as a great solution to optimize data analysis and document the process in an interactive way. It simplifies its use. However, it demands from users a greater discipline to guarantee reuse and reproducibility.

There are several researches dedicated to understanding and suggesting improvement points or tools (extensions) that can help in the process, so that notebook development does not allow errors originating from cell executions out of order, bad readability, bad modularity, lack of debugging, bad programming practices, among others (KOENZEN; ERNST; STOREY, 2020; KERY et al., 2020; HEAD et al., 2019; PIMENTEL et al., 2019; WANG; LI; ZELLER, 2020a; WANG et al., 2020; CHATTOPADHYAY et al., 2020; RULE; TABARD; HOLLAN, 2018a).

The process of exploring the data as mentioned before is exhaustive, repetitive and full of details and a small problem can impact a whole bad analysis or even wrong results. Which can generate dissatisfaction of data scientists <sup>7</sup>.

## 2.5 SOFTWARE DEVELOPMENT IDES SUPPORTING NOTEBOOKS

The Jupyter Notebook's popularity makes it an important tool for data science, especially for beginners, due to its layout simplicity and user-friendliness. However, if the analysis

<sup>7</sup><https://conferences.oreilly.com/jupyter/jup-ny/public/schedule/detail/68282.html>

requires more advanced functionalities, the Jupyter Notebook could not be helpful due to its limitations.

In the annual survey carried out by the Stack Overflow<sup>8</sup> platform, Jupyter appeared in 2016 as one of the most used tools by developers.

Over the years, its number of users has grown, and in 2019, the survey itself noted its results:

*"Visual Studio Code is a dominant player among developer environment tools this year. There are differences in tool choices by developer type and role, but Visual Studio Code was a top choice across the board. Developers who write code for mobile apps are more likely to choose Android Studio and Xcode. The popular choice for DevOps and SREs is Vim, and data scientists are more likely to work in IPython/Jupyter, PyCharm, and RStudio."*<sup>9</sup>

In the 2021 survey<sup>10</sup>, the Jupyter users itself start to migrate to another tools:

*"We see IPython/Jupyter users want to work in VS Code. This is likely due to VS Codes adding a Notebook API to their IDE."*<sup>11</sup>

The limitations of Jupyter Notebook can generate dissatisfaction in its users, either because of its difficulties or even bugs resulting from them. However, despite these limitations, Jupyter Notebook is a solution that adds value to the exploratory data analysis process, as highlighted earlier by its growing popularity.

As a solution to these limitations, many IDE's have added to their platforms the possibility of creating and importing the Jupyter file (.ipynb), adding a set of typical IDE's functionalities to the Jupyter solution without abandoning its popular concept of computational narrative.

The VSCode IDE<sup>12</sup> provides for its user's text editor, with version control, plugins, coding assistant and also allows the use of .py files associated with the notebook.

The Pycharm IDE<sup>13</sup> is another popular and robust editor as VSCode and has provided integration with the notebook.

The DataSpell IDE<sup>14</sup> is dedicated to data science projects. It is similar to a notebook and provides intelligent coding assistance, version control, a debugger, connection to different database tools, and others.

---

<sup>8</sup><https://insights.stackoverflow.com/survey>

<sup>9</sup><https://insights.stackoverflow.com/survey/2019>

<sup>10</sup><https://insights.stackoverflow.com/survey/2021>

<sup>11</sup><https://insights.stackoverflow.com/survey/2021#section-worked-with-vs-want-to-work-with-collaboration-tools>

<sup>12</sup><https://code.visualstudio.com/docs/datascience/jupyter-notebooks>

<sup>13</sup><https://www.jetbrains.com/help/pycharm/jupyter-notebook-support.html>

<sup>14</sup><https://www.jetbrains.com/help/dataspell/jupyter-notebook-support.html>



The Jupyter Lab itself, is a natural evolution of Jupyter Notebook to advanced user needs, providing an improved interface with a file browser, consoles, terminals, text editors, Markdown editors, CSV editors, JSON editors, interactive maps and widgets.

All these Jupyter Lab functionalities, in addition to improving the user experience, allow the build of extensions that expand the scope of features of the notebook development environment<sup>15</sup>

In addition to Jupyter Notebook and the IDE's (with notebooks) mentioned above, there are other platforms with their notebooks or adaptations (see Appendix B.1).

## 2.6 CHAPTER SUMMARY

This chapter covers the fundamental concepts relevant to this work, including Computational Narratives, Computational Notebooks, Jupyter Notebooks, and other notebooks such as Google Collaboratory and RMarkdown. It begins by defining computational narratives and their importance, followed by the correlation with computational notebooks. It describes the Jupyter Notebook, its issues and challenges, and concludes with a discussion of IDEs that support notebooks, highlighting the significance of these tools for data analysis and data scientists.

Computational narratives are essential for organizing and contextualizing data analysis, allowing humans to understand complex information. Computational notebooks, such as Jupyter, combine text, code, and visualizations in a single document, facilitating reproducibility and collaboration. However, they present challenges, including execution order errors and poor programming practices. IDEs like VSCode, PyCharm, and DataSpell are integrating notebook functionalities, offering advanced features to enhance user experience and effectiveness in data analysis.

In the next chapter, related work will be reviewed, grouped into five key areas: extensions for Jupyter notebooks, usage patterns, quality assessment, bug studies, and bug taxonomies.

---

<sup>15</sup><https://ipython-books.github.io/36-introducing-jupyterlab>

# 3

## RELATED WORK

In this section, we discuss the main work related to our study, grouping it into five conceptual sections. Session 3.1 presents work that analyzes the main pain points of Jupyter users and brings the development of an extension as a solution for them. Session 3.2 highlights particularities in the use of computing notebooks. Session 3.3 discusses work on analyzing the quality of notebooks already developed by the community. Session 3.4 lists more recent works that study the occurrence of bugs in open source software. Finally, Session 3.5 discusses bug studies that have similarities to our study.

### 3.1 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 (JUPYTER, 2015). Some studies have been proposing different ways to improve these aspects.

Rule et al. (RULE et al., 2018) 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. It shows how the analysis process in a notebook can be confusing and hard to understand, especially in large documents. Head et al. (HEAD et al., 2019) 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. (KERY et al., 2020) 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 (WANG et al., 2020), and the second one to retrieve and install the notebook's experimental dependencies (WANG; LI; ZELLER, 2021).

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.

### **3.2 JUPYTER NOTEBOOKS - HOW DATA SCIENTISTS USE IT**

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. (KOENZEN; ERNST; STOREY, 2020) studied how these duplication 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. (WANG et al., 2019) 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.

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 problem with the tool.

### **3.3 JUPYTER NOTEBOOK - NOTEBOOK QUALITY**

Chattopadhyay et al. conducted a study (CHATTOPADHYAY et al., 2020) that involved observing five 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. (RULE; TABARD; HOLLAN, 2018b) 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. (PIMENTEL et

al., 2019) 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 (PIMENTEL et al., 2021). 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. (WANG; LI; ZELLER, 2020b) developed a preliminary study where the results revealed a high amount of bad coding practices in Jupyter notebooks. However, unlike the previous study, Patra et al. (PATRA; PRADEL, 2021) 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 categorize and quantify types of bugs and root causes in the domain of Jupyter notebooks.

All 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. On the other hand, some of these studies cite or highlight some bugs, but they need to go into depth, as we are doing here in this work.

### 3.4 EMPIRICAL STUDIES ON BUGS

Some related work are not directly related to data science, such as: Zhang et al. (ZHANG et al., 2018) 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. (ISLAM et al., 2019) extended the search for other popular deep learning libraries, Caffe, Keras, Tensorflow, Theano, and Torch. In addition, Thung et al. (THUNG et al., 2012) analyzed bugs in machine learning systems, but their research used the issues reported on 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 (WANG et al., 2021), bugs in IoT systems (MAKHSHARI; MESBAH, 2021), bugs in autonomous vehicles (GARCIA et al., 2020) and bugs involving Infrastructure as Code Scripts (RAHMAN et al., 2020).

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.

### 3.5 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 (WANG et al., 2021), bugs in IoT systems (MAKHSHARI; MESBAH, 2021), bugs in autonomous vehicles (GARCIA et al., 2020) and bugs involving Infrastructure as Code Scripts (RAHMAN et al., 2020).

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 (such as 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.

It is also important to highlight that to create a new taxonomy for bugs empirically, studies such as that of autonomous vehicles (GARCIA et al., 2020), observed taxonomies from previous studies and adapted them to their 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 "five whys technique" (MAKHSHARI; MESBAH, 2021) or "Cohen's Kappa" (RAHMAN et al., 2020), 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 the questions, analyzes 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.

### 3.6 CHAPTER SUMMARY

This chapter reviews related work, grouping them into five key areas: extensions for Jupyter notebooks, usage patterns, quality assessment, bug studies, and bug taxonomies. Several studies aimed to enhance Jupyter notebooks by developing extensions to improve navigation, code versioning, and reproducibility. Other research examined how data scientists use notebooks, revealing practices such as code duplication and real-time collaboration challenges. Quality assessments highlighted issues with notebook readability, replication, and reproducibility, while studies on coding practices identified common problems and inconsistencies.

Our study focuses on identifying and classifying bugs in Jupyter notebooks to provide

a systematic overview of the challenges faced by developers. Unlike previous research that introduced new features or analyzed specific notebook uses, our goal is to map and quantify bugs, offering insights into their frequency, types, root causes, and impact on daily use. By characterizing these bugs, we aim to inform future improvements in Jupyter notebooks, enhancing their utility for the data science community.

The next chapter details a comprehensive methodology for characterizing Jupyter Notebook bugs, including GitHub repository mining, StackOverflow post analysis, and interviews with data scientists, addressing key research questions, data collection and analysis, and validation of results.

# 4

## METHODOLOGY

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.

### 4.1 RESEARCH DESIGN

To fulfill the purpose highlight, our study aims to answer the following research questions (RQs):

- 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 Notebooks projects. It is an important step for researchers and practitioners who use this tool in their daily lives.
- 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.

In order to answer these questions, the following steps were performed. Firstly, (i) a

*GitHub*<sup>1</sup> *Repository Mining* analysis was performed to characterize bugs in the context of Jupyter Notebooks<sup>2</sup> projects. In this analysis, only commits related to bug fixing were considered by inspecting the commit message (MAKHSHARI; MESBAH, 2021; ISLAM et al., 2019; WANG et al., 2021; GARCIA et al., 2020). Next, (ii) *the StackOverflow*<sup>3</sup> *posts* analysis was performed to characterize data science difficulties/issues/questions when using Jupyter Notebooks. Next, (iii) *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 (SALDANA, 2015). Next, (iv) *semi-structured interviews* were conducted with data scientists to obtain and validate insights on the main issues when developing using Jupyter Notebooks projects. Finally, (v) *a Survey* was carried out with 91 developers to validate out findings and proposed taxonomy. Figure 4.1 shows the overall research methodology used in our study. All the quantitative and qualitative data is available online at the accompanying website<sup>4</sup>.

---

<sup>1</sup><https://github.com>

<sup>2</sup><https://jupyter.org>

<sup>3</sup><https://en.stackoverflow.com>

<sup>4</sup><https://github.com/bugsjupyterempiricalstudy/BugJupyterPaper>



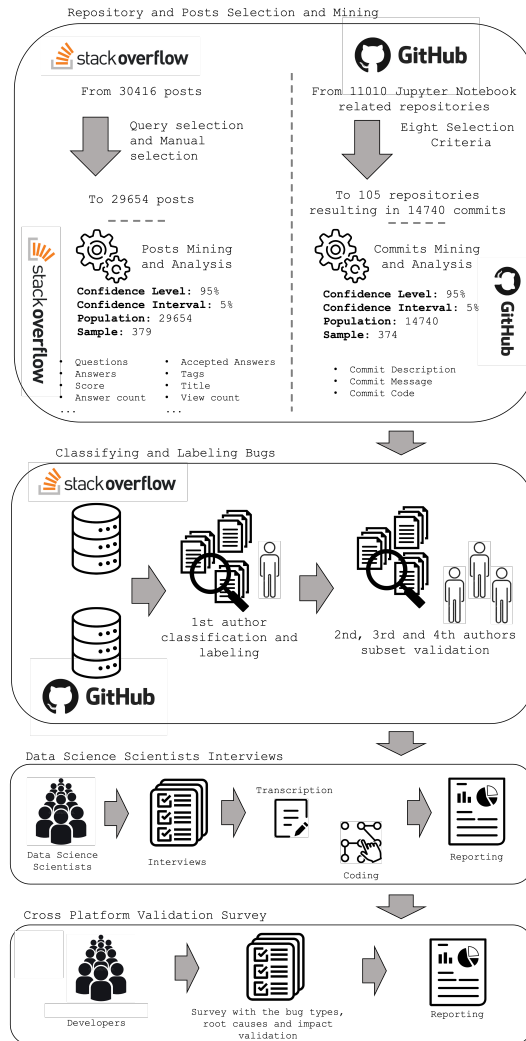


Figure 4.1 Research Methodology.

## 4.2 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. Following, the projects were sorted by star. Next, to filter out the most relevant and active projects from the 11010 projects collected, some inclusion and exclusion criteria were applied as recommended by (MUNAIAH et al., 2016).

- **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*. It resulted in 11010 projects;

- Projects without description ("None") were removed. It resulted in 9866 projects;
- Projects written in Chinese, Japanese, and any language other than English were removed. It resulted in 8612 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*. It resulted in 6910 projects;
- Projects without any update(commit) in 2020 were removed. It resulted in 6885 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. It resulted in 3714 projects;
- The repository must have at least ten contributors in 2020. This criterion was used to eliminate irrelevant repositories, c.f., (AGRAWAL et al., 2018), (KRISHNA et al., 2018), (RAHMAN et al., 2018). It 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', 'fault', 'faults', 'flaws', 'flaw', 'failure', 'correction', 'corrections'*. This criterion was used to filter out commits not related to bug fixing. (MAKHSHARI; MESBAH, 2021; GARCIA et al., 2020). It resulted in 105 projects;

After filtering, we selected the top 105 Jupyter repositories, resulting in 14740 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 <sup>5</sup> resulting in 30416 posts. Incomplete and inaccessible posts were removed bringing out 29654 posts in our StackOverflow raw dataset. Finally, the .csv generated was analyzed.

### 4.3 CLASSIFYING AND LABELING BUGS

We created a spreadsheet with all GitHub commits and StackOverflow posts filtered, 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

---

<sup>5</sup><https://data.stackexchange.com/stackoverflow/query/1541588/jupyternotebookbugs>

and posts. The coding process applied used a set of first-cycle, and second-cycle coding methods for data analysis (SALDANA, 2015). The first methods are those processes during the initial coding of data. The second methods, if needed, are 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. In order 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 (MAKHSHARI; MESBAH, 2021; RAHMAN et al., 2020; ISLAM et al., 2019; THUNG et al., 2012). 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 (MAKHSHARI; MESBAH, 2021; YANG et al., 2021; GARCIA et al., 2020; ZHANG et al., 2018). We applied the Root Cause Analysis (MAKHSHARI; MESBAH, 2021) using the five whys technique (SERRAT, 2017). Finally, regarding impact, we analyzed major effects of bugs by reading the commit message, pull request messages and the associated issues. In the StackOverflow, the question description was important to understand the impact (GARCIA et al., 2020; ISLAM et al., 2019; THUNG et al., 2012).

Type of Bug	Occurrences		Voting Score		Views		Answers		Comments		Favorites	
	Qtd	%	Qtd	%	Qtd	%	Qtd	%	Qtd	%	Qtd	%
Environments and Settings	1118	43.2%	2077	49.9%	3563941	54.3%	1374	50.8%	1822	42.1%	410	48.6%
Implementation	569	22.0%	333	8.0%	799413	12.2%	516	19.1%	1086	25.1%	57	6.8%
Kernel	278	10.8%	406	9.8%	625782	9.5%	242	8.9%	532	12.3%	81	9.6%
Conversion	172	6.7%	401	9.6%	424601	6.5%	159	5.9%	184	4.3%	68	8.1%
Connection	160	6.2%	317	7.6%	416104	6.3%	172	6.4%	238	5.5%	66	7.8%
Processing	126	4.9%	333	8.0%	573660	8.7%	124	4.6%	249	5.8%	88	10.4%
Cell Defect	93	3.6%	174	4.2%	91801	1.4%	73	2.7%	135	3.1%	37	4.4%
Portability	69	2.7%	119	2.9%	65904	1.0%	46	1.7%	82	1.9%	36	4.3%

**Table 4.1** Quantity and Percentage considering different StackOverflow Metrics.

Table 4.1 shows different metrics extracted from StackOverflow database. The metrics are the following described: 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; *Favorites* this is what we used to refer as 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. It 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 14740 commits and 30416 posts were collected, in this stage of classification and labeling, 4 experts participated. the first and second specialist applied the coding process using a set of first-cycle, and second-cycle coding methods (SALDANA, 2015) to identify the bug types. In addition, both specialist used the Root Cause Analysis (MAKHSHARI; MESBAH, 2021) using the five whys technique (SERRAT, 2017) 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 (FUSCH; NESS, 2015). This saturation was achieved when analyzing 855 of 14740 commits, giving us a margin of error of 3% at 95% confidence level. In addition, analyzing 2585 of 30416 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 2585 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 (14914 posts) or only questions related to Jupyter Notebook usage (1950 posts not related to Jupyter). Next, three specialist (2nd, 3rd, and 4th) independently classified 145 commits randomly, and 137 posts were selected to validate the first author classification. When a conflict happens, all specialist vote to solve the conflict. We measured the inter-rater agreement among the specialist using Cohen’s Kappa coefficient (TABATA et al., 2013). A training session was performed among the specialist 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 (LANDIS; KOCH, 1977), which is ‘substantial agreement’.

#### 4.4 DATA SCIENTISTS INTERVIEWS

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, we conducted semi-structured interviews with data scientists. We contacted by email profes-

sionals from companies and researchers who work with data science by inviting them to participate in our interview. Once we received the acceptance reply, we started the interviews. The list of interview questions is available at the website <sup>6</sup>

**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 eighteen open questions. The participants were informed 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 2nd and 3rd specialist also support the interview improvement, solving questions 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 <sup>7</sup>. All the interviews were conducted remotely, and we recorded the audio to further analysis with the participants' consent. The interviews took about 43 minutes on average. We transcribed the recorded interviews using QDA Miner<sup>8</sup>.

**Participants.** After conducting a pilot interview with one data scientist (not included in the study) as a pretest (SEIDMAN, 2006), nineteen data scientists were interviewed. All of them have at least one year of data science experience from several companies and domains as seen in Table 4.2. The data scientists came from 12 different companies, working in domains such as mobile games, finance, car, petrochemical, mining, etc. 40% of participants hold a Ph.D., 25% hold a master's degree, 25% hold a bachelor's degree, and 10% conducted post-doctoral studies.

**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 <sup>9</sup> 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 (SALDANA, 2015). 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 on clustered into categories. Analyzing our data, we could define categories to understand the answers from interview participants. Next, the specialist

---

<sup>6</sup><https://github.com/bugsjupyterempiricalstudy/BugJupyterPaper>

<sup>7</sup><https://github.com/bugsjupyterempiricalstudy/BugJupyterPaper>

<sup>8</sup><https://provalisresearch.com/products/qualitative-data-analysis-software/freeware/>

<sup>9</sup><https://otranscribe.com/>

<b>Id</b>	<b>Role</b>	<b>Company Area</b>	<b>Exp. (Years)</b>
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

**Table 4.2** Interview participants background.

resolved the potential conflicts in the labels and categories. It resulted in 52 codes, 7 categories, and 5 challenges.

## 4.5 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-minutes survey designed to validate our findings and proposed bug taxonomy. It was composed of 11 open questions and 14 closed questions. Nine of the fourteen 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 Kitchenham and Pfleeger’s guidelines for personal opinion surveys (KITCHENHAM; PFLEEGER, 2008).

We pilot out a survey with two researchers (both with Ph.D. degrees) with experience in the area to get feedback on the questions and their corresponding answers; difficulties faced to answer 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 inform their email to receive a summary of the results. The survey instrument is available on the website.<sup>10</sup>

**The Respondents** are spread out over fourteen countries around four continents. The top three countries where the respondents come from are Brazil, India, and the United

<sup>10</sup><https://github.com/bugsjupyterempiricalstudy/BugJupyterPaper>

States. The professional experience of these 90 respondents varies from less than a year to eight to ten 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 (SALDAÑA, 2009) 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 (SALDAÑA, 2009). The First cycle methods are the steps applied in the initial coding of data. 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 specialist with previous experience in this type of study. Each author analyzed the answers independently and conduct the coding process. In the end, both specialist met to analyze the differences in coding. After that, we defined an "agreement level" of 0.80 measured using Cohen's kappa (LANDIS; KOCH, 1977). As Cohen Kappa measures agreement between only two evaluators, the evaluation of the first expert was used versus the consensus of the evaluation of the others.

## 4.6 CHAPTER SUMMARY

This chapter details the methodology used to characterize Jupyter Notebook bugs through GitHub repository mining, StackOverflow post analysis, and interviews with data scientists. The study investigates four research questions (RQs): the frequency of bug types (RQ1), their root causes (RQ2), their impacts (RQ3), and the challenges faced by data scientists (RQ4). The methodology encompasses multiple steps: mining GitHub repositories to identify bug-related commits, analyzing StackOverflow posts to understand user issues, and manually labeling and classifying bugs to identify types, causes, and impacts. Semi-structured interviews with data scientists validate these findings, while a survey involving 91 developers further strengthens the results and the proposed taxonomy.

Repositories were selected based on criteria such as language, activity, and number of contributors, resulting in 105 repositories and 14,740 valid commits. StackOverflow posts were filtered, resulting in 29,654 relevant entries. A comprehensive spreadsheet was created to categorize bug types, their root causes, and impacts through iterative classification



and coding techniques. Interviews with 19 data scientists provided qualitative insights, and a survey validated the results with diverse demographic representation. The study achieved high agreement among classifiers and utilized correlation analysis to validate the robustness of the collected data, providing comprehensive insights into Jupyter Notebook bugs and their implications for users.

The next chapter presents a comprehensive analysis of bugs in Jupyter projects, covering types, root causes, and impacts. It proposes an initial taxonomy refined through qualitative and statistical analyses based on GitHub and StackOverflow data. It highlights challenges in code quality, deployment, and the need for software engineering practices to enhance user experience.

# 5

## RESULTS

This Chapter reports the answers to our targeted research questions and findings collected from GitHub, StackOverflow, and interview responses. In addition, a survey was carried out and an analysis using association rules (Apriori Algoritimo) for general validations.

### 5.1 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 5.1 shows the taxonomy, and then we describe the types of bugs with examples and their occurrence percentage (in parenthesis) in StackOverflow and GitHub.

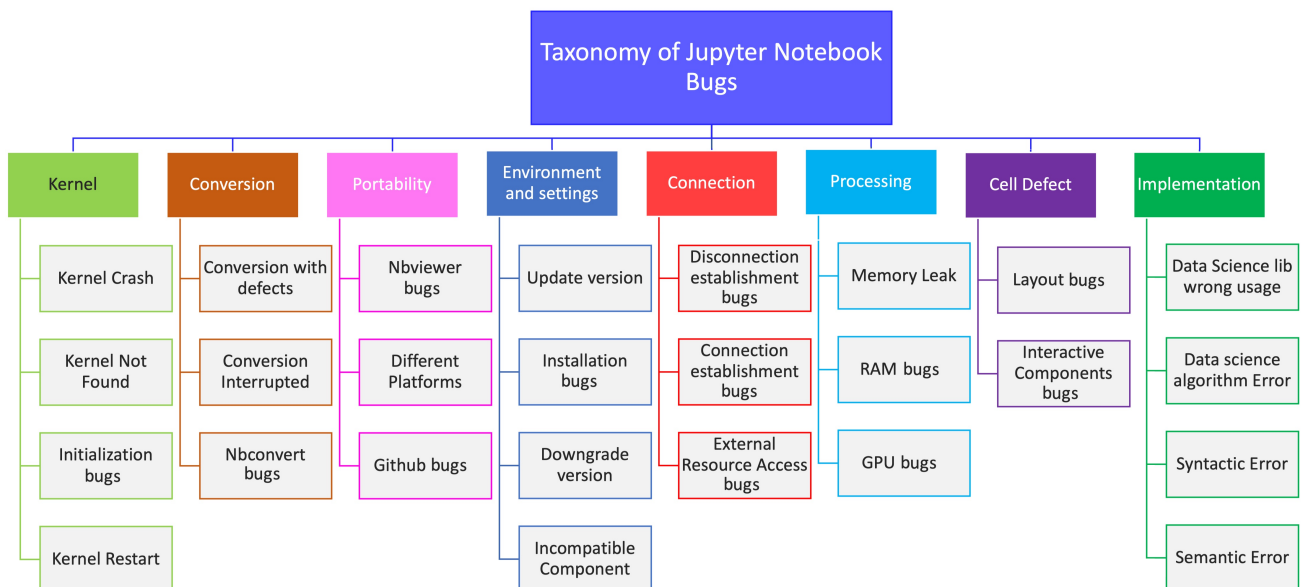


Figure 5.1 Taxonomy of Jupyter Notebook bugs.

**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:* It 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.
- *Initialization Bugs:* It 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) where 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%) .** It comprehends bugs related to errors during notebook conversion from .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:* It occurs when there is an attempt to convert a notebook to another format, but this conversion is interrupted.
- *Conversion with defects:* It happens when a conversion task finishes successfully, but its result contains unintended defects, for example, PDFs generated without images.
- *Nbconvert bugs:* It is related to bugs from nbconvert module, responsible for conversions using command line commands. In these cases, the conversation 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%).** It involves bugs that are related to Jupyter notebook execution in different environments. Although this feature is one of the pillars in the Jupyter project (JUPYTER, 2015), 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.* It 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 one, it 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%).** It 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, deprecated libraries, incompatibility between components and libraries, incompatibility with operational systems, problems with a package manager (such as Anaconda, PIP), and problems with installation and configuration of extensions.

- *Update and Downgrade Version:* It 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. The Bug #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%).** It happens when connecting the notebook with external resources, such as databases, hardware, and repositories. It can occur in two ways:

- *External Resource Access Bugs:* It 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: The 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: "... during Arduino usage some problems are difficult to know the root cause. In 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:* It 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 (CHATTOPADHYAY et al., 2020) 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 manipulat-*

*ing 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%).** It 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*: It refers to cell rendering problems, such as results beyond the margin, unexpected formulas, testing formatting, blank cells, graphics visualization problems, and so on. It can happen in any Jupyter notebook cell.
- *Interactive Components Bugs*: It 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."*

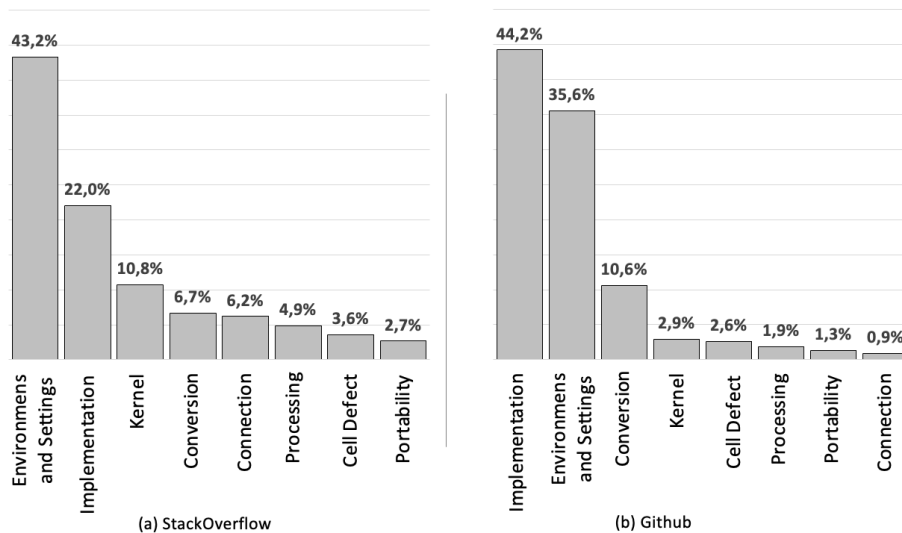
**Implementation | IP - (*StackOverflow* - 22% / *GitHub* - 44.2%).** Bugs related to implementation in general, syntax, logical, non-instantiated variables, algorithms, and semantics are examples of this type of bug. Analyzing all the posts and commits, we identified the following implementation bugs:

- *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, TensorFlow, and so on.
- *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** - In order to understand the frequency of each bug type previously discussed, we statistically analyzed the labeled data. Figure 5.2 shows the distribution of bug types in GitHub and StackOverflow. Bugs were caught on different platforms that have different dynamics and functionality. While on Github, bugs are reported during the development of a project, and their resolution directly impacts that project, 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.



**Figure 5.2** Frequency of bug types (a) StackOverflow and (b) Github.

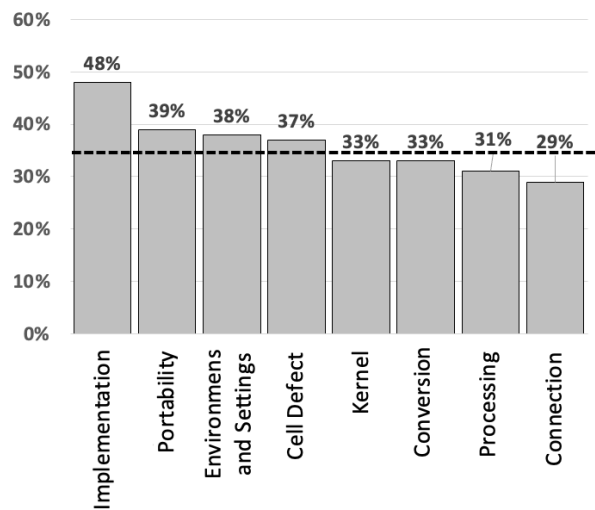
Looking at Github and StackOverflow datasets, the most frequent bug type was the "Environment and Settings" with 35.6% and 43.2%, respectively. It was also reinforced by the interviewees, which highlighted problems with version control, component incompatibility, wrong or missing installations, and problems 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 (see Fig. 5.3).

The "Implementation" and "Environment 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 Fig. 5.2).

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



**Figure 5.3** Average annual growth of bugs extracted from StackOverflow.

Previous research (PIMENTEL et al., 2019; PIMENTEL et al., 2021; WANG; LI; ZELLER, 2021) 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 (see Table 5.1). While the "ImportError", "ModuleNotFoundError" and "AttributeError" frequently occur in Environment and Settings bugs, the "TypeError", "AttributeError" and "NameError" appears in Implementation bugs. Table 5.1 summarizes the main problems found in each type of bug.



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	

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

**Table 5.1** Python Exceptions per Type of Bugs.

Finally, we used the Apriori algorithm (AGRAWAL; SRIKANT, 1994) 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 5.2 shows the top 10 associations between bug types and exceptions.

It is possible to observe in Table 5.2 that some relationships highlighted only by the volume of occurrence (Table 5.1) are validated, "ImportError", "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 existence of the occurrence relationship between exceptions and the type of bug. On the other hand, the confidence level of "AttributeError" against "Environments and Settings" is 52%. With a "Lift" less than one, there is unlikely any relationship between these occurrences.

It happens similarly to "TypeError", "AttributeError" and "NameError" concerning the "implementation" bugs, highlighted in Table 5.1. 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, 47%, but the "lift" greater than one 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 one as well.

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

**Table 5.2** Apriori Analysis Python Exceptions per Type of Bugs.

### 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, 38%, and 48% respectively. Although the Portability and Cell Defect bugs have had fewer occurrences, they have had above-average growth over the years.

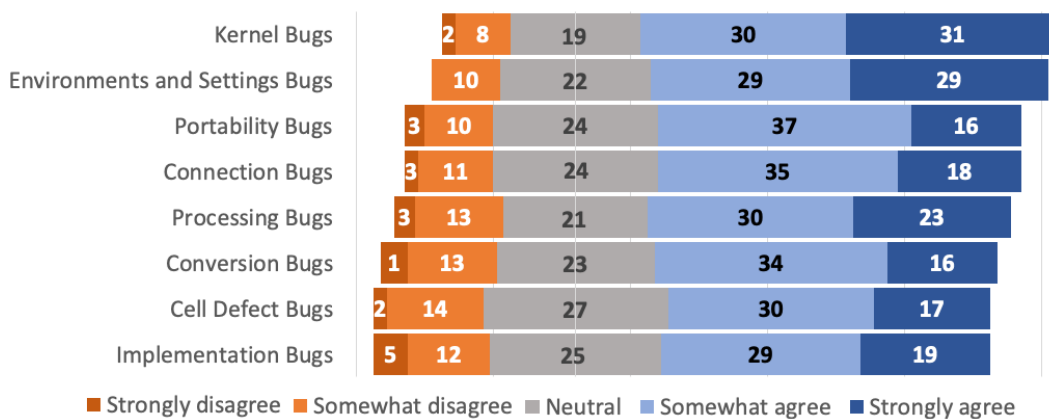


Figure 5.4 Bug Type Survey Validation.

In our survey (Figure 5.4), we defined each bug type category and then asked the respondents if they agreed/disagreed with each bug type. Among the 90 survey respondents, (i) 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"). This is important to reinforce our findings.

## 5.2 ROOT CAUSES OF BUGS (RQ2)

The root cause of bugs helps us understand their origin and how we can correct them. Table 5.3 shows the distribution of bug types according to their root causes. Following, 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 interviewee's speech below:

✓ *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 bug caused by code errors. Such as incorrectly assigning variables, developing repeating structures, errors in using libraries and functions, wrong plotting configuration, etc. This root cause has the main characteristic of causing a Run Time 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. Similar report below 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:* The 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%).** It happens when a component was or will be suspended.

*Example:* The post #63279999 and #67344009 (Stackoverflow) raised a Warning related to a matplotlib.

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

*Example:* The post #63339420 (Stack Overflow) 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:* The 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 "Run Time Error". It 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:* The 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:* The post #44634070 (Stackoverflow) highlights a kernel failure caused by an unknown situation, similarly the respondent reports: ✓ *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 5.3, shows 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.

Root Causes	ES		IP		KN		CV		CN		PC		CD		PB		Total
	SO	GH	SO	GH	SO	GH	SO	GH	SO	GH	SO	GH	SO	GH	SO	GH	
Install and Configuration Problems	635	117	36	0	56	12	44	0	42	0	0	0	6	7	12	3	970
Coding error	17	18	397	185	4	3	12	49	4	4	0	3	17	7	4	0	724
Version Problems	374	158	0	25	45	7	24	2	30	0	0	0	11	0	6	0	682
Unknown	34	9	58	54	117	2	60	1	38	2	1	0	22	1	14	0	413
Hardware software limitations	12	0	15	0	5	0	20	39	19	1	36	0	33	4	32	8	224
Logic error	2	1	32	114	9	1	0	0	0	0	4	0	4	2	0	0	169
Memory Error	3	0	20	0	34	0	2	0	7	0	78	12	0	1	1	0	158
TimeOut	4	0	7	0	5	0	9	0	18	0	6	1	0	0	0	0	50
Deprecation	21	1	1	0	1	0	0	0	1	0	0	0	0	0	0	0	25
Permission denied	16	0	3	0	2	0	1	0	1	1	1	0	0	0	0	0	25
Total	1118	304	569	378	278	25	172	91	160	8	126	16	93	22	69	11	

**Table 5.3** Frequency of Bug Type vs Root Cause  
(KN) Kernel, (CV) Conversion, (PB) Portability, (ES) Environment and Settings, (CN) Connection, (PC) Processing,  
(CD) Cell Defect, (IP) Implementation.

**Table 5.3** Frequency of Bug Type vs Root Cause

We could not identify all 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. It 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 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 (AGRAWAL; SRIKANT, 1994) was also applied to understand the association between the bug type and root causes. Table 5.4 shows the top 10 associations between bug types and root causes. The 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 these combinations.

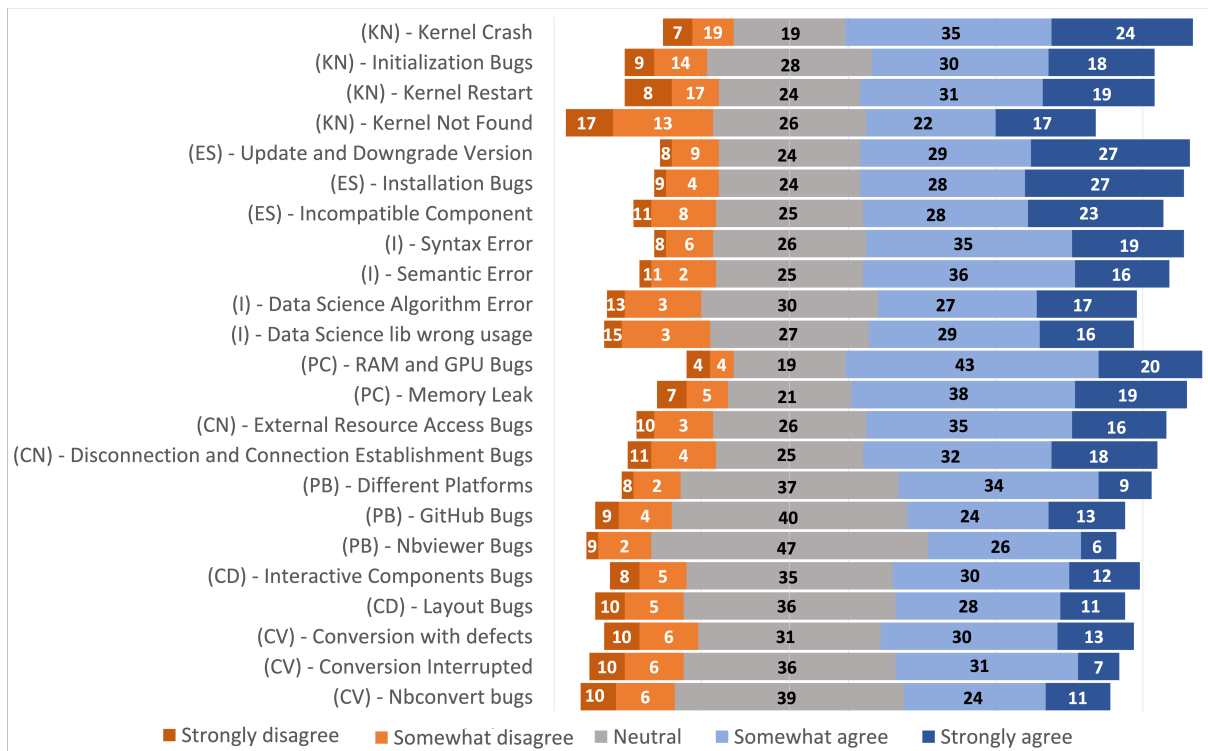


Antecedents	Consequents	Support	Confidence	Lift
Coding error	implementation	0.1710	0.8255	3.0087
Version Problems	Environments and Settings	0.1560	0.7843	1.9057
Install and Configuration Problems	Environments and Settings	0.2203	0.7748	1.8825
implementation	Coding error	0.1710	0.6231	3.0087
Environments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

**Table 5.4** Apriori Analysis Bug Type and Root Cause.

**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 5.5** Root Cause Survey Validation. (KN) Kernel, (CV) Conversion, (PB) Portability, (ES) Environment and Settings, (CN) Connection, (PC) Processing, (CD) Cell Defect, (IP) Implementation.

Figure 5.5 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 had agreement levels higher than disagreement levels. These results reinforce our findings and validate the proposed taxonomy.

**5.3 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 5.5 shows the distribution of bug types according to their impact. All of them are following described.

**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 the 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 the 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.

*Example:* The post #69351179 (Stackoverflow) shows a conversion error when using nbconvert.

**Run time Error - (*StackOverflow* - 57.5% | *GitHub* - 31.2%)**

Execution failure, usually accompanied by an error message.

*Example:* In post #70058518 (Stack Overflow) 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 the post #65666665 (Stack Overflow) the user reports discomfort with the "Warnings" 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.

Impacts	ES		IP		KN		CV		CN		PC		CD		PB		Total
	SO	GH	SO	GH	SO	GH	SO	GH	SO	GH	SO	GH	SO	GH	SO	GH	
Run Error	900	197	472	65	2	4	24	0	9	0	71	1	2	0	6	0	1753
Incorrect Functionality	41	90	64	261	0	10	96	91	7	8	1	0	86	19	55	11	840
Crash	142	16	1	0	275	11	49	0	142	0	8	1	4	0	8	0	657
Bad Performance	7	1	18	46	1	0	2	0	2	0	46	14	1	3	0	0	141
Warning	28	0	14	6	0	0	1	0	0	0	0	0	0	0	0	0	49
Total	1118	304	569	378	278	25	172	91	160	8	126	16	93	22	69	11	

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

**Table 5.5** Frequency of Impact vs Bug type

The most frequent impacts are Run Time Errors, Incorrect Functionality, and Crashes. The Run Time 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 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%), whose occurrence does not prevent the correct execution, but decreases the quality or performance and Warning (StackOverflow - 1.7% | GitHub - 0.7%), which does not impact on notebook functioning, but triggering an alert for the user.

The Apriori algorithm (AGRAWAL; SRIKANT, 1994) was also applied to understand the association between the bug type and impact. Table 5.6 shows the top 10 associations between bug types and impact. The results of the algorithm highlight the main relationships between types of bugs and impacts, "Run Time Error" is for "Environments and Configurations" and "Incorrect Functionality" is for "implementation", all have a "Lift" above 1 with emphasis on the bidirectional relationship found in the results for both combinations. In addition, despite the combination "Kernel Crash" not appearing in the number of occurrences in the data, the analysis of the algorithm highlights it as one of the main relationships between types of bugs and impacts, with a confidence level above 97%, reinforcing the statements of all interview participants have already encountered the bug/impact. This difference in the results between the analyses may be further evidence

that there is, in fact, a simple workaround solution used by users for the bug/impact, such as the "restart" of the kernel, as also mentioned by some interviewees.

Antecedents	Consequents	Support	Confidence	Lift
Kernel	Crash	0.0811	0.9753	5.0530
Environments and Settings	Run Time Error	0.3199	0.7773	1.5172
conversion	Incorrect Functionality	0.0549	0.7110	2.9734
Run Time Error	Environments and Settings	0.3199	0.6244	1.5172
implementation	Run Time Error	0.1578	0.5749	1.1222
Crash	Kernel	0.0811	0.4201	5.0530
Incorrect Functionality	implementation	0.0917	0.3833	1.3969
implementation	Incorrect Functionality	0.0917	0.3340	1.3969
Run Time Error	implementation	0.1578	0.3079	1.1222
Incorrect Functionality	conversion	0.0549	0.2297	2.9734

**Table 5.6** Apriori Analysis Bug Type and Impact.

### Finding 3

The most frequent impacts from bugs in Jupyter notebooks are: Run Time 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.

## 5.4 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. (KIM et al., 2016) 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 (Table 4.2), 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."*

**The 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 that 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."*

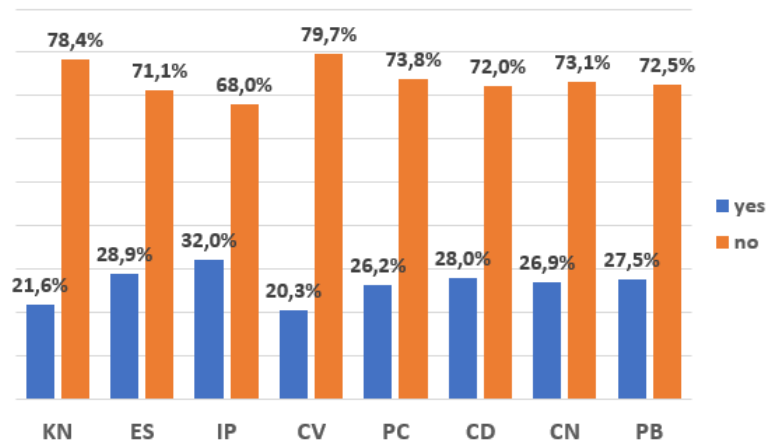
**Testing and debugging.** Some interviewees pointed out the lack of basic testing tools as a challenge to be addressed. They also detailed the process of fixing a bug using a trial and error approach. Among the interviewees, especially those with a software engineering background, they pointed out specific 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."*

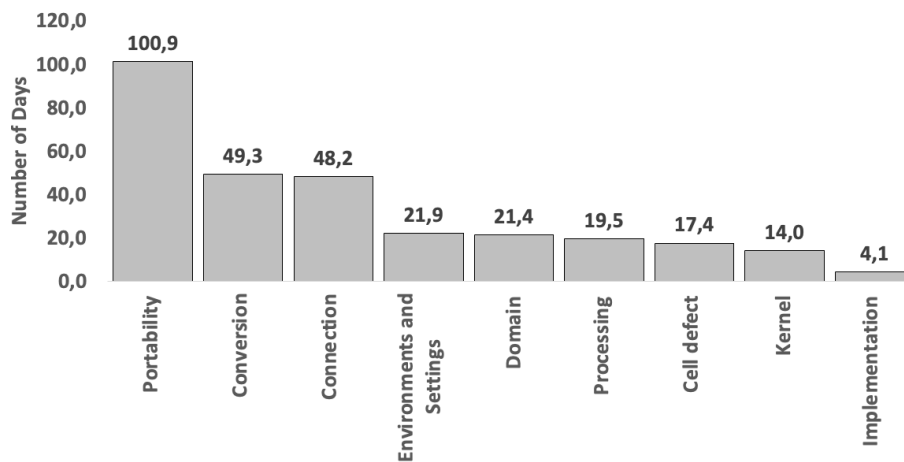
In addition, software debugging is an essential activity to improve code quality during development, as well as the difficulty in developing tests; according to the interviewees'

reports, debugging the code may have the same root cause, the difficulty of inspecting the code, can influence the data scientist's ability to identify or even fix a bug, which may affect the number of accepted answers (the answer that solved the problem), the number of unanswered posts, or the acceptance time of an answer in StackOverflow.

Figure 5.6 shows the number of questions reported with the accepted answer, considering each bug type in our StackOverflow dataset. All the bug types have a similar average (27.9%) of accepted answers. Figure 5.7 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, at least 4 out of 8 bug types are above average.



**Figure 5.6** Bug reports with accepted answers (StackOverflow).



**Figure 5.7** Average accepted response time (in days).



#### Finding 4

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; and next, it can be encapsulated and added to another system to use it (CHAT-TOPADHYAY et al., 2020). Some interviewees (DS3, DS5, DS7, DS8, DS9, DS11, DS12, DS14, DS15, DS16, DS17, 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:

✓ *DS11: "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 appears in StackOverflow annual survey<sup>1</sup> since 2017, and as shown in Table 5.7, the posts and bugs on StackOverflow only increased. It reinforces the im-

<sup>1</sup><https://insights.StackOverflow.com/survey>

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

**Table 5.7** Jupyter History in StackOverflow - General Posts related to Jupyter vs Posts related to Bugs in Jupyter

portance of evolving the tool with features to mitigate bugs and help data scientists to do exploratory analysis, prototyping, 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.

## 5.5 DISCUSSION

In this section, we discuss the implications of our study's results. In particular, 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" (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 the software engineering community researchers and practitioners alike to increase awareness about the existing techniques and make such tools available for data scientists. Also, 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, it is not standard practice yet, as mentioned by interviewee 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. 95% of our respondents reported understanding that the main (or only) usefulness is in these steps, as 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 (see Table 5.8) 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's why it's 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.

Feature	Description	Occurrences
Indentation corrector <sup>2</sup>	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, etc.	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 (KERY et al., 2020)	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 (HEAD et al., 2019)	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 <sup>3</sup>	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%

**Table 5.8** Features mentioned by respondents.

## 5.6 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.
  - 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 to 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.

## 5.7 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 predefined list also used in previous research (GARCIA et al., 2020; MAKHSHARI; MESBAH, 2021), 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. In order to mitigate this threat, three authors (2nd, 3rd, and 4th) 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 Stack Overflow can be a threat to our study. To mitigate this threat, we used an approach similar to (ISLAM et al., 2019), 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 Stack Overflow 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 Stack Overflow, and authors' judgment. We mitigated this threat by investigating 105 Jupyter Notebook repositories, 14,740 valid commits, 30,416 Stack Overflow 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 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 analysis, sometimes asking for clarifications.

## 5.8 CHAPTER SUMMARY

This chapter examines various types of bugs encountered in Jupyter projects, drawing from data collected from GitHub, StackOverflow, and interviews. It proposes an initial taxonomy refined through qualitative and statistical analyses, highlighting the prevalence of bugs related to environment settings and implementation.

The investigation of root causes of bugs in Jupyter projects reveals common issues like installation, version conflicts, and coding errors. Statistical analysis validates these findings, emphasizing the importance of addressing these issues to improve user experience and efficiency in Jupyter projects.

Examining the impacts of bugs categorizes them based on severity, with runtime errors and incorrect functionality emerging as the most common. Statistical analysis highlights the relationships between bug types and impacts, underscoring the critical need to address bugs effectively to maintain stability and usability in Jupyter notebook projects.

Next chapter concludes a detailed study on the challenges in Jupyter Notebook development, providing insights into the most common bugs and their causes, such as configuration and version issues. It highlights the lack of standard software engineering practices, such as version control, and the need for advanced debugging tools. The research also lists user-desired features to facilitate the transition from analysis to production. The research outputs include empirical data, a taxonomy of bugs, and a published paper. The chapter suggests future improvements and the development of algorithms to classify bugs, aiming to enhance the tool and assist data scientists in their projects.

# 6

## CONCLUSIONS

The research contributes valuable insights into the challenges and opportunities within Jupyter Notebook development, addressing key issues that impact data scientists.

### 6.1 SUMMARY OF RESEARCH CONTRIBUTIONS

The main contributions of the work are:

**Identification of Most Frequent Bugs.** The study reveals that the most common bugs in Jupyter notebooks are related to Environments and Settings, constituting a significant portion of the analyzed posts from StackOverflow and GitHub issues. Configuration issues, version problems, and deprecation-related issues emerge as major root causes. This finding emphasizes the need for interventions to streamline these issues, potentially saving substantial time for data scientists.

**Addressing Incorrect Algorithm Implementations.** Incorrect algorithm implementations are a prevalent source of bugs, indicating a lack of familiarity among data scientists with well known software quality assurance techniques such as unit testing and bug localization. The study calls for action within the software engineering community to increase awareness and provide tools that seamlessly integrate with Jupyter notebooks, facilitating the adoption of these techniques.

**Highlighting Missing Software Engineering Functionalities.** The research underscores the absence of standard software engineering practices in Jupyter notebook development. Notably, the lack of widespread use of version control systems like Git in Jupyter notebooks is highlighted. The study suggests the need for tools that enable visualizing differences in GUI components across notebook versions. Additionally, it identifies a demand for features like previewing raw datasets, advanced debugging capabilities, and tools for efficient code cleaning and adaptation.

**Understanding Jupyter Notebook's Strengths and Challenges.** The study acknowledges Jupyter Notebook's strength in data analysis, investigation, and exploration. However, it points out challenges in transitioning analysis into production code. A sub-



stantial percentage of respondents reported difficulties in cleaning and adapting code for deployment. This insight sheds light on the practical challenges faced by users and provides a basis for improving the tool's functionality.

**Compilation of User-Requested Features.** Through bug analysis and interviews, the research compiles a non-exhaustive list of desired features by users. The identified features encompass extensions for data exploration, transformation of analysis into production code, and streamlined solutions for common issues. The study emphasizes the importance of unified packages or standard tool versions to ensure seamless integration and avoid compatibility and configuration errors.

In summary, this work contributes significantly to the understanding of challenges in Jupyter notebook development and provides valuable recommendations for improving the tool's functionality, fostering collaboration between data scientists and the software engineering community.

## 6.2 RESEARCH PRODUCTS

The conducted research has generated several tangible results that provide support for highly relevant contributions to the scientific community. These contributions have been thoroughly explored throughout this dissertation and summarized in this chapters .

**Empirical Data.** Encompassing raw data, analyzed data, and the associated procedures for their collection and analysis. Empirical data serves as a robust foundation for understanding and validating the presented findings, promoting transparency and replicability in research. <sup>1</sup>

**Taxonomy.** Noteworthy is the creation of an initial Taxonomy (Figure 5.1), providing a primary classification of bugs in Jupyter projects. This taxonomic framework offers a systematic and comprehensible organization of the various bug categories identified, facilitating understanding and practical approaches to these issues.

**Journal Paper.** A paper resulting from this investigation was accepted for publication:

- SANTANA, Taijara; ANSELMO, Paulo; ALMEIDA, Eduardo; AHMED, Iftekhhar. Bug Analysis in Jupyter Notebook Projects: An Empirical Study. In: Transactions on Software Engineering and Methodology. (SANTANA et al., 2024)

## 6.3 FUTURE WORK

Based on the study's conclusions, some suggestions for future work could include:

**Enhancement of Analysis-to-Production Transition.** Given the reported difficulty users face in transitioning from analysis in Jupyter Notebook to production code, a future project could explore solutions to facilitate this transition. This might involve developing features that streamline code cleaning and adaptation, as well as building

---

<sup>1</sup><https://github.com/bugsjupyterempiricalstudy/BugJupyterPaper>

bridges between the exploratory environment of Jupyter and more traditional production environments. This approach aims to optimize the effective integration of analysis results into the software development workflow.

**Machine Learning Algorithm.** The development of a machine learning algorithm using the manually categorized dataset as a base. This algorithm could be trained to classify the entire historical dataset available on Stack Overflow, enabling the evolution of the current taxonomy.

## 6.4 CONCLUDING REMARKS

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 Stack Overflow 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, ten 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 Run Time 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 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.

## REFERENCES

AGRAWAL, A. et al. We don't need another hero?: the impact of "heroes" on software development. In: PAULISCH, F.; BOSCH, J. (Ed.). *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2018, Gothenburg, Sweden, May 27 - June 03, 2018*. Gothenburg, Sweden: ACM, 2018. p. 245–253. Disponível em: <<https://doi.org/10.1145/3183519.3183549>>.

AGRAWAL, R.; SRIKANT, R. Fast algorithms for mining association rules in large databases. In: *Proceedings of the 20th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994. (VLDB '94), p. 487–499. ISBN 1558601538.

BEGEL, A.; ZIMMERMANN, T. Analyze this! 145 questions for data scientists in software engineering. In: JALOTE, P.; BRIAND, L. C.; HOEK, A. van der (Ed.). *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*. Hyderabad, India: ACM, 2014. p. 12–23. Disponível em: <<https://doi.org/10.1145/2568225.2568233>>.

CAO, L. Data science: A comprehensive overview. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 50, n. 3, Jun 2017.

CHATTOPADHYAY, S. et al. What's wrong with computational notebooks? pain points, needs, and design opportunities. In: *CHI '20: CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, April 25-30, 2020*. Honolulu, HI, USA: ACM, 2020. p. 1–12.

DHAR, V. Data science and prediction. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 56, n. 12, p. 64–73, dec 2013. ISSN 0001-0782.

FUSCH, P.; NESS, L. Are we there yet? data saturation in qualitative research. In: *Qualitative Report*. Minneapolis, MM, USA: Nova Southeastern University, 2015. p. 1408–1416.

GARCIA, J. et al. A comprehensive study of autonomous vehicle bugs. In: *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*. Seoul, South Korea: ACM, 2020. p. 385–396.

HEAD, A. et al. Managing messes in computational notebooks. In: BREWSTER, S. A. et al. (Ed.). *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*. Glasgow, Scotland, UK: ACM, 2019. p. 270. Disponível em: <<https://doi.org/10.1145/3290605.3300500>>.

- ISLAM, M. J. et al. 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*. New York, NY, USA: Association for Computing Machinery, 2019. (ESEC/FSE 2019), p. 510–520. ISBN 9781450355728.
- JUPYTER, P. *Project Jupyter: Computational Narratives as the Engine of Collaborative Data Scienc.* 2015. Last accessed 07 July 2015. Disponível em: <<https://blog.jupyter.org/project-jupyter-computational-narratives-as-the-engine-of-collaborative-data-science-2b5fb94c3c58>>.
- KANDEL, S. et al. Enterprise data analysis and visualization: An interview study. *IEEE Trans. Vis. Comput. Graph.*, v. 18, n. 12, p. 2917–2926, 2012.
- KERY, M. B. et al. The story in the notebook: Exploratory data science using a literate programming tool. In: MANDRYK, R. L. et al. (Ed.). *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI 2018, Montreal, QC, Canada, April 21-26, 2018*. Montreal, QC, Canada: ACM, 2018. p. 174. Disponível em: <<https://doi.org/10.1145/3173574.3173748>>.
- KERY, M. B. et al. mage: Fluid moves between code and graphical work in computational notebooks. In: IQBAL, S. T. et al. (Ed.). *UIST '20: The 33rd Annual ACM Symposium on User Interface Software and Technology, Virtual Event, USA, October 20-23, 2020*. Virtual Event, USA: ACM, 2020. p. 140–151. Disponível em: <<https://doi.org/10.1145/3379337.3415842>>.
- KIM, M. et al. The emerging role of data scientists on software development teams. In: *Proceedings of the 38th International Conference on Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2016. (ICSE '16), p. 96–107. ISBN 9781450339001. Disponível em: <<https://doi.org/10.1145/2884781.2884783>>.
- KITCHENHAM, B. A.; PFLEEGER, S. L. Personal opinion surveys. In: SHULL, F.; SINGER, J.; SJØBERG, D. I. (Ed.). *Guide to Advanced Empirical Software Engineering*. [S.l.]: Springer London, 2008. p. 63–92.
- KOENZEN, A. P.; ERNST, N. A.; STOREY, M. D. Code duplication and reuse in jupyter notebooks. In: HOMER, M. et al. (Ed.). *IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2020, Dunedin, New Zealand, August 10-14, 2020*. Dunedin, New Zealand: IEEE, 2020. p. 1–9.
- KRISHNA, R. et al. What is the connection between issues, bugs, and enhancements?: lessons learned from 800+ software projects. In: PAULISCH, F.; BOSCH, J. (Ed.). *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2018, Gothenburg, Sweden, May 27 - June 03, 2018*. Gothenburg, Sweden: ACM, 2018. p. 306–315. Disponível em: <<https://doi.org/10.1145/3183519.3183548>>.
- LANDIS, J. R.; KOCH, G. G. The measurement of observer agreement for categorical data. *Biometrics*, International Biometric Society, v. 33, n. 1, 1977.

MAKHSHARI, A.; MESBAH, A. Iot bugs and development challenges. In: *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. Madrid, Spain: IEEE, 2021. p. 460–472.

MUNAIHAH, N. et al. Curating github for engineered software projects. *PeerJ Prepr.*, v. 4, p. e2617, 2016. Disponible em: <<https://doi.org/10.7287/peerj.preprints.2617v1>>.

PATRA, J.; PRADEL, M. Nalin: Learning from runtime behavior to find name-value inconsistencies in jupyter notebooks. In: *44th IEEE/ACM International Conference on Software Engineering, ICSE 2022, Pittsburgh, USA, 22-30 May 2021*. Pittsburgh, USA: Association for Computing Machinery, 2021.

PERKEL, J. M. Why jupyter is data scientists' computational notebook of choice. In: . <https://www.nature.com/articles/d41586-018-07196-1>: MEDLINE, 2018. Disponible em: <<https://doi.org/10.1038/d41586-018-07196-1>>.

PIMENTEL, J. F. et al. A large-scale study about quality and reproducibility of jupyter notebooks. In: STOREY, M. D.; ADAMS, B.; HAIDUC, S. (Ed.). *Proceedings of the 16th International Conference on Mining Software Repositories, MSR 2019, 26-27 May 2019, Montreal, Canada*. Montreal, Canada: IEEE / ACM, 2019. p. 507–517.

PIMENTEL, J. F. et al. Understanding and improving the quality and reproducibility of jupyter notebooks. *Empir. Softw. Eng.*, v. 26, n. 4, p. 65, 2021. Disponible em: <<https://doi.org/10.1007/s10664-021-09961-9>>.

RAHMAN, A. et al. Characterizing the influence of continuous integration: empirical results from 250+ open source and proprietary projects. In: BAYSAL, O.; MENZIES, T. (Ed.). *Proceedings of the 4th ACM SIGSOFT International Workshop on Software Analytics, SWAN@ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 5, 2018*. Lake Buena Vista, FL, USA: ACM, 2018. p. 8–14. Disponible em: <<https://doi.org/10.1145/3278142.3278149>>.

RAHMAN, A. et al. Gang of eight: A defect taxonomy for infrastructure as code scripts. In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2020. (ICSE '20), p. 752–764. ISBN 9781450371216.

RULE, A. et al. Ten simple rules for writing and sharing computational analyses in jupyter notebooks. In: . MEDLINE, 2019. Disponible em: <<https://doi.org/10.1371/journal.pcbi.1007007>>.

RULE, A. et al. Aiding collaborative reuse of computational notebooks with annotated cell folding. *Proc. ACM Hum.-Comput. Interact.*, Association for Computing Machinery, New York, NY, USA, v. 2, n. CSCW, nov 2018. Disponible em: <<https://doi.org/10.1145/3274419>>.

RULE, A.; TABARD, A.; HOLLAN, J. D. Exploration and explanation in computational notebooks. In: MANDRYK, R. L. et al. (Ed.). *Proceedings of the 2018 CHI Conference on*

*Human Factors in Computing Systems, CHI 2018, Montreal, QC, Canada, April 21-26, 2018*. Montreal, QC, Canada: ACM, 2018. p. 32.

RULE, A.; TABARD, A.; HOLLAN, J. D. Exploration and explanation in computational notebooks. In: \_\_\_\_\_. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2018. p. 1–12. ISBN 9781450356206. Disponível em: <<https://doi.org/10.1145/3173574.3173606>>.

SALDAÑA, J. *The Coding Manual for Qualitative Researchers*. Sage, 2009. ISBN 9781847875488. Disponível em: <<https://books.google.com.br/books?id=OE7LngEACAAJ>>.

SALDANA, J. *The Coding Manual for Qualitative Researchers*. [S.l.]: SAGE Publications, 2015. ISBN 9781473943582.

SANTANA, T. L. de et al. Bug analysis in jupyter notebook projects: An empirical study. *ACM Trans. Softw. Eng. Methodol.*, Association for Computing Machinery, New York, NY, USA, jan 2024. ISSN 1049-331X. Just Accepted. Disponível em: <<https://doi.org/10.1145/3641539>>.

SEIDMAN, I. *Interviewing as Qualitative Research: A Guide for Researchers in Education and the Social Sciences, 3rd Edition*. [S.l.]: Teachers College Press, 2006. ISBN 0807746665, 978-0807746660.

SERRAT, O. The five whys technique. In: \_\_\_\_\_. *Knowledge Solutions: Tools, Methods, and Approaches to Drive Organizational Performance*. Singapore: Springer Singapore, 2017. p. 307–310. ISBN 978-981-10-0983-9. Disponível em: <[https://doi.org/10.1007/978-981-10-0983-9\\_32](https://doi.org/10.1007/978-981-10-0983-9_32)>.

TABA, S. E. S. et al. Predicting bugs using antipatterns. In: *2013 IEEE International Conference on Software Maintenance*. [S.l.: s.n.], 2013. p. 270–279.

TAO, Y. et al. Understanding performance concerns in the api documentation of data science libraries. In: \_\_\_\_\_. *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2020. p. 895–906. ISBN 9781450367684.

THUNG, F. et al. An empirical study of bugs in machine learning systems. In: *23rd IEEE International Symposium on Software Reliability Engineering, ISSRE 2012, Dallas, TX, USA, November 27-30, 2012*. Dallas, TX, USA: IEEE Computer Society, 2012. p. 271–280.

WANG, A. Y. et al. How data scientists use computational notebooks for real-time collaboration. *Proc. ACM Hum.-Comput. Interact.*, Association for Computing Machinery, New York, NY, USA, v. 3, n. CSCW, nov 2019. Disponível em: <<https://doi.org/10.1145/3359141>>.

WANG, D. et al. 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*. New York, NY, USA: Association for Computing Machinery, 2021. (ESEC/FSE 2021), p. 20–31. ISBN 9781450385626.

WANG, J. et al. Assessing and restoring reproducibility of jupyter notebooks. In: \_\_\_\_\_. *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2020. p. 138–149. ISBN 9781450367684.

WANG, J.; LI, L.; ZELLER, A. Better code, better sharing: on the need of analyzing jupyter notebooks. In: ROTHERMEL, G.; BAE, D. (Ed.). *ICSE-NIER 2020: 42nd International Conference on Software Engineering, New Ideas and Emerging Results, Seoul, South Korea, 27 June - 19 July, 2020*. Seoul, South Korea: ACM, 2020. p. 53–56.

WANG, J.; LI, L.; ZELLER, A. 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*. New York, NY, USA: Association for Computing Machinery, 2020. (ICSE-NIER '20), p. 53–56. ISBN 9781450371261. Disponível em: <<https://doi.org/10.1145/3377816.3381724>>.

WANG, J.; LI, L.; ZELLER, A. Restoring execution environments of jupyter notebooks. In: *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. Madrid, Spain: IEEE, 2021. p. 1622–1633.

YANG, C. et al. Subtle bugs everywhere: Generating documentation for data wrangling code. In: *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*. Melbourne, Australia: IEEE, 2021. p. 304–316. Disponível em: <<https://doi.org/10.1109/ASE51524.2021.9678520>>.

ZHANG, Y. et al. An empirical study on tensorflow program bugs. In: *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*. New York, NY, USA: Association for Computing Machinery, 2018. (ISSTA 2018), p. 129–140. ISBN 9781450356992.

## APPENDIX A

### LIST OF GITHUB PROJECTS

Table A.1 shows all GitHub projects analyzed in this work.

<b>Id</b>	<b>URL Repository</b>
65392849	<a href="https://github.com/ratschlab/metagraph">https://github.com/ratschlab/metagraph</a>
237890763	<a href="https://github.com/fastai/fastpages">https://github.com/fastai/fastpages</a>
126337574	<a href="https://github.com/pyiron/pyiron">https://github.com/pyiron/pyiron</a>
219892487	<a href="https://github.com/aws/aws-neuron-sdk">https://github.com/aws/aws-neuron-sdk</a>
99244384	<a href="https://github.com/Qiskit/qiskit-tutorials">https://github.com/Qiskit/qiskit-tutorials</a>
35236880	<a href="https://github.com/AllenInstitute/AllenSDK">https://github.com/AllenInstitute/AllenSDK</a>
114581326	<a href="https://github.com/MinaProtocol/mina">https://github.com/MinaProtocol/mina</a>
247054942	<a href="https://github.com/dsfsi/covid19za">https://github.com/dsfsi/covid19za</a>
114898943	<a href="https://github.com/SeldonIO/seldon-core">https://github.com/SeldonIO/seldon-core</a>
141692400	<a href="https://github.com/UCL-CCS/EasyVVUQ">https://github.com/UCL-CCS/EasyVVUQ</a>
86884600	<a href="https://github.com/openforcefield/openff-toolkit">https://github.com/openforcefield/openff-toolkit</a>
112729129	<a href="https://github.com/CLIMADA-project/climada-python">https://github.com/CLIMADA-project/climada-python</a>
129317474	<a href="https://github.com/tensorflow/docs">https://github.com/tensorflow/docs</a>
125098252	<a href="https://github.com/onnx/tensorflow-onnx">https://github.com/onnx/tensorflow-onnx</a>
178290396	<a href="https://github.com/fastestimator/fastestimator">https://github.com/fastestimator/fastestimator</a>
40307735	<a href="https://github.com/flatironinstitute/CaImAn">https://github.com/flatironinstitute/CaImAn</a>
142140875	<a href="https://github.com/signals-dev/Orion">https://github.com/signals-dev/Orion</a>
160882537	<a href="https://github.com/ecmwf/magics">https://github.com/ecmwf/magics</a>
62352963	<a href="https://github.com/kundajelab/tfmodisco">https://github.com/kundajelab/tfmodisco</a>
249056404	<a href="https://github.com/Yu-Group/covid19-severity-prediction">https://github.com/Yu-Group/covid19-severity-prediction</a>
265612440	<a href="https://github.com/coqui-ai/TTS">https://github.com/coqui-ai/TTS</a>
161231707	<a href="https://github.com/rapidsai/cuxfilter">https://github.com/rapidsai/cuxfilter</a>
246738676	<a href="https://github.com/google/automl">https://github.com/google/automl</a>
151619717	<a href="https://github.com/google-research/google-research">https://github.com/google-research/google-research</a>
40161124	<a href="https://github.com/vega/ipyvega">https://github.com/vega/ipyvega</a>
34146607	<a href="https://github.com/mmagnus/rna-tools">https://github.com/mmagnus/rna-tools</a>
248347348	<a href="https://github.com/lindawangg/COVID-Net">https://github.com/lindawangg/COVID-Net</a>
195062961	<a href="https://github.com/kili-technology/kili-playground">https://github.com/kili-technology/kili-playground</a>
261861733	<a href="https://github.com/keras-team/keras-io">https://github.com/keras-team/keras-io</a>
234676361	<a href="https://github.com/usc-isi-i2/kgtk">https://github.com/usc-isi-i2/kgtk</a>
24733345	<a href="https://github.com/3dmol/3Dmol.js">https://github.com/3dmol/3Dmol.js</a>
166853419	<a href="https://github.com/lappis-unb/rasa-ptbr-boilerplate">https://github.com/lappis-unb/rasa-ptbr-boilerplate</a>
246634306	<a href="https://github.com/NVIDIA/TRTorch">https://github.com/NVIDIA/TRTorch</a>
81518954	<a href="https://github.com/quiltdata/quilt">https://github.com/quiltdata/quilt</a>



10456942	<a href="https://github.com/MTG/essentia">https://github.com/MTG/essentia</a>
244029618	<a href="https://github.com/data-science-on-aws/workshop">https://github.com/data-science-on-aws/workshop</a>
161840815	<a href="https://github.com/goldmansachs/gs-quant">https://github.com/goldmansachs/gs-quant</a>
90328920	<a href="https://github.com/intel-analytics/analytics-zoo">https://github.com/intel-analytics/analytics-zoo</a>
278415292	<a href="https://github.com/open-mmlab/mmlclassification">https://github.com/open-mmlab/mmlclassification</a>
203975315	<a href="https://github.com/gazprom-neft/seismiqb">https://github.com/gazprom-neft/seismiqb</a>
180192894	<a href="https://github.com/google/neural-tangents">https://github.com/google/neural-tangents</a>
75641327	<a href="https://github.com/ExoCTK/exoctk">https://github.com/ExoCTK/exoctk</a>
233594133	<a href="https://github.com/nccr-itmo/FEDOT">https://github.com/nccr-itmo/FEDOT</a>
169880381	<a href="https://github.com/facebook/Ax">https://github.com/facebook/Ax</a>
157752451	<a href="https://github.com/rapidsai/cugraph">https://github.com/rapidsai/cugraph</a>
119419202	<a href="https://github.com/wandb/examples">https://github.com/wandb/examples</a>
254453761	<a href="https://github.com/reichlab/covid19-forecast-hub">https://github.com/reichlab/covid19-forecast-hub</a>
193922326	<a href="https://github.com/enzoampil/fastquant">https://github.com/enzoampil/fastquant</a>
56300322	<a href="https://github.com/pastas/pastas">https://github.com/pastas/pastas</a>
186871969	<a href="https://github.com/peterdsharpe/AeroSandbox">https://github.com/peterdsharpe/AeroSandbox</a>
85799683	<a href="https://github.com/christophM/interpretable-ml-book">https://github.com/christophM/interpretable-ml-book</a>
110973589	<a href="https://github.com/Eomys/pylecan">https://github.com/Eomys/pylecan</a>
32540955	<a href="https://github.com/datacarpentry/python-ecology-lesson">https://github.com/datacarpentry/python-ecology-lesson</a>
101073856	<a href="https://github.com/zlatko-minev/pyEPR">https://github.com/zlatko-minev/pyEPR</a>
43358723	<a href="https://github.com/OceanParcels/parcels">https://github.com/OceanParcels/parcels</a>
240286467	<a href="https://github.com/galaxyproject/SARS-CoV-2">https://github.com/galaxyproject/SARS-CoV-2</a>
96946693	<a href="https://github.com/tensorflow/tpu">https://github.com/tensorflow/tpu</a>
251355291	<a href="https://github.com/micro-manager/pycro-manager">https://github.com/micro-manager/pycro-manager</a>
123097163	<a href="https://github.com/GeoscienceAustralia/dea-notebooks">https://github.com/GeoscienceAustralia/dea-notebooks</a>
223636350	<a href="https://github.com/pycaret/pycaret">https://github.com/pycaret/pycaret</a>
211972484	<a href="https://github.com/tensorflow/fairness-indicators">https://github.com/tensorflow/fairness-indicators</a>
249628101	<a href="https://github.com/datadesk/california-coronavirus-data">https://github.com/datadesk/california-coronavirus-data</a>
116963730	<a href="https://github.com/LiberTEM/LiberTEM">https://github.com/LiberTEM/LiberTEM</a>
107937815	<a href="https://github.com/aws/amazon-sagemaker-examples">https://github.com/aws/amazon-sagemaker-examples</a>
251363143	<a href="https://github.com/Azure-Samples/Synapse">https://github.com/Azure-Samples/Synapse</a>
165826654	<a href="https://github.com/deepmind/deepmind-research">https://github.com/deepmind/deepmind-research</a>
97071697	<a href="https://github.com/ironmussa/Optimus">https://github.com/ironmussa/Optimus</a>
237077697	<a href="https://github.com/AnalyticalGraphicsInc/STKCodeExamples">https://github.com/AnalyticalGraphicsInc/STKCodeExamples</a>
143113454	<a href="https://github.com/alan-turing-institute/MLJ.jl">https://github.com/alan-turing-institute/MLJ.jl</a>
247269797	<a href="https://github.com/Azure/live-video-analytics">https://github.com/Azure/live-video-analytics</a>
180872240	<a href="https://github.com/numenta/nupic.research">https://github.com/numenta/nupic.research</a>
24510911	<a href="https://github.com/NOAA-GFDL/MOM6-examples">https://github.com/NOAA-GFDL/MOM6-examples</a>
249465310	<a href="https://github.com/rajeshrinet/pyross">https://github.com/rajeshrinet/pyross</a>
1404690	<a href="https://github.com/PmagPy/PmagPy">https://github.com/PmagPy/PmagPy</a>
287642401	<a href="https://github.com/whylabs/whylogs">https://github.com/whylabs/whylogs</a>
108200238	<a href="https://github.com/geomstats/geomstats">https://github.com/geomstats/geomstats</a>
102973646	<a href="https://github.com/fastai/fastai">https://github.com/fastai/fastai</a>
239769118	<a href="https://github.com/mackelab/sbi">https://github.com/mackelab/sbi</a>

112795497	<a href="https://github.com/SURGroup/UQpy">https://github.com/SURGroup/UQpy</a>
254142161	<a href="https://github.com/Seagate/cortex">https://github.com/Seagate/cortex</a>
222507066	<a href="https://github.com/fastai/nbdev">https://github.com/fastai/nbdev</a>
212628259	<a href="https://github.com/Azure/Azure-Sentinel-Notebooks">https://github.com/Azure/Azure-Sentinel-Notebooks</a>
54376220	<a href="https://github.com/AtsushiSakai/PythonRobotics">https://github.com/AtsushiSakai/PythonRobotics</a>
108053674	<a href="https://github.com/tensorflow/probability">https://github.com/tensorflow/probability</a>
180372498	<a href="https://github.com/plotly/dash-sample-apps">https://github.com/plotly/dash-sample-apps</a>
73234795	<a href="https://github.com/SMTorg/smt">https://github.com/SMTorg/smt</a>
235856774	<a href="https://github.com/tensorflow/docs-l10n">https://github.com/tensorflow/docs-l10n</a>
17371412	<a href="https://github.com/h2oai/h2o-3">https://github.com/h2oai/h2o-3</a>
205382115	<a href="https://github.com/SubstraFoundation/distributed-learning-contributivity">https://github.com/SubstraFoundation/distributed-learning-contributivity</a>
47548304	<a href="https://github.com/ARM-software/lisa">https://github.com/ARM-software/lisa</a>
201998885	<a href="https://github.com/theislab/scarches">https://github.com/theislab/scarches</a>
200722670	<a href="https://github.com/NVIDIA/NeMo">https://github.com/NVIDIA/NeMo</a>
145183271	<a href="https://github.com/Azure/Azure-Sentinel">https://github.com/Azure/Azure-Sentinel</a>
60551519	<a href="https://github.com/spacetelescope/jwst">https://github.com/spacetelescope/jwst</a>
256832500	<a href="https://github.com/castorini/pygaggle">https://github.com/castorini/pygaggle</a>
8308849	<a href="https://github.com/siddhartha-gadgil/ProvingGround">https://github.com/siddhartha-gadgil/ProvingGround</a>
148885351	<a href="https://github.com/scikit-hep/boost-histogram">https://github.com/scikit-hep/boost-histogram</a>
42720167	<a href="https://github.com/KDE/cantor">https://github.com/KDE/cantor</a>
169442310	<a href="https://github.com/amaiya/ktrain">https://github.com/amaiya/ktrain</a>
178316220	<a href="https://github.com/collective-action/tech">https://github.com/collective-action/tech</a>
266882295	<a href="https://github.com/undark-lab/swyft">https://github.com/undark-lab/swyft</a>
190487408	<a href="https://github.com/Synthetixio/SIPs">https://github.com/Synthetixio/SIPs</a>
142918121	<a href="https://github.com/batfish/pybatfish">https://github.com/batfish/pybatfish</a>
7228447	<a href="https://github.com/scikit-hep/iminuit">https://github.com/scikit-hep/iminuit</a>
43830532	<a href="https://github.com/dereneaton/ipyrad">https://github.com/dereneaton/ipyrad</a>

Table A.1: List of GitHub projects

## APPENDIX B

### LIST OF OTHER NOTEBOOKS

Table B.1 shows other computational notebook solutions.

<b>Id</b>	<b>URL Repository</b>
Google Collaboratory	<a href="https://research.google.com/colaboratory/">https://research.google.com/colaboratory/</a>
RMarkdown	<a href="https://rmarkdown.rstudio.com/">https://rmarkdown.rstudio.com/</a>
nteract	<a href="https://nteract.io/">https://nteract.io/</a>
Azure Notebooks	<a href="https://visualstudio.microsoft.com/pt-br/vs/features/notebooks-at-microsoft/">https://visualstudio.microsoft.com/pt-br/vs/features/notebooks-at-microsoft/</a>
Deepnote	<a href="https://deepnote.com/">https://deepnote.com/</a>
Apache Zeppelin	<a href="https://zeppelin.apache.org/">https://zeppelin.apache.org/</a>
Mode Notebooks	<a href="https://mode.com/notebooks/">https://mode.com/notebooks/</a>
JetBrains Datalore	<a href="https://datalore.jetbrains.com/">https://datalore.jetbrains.com/</a>
Nextjournal	<a href="https://nextjournal.com/">https://nextjournal.com/</a>
Count	<a href="https://count.co/">https://count.co/</a>
Hex	<a href="https://hex.tech/">https://hex.tech/</a>
Kaggle	<a href="https://www.kaggle.com/docs/notebooks">https://www.kaggle.com/docs/notebooks</a>
Databricks notebooks	<a href="https://databricks.com/product/collaborative-notebooks">https://databricks.com/product/collaborative-notebooks</a>
CoCalc	<a href="https://cocalc.com/features/jupyter-notebook">https://cocalc.com/features/jupyter-notebook</a>
Observable notebook	<a href="https://observablehq.com/">https://observablehq.com/</a>
Jupyterlite	<a href="https://jupyterlite.readthedocs.io/en/latest">https://jupyterlite.readthedocs.io/en/latest</a>

Table B.1: Other Notebooks List