DistilKaggle: A Distilled Dataset of Kaggle Jupyter Notebooks

Mojtaba Mostafavi
Department of Computer Engineering
Sharif University of Technology
m.mostafavi@sharif.edu

Arash Asgari
Department of Computer Engineering
Sharif University of Technology
ash.asgari@sharif.edu

Mohammad Abolnejadian
Department of Computer Engineering
Sharif University of Technology
mohammad.abolnejadian@sharif.edu

Abbas Heydarnoori
Department of Computer Science
Bowling Green State University
aheydar@bgsu.edu

***Abstract*—Jupyter notebooks have become indispensable tools for data analysis and processing in various domains. However, despite their widespread use, there is a notable research gap in understanding and analyzing the contents and code metrics of these notebooks. This gap is primarily attributed to the absence of datasets that encompass both Jupyter notebooks and extracted their code metrics. To address this limitation, we introduce DistilKaggle, a unique dataset specifically curated to facilitate research on code metrics in Jupyter notebooks, utilizing the Kaggle repository as a prime source. Through an extensive study, we identify thirty-four code metrics that significantly impact Jupyter notebook code quality. These features such as *lines of code cell, mean number of words in markdown cells, performance tier of developer, etc.* are crucial for understanding and improving the overall effectiveness of computational notebooks. The DistilKaggle dataset is derived from a vast collection of notebooks, and we present two distinct datasets: (i) Code Cells and markdown Cells Dataset which is presented in two CSV files, allowing for easy integration into researchers' workflows as dataframes. It provides a granular view of the content structure within 542,051 Jupyter notebooks, enabling detailed analysis of code and markdown cells; and (ii) Notebook Code Metrics Dataset focused on the identified code metrics of notebooks. Researchers can leverage this dataset to access Jupyter notebooks with specific code quality characteristics, surpassing the limitations of filters available on the Kaggle website. Furthermore, the reproducibility of the notebooks in our dataset is ensured through the code cells and markdown cells datasets, offering a reliable foundation for researchers to build upon. Given the substantial size of our datasets, less than 5 GB, it becomes an invaluable resource for the research community, surpassing the capabilities of individual Kaggle users to collect such extensive data. For accessibility and transparency, both the datasets https://doi.org/10.5281/zenodo.10317389 and the code https://github.com/theablemo/DistilKaggle utilized in crafting this dataset are publicly available.**

***Index Terms*—Open dataset, Kaggle, Jupyter notebooks, Code metrics, Code quality**

I. Introduction

Jupyter notebooks have emerged as the predominant coding environment for data scientists [1]. They offer numerous advantages over traditional software development environments, including seamless documentation, code sharing, result analysis, visual and intuitive code development, and the ability to compile, execute, and modify code cells without re-executing the entire notebook. The quality and comprehensibility of implementations in computational notebooks are pivotal for various purposes, such as education for imparting coding best practices to future data scientists [2], notebook reusability [3], and maintainability [4]–[6]. Recent studies have concentrated on enhancing the quality of Jupyter notebooks through improved documentation [5], [7]–[10], notebook structure [11]–[13], and managing notebook variants and revisions [13], [14]. Many of these applications and research use various datasets of computational notebooks.

Jupyter notebooks possess several features distinguishing them from other programming environments, such as interactive programming, code cells, markdown cells, the possibility of arranging cell executions in different orders within a notebook, and the availability of each cell's output right after the code, etc. In many cases, these features can overshadow script-based coding styles, such as independent Python scripts. For instance, in Jupyter notebooks, the result of each executed cell is attached to it, motivating developers to use commands to produce outputs. In addition to code cells, Jupyter notebooks include another type of cell called a markdown cell, where developers can explain each part of their code. This feature incentivizes developers to enhance the quality of their code documentation by utilizing HTML tags in markdown cells.

The widespread adoption of notebooks in recent years, along with their distinctive features, has motivated our commitment to delivering a comprehensive and current collection for researchers to utilize with enhanced efficiency and ease. This dataset comprises notebooks paired with features designed to assess code quality, as elaborated upon in this paper.

Quaranta et al. [15] introduced KGTorrent, a dataset comprising 248,761 Jupyter notebooks, totaling over 180GB in size. Despite its utility for the community, we identified several shortcomings in this dataset, leading us to present DistilKaggle. The motivations behind introducing DistilKaggle are outlined below:

1) The latest Jupyter notebook in the KGTorrent dataset dates back to October 2020, and the code provided by the authors to refresh the dataset no longer functions due to changes in Kaggle's notebook downloading policies. In contrast, the notebooks in this paper were published on the Kaggle platform between October 2020 and October 2023.
2) Downloading notebooks in a quantity comparable to the dataset presented here would take months due to Kaggle API's request rate limit. Our dataset offers a convenient access to a substantial volume of valuable data—Jupyter notebooks.
3) In contrast to the substantial 180GB size of the KG-Torrent dataset, our dataset has been streamlined to a manageable size of 3GB by organizing notebooks into two dataframes for code cells and markdown cells, enhancing download efficiency and facilitating a swift utilization. Despite this reduction, full reproducibility is maintained through the provided dataframes. Still, the complete notebooks dataset is provided upon request.
4) A common step in works on code quality involves extracting code quality metrics. After an extensive review of previous works, we identified 34 static code quality features, like lines of code cell, mean number of words in markdown cells, performance tier of developer. These metrics are extracted from all the notebooks in our dataset and presented in CSV format. We anticipate that these dataset metrics will significantly save researchers' time and effort. Additionally, for works requiring notebooks with specific characteristics, the features dataframe empowers researchers to create custom subset data by filtering based on their study's requirements.

In this paper, we collected a set of notebooks' features that either were proposed by previous studies for static code analysis or were presented as being effective for code quality by the studies focused on Jupyter notebooks. In the next step, we extracted these features from our notebooks' dataset. So, we made the following contributions:

- A dataset of Jupyter notebooks crawled from the Kaggle platform.
- Code for crawling and scrapping the notebooks.
- A dataset of features extracted from each notebook.
- Code for feature extraction.

II. Dataset

There are different platforms with large datasets of notebooks that provide a large amount of information about the notebooks and their creators. Each of these platforms also has a different level of popularity among scholars and data scientists, which may increase the validity of the studies that are conducted based on their data. Also, based on the aim of the studies, the researchers need a set of information that is only available on a small number of these platforms.

Kaggle, a subsidiary of Google, is an online community of data scientists and machine learning practitioners. Kaggle allows users to find and publish datasets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges. We chose the Kaggle platform as the source of the notebooks for this study for various reasons.

To the best of author's knowledge, DistilKaggle is one of the first Jupyter notebook metrics dataset on this scale and can be used for different data mining and other code analysis purposes.

In the following sections, we will explain the metrics, whose names may not inherently convey their nature, in detail and the methodology of obtaining them. All of the metrics, their abbreviation in DistilKaggle, and the paper(s) introduced them are presented in Table I.

A. Metrics of Notebooks

We discuss notebook metrics in two groups.

1) Notebook-Based Metrics: The first set of metrics are those that have been specifically introduced for Jupyter notebooks. In recent years, several articles have worked on the code quality of Jupyter notebooks, and various metrics have been introduced in them [2], [4], [5], [9], [10], [12], [12], [16]–[21]. After extracting all these metrics, we discussed with five notebook developers to identify the ones that are effective in code comprehension. Finally, out of 23 available metrics, 17 of them were selected, and we implemented appropriate algorithms to obtain all of them. These criteria are introduced Table I and some of them are described here:

- **Number of Visualization Data Type**: Considering that it is possible to display the output of each cell in notebooks by executing it, this metric shows the number of visual outputs, including all kinds of images and graphs.
- **Number of Executed Cells**: This attribute specifies the number of cells that have been executed in a notebook and have an execution order number.
- **H1, H2 and H3 headlines**: These metrics show the number of headlines that are written inside the markdowns and are usually used to express titles and subtitles.
- **Distribution of Markdowns**: For calculating the distribution of markdowns throughout a notebook, we suggest the inverse coefficient of variation. The coefficient of variation (CV), also known as relative standard deviation (RSD), is a statistical measure commonly used for comparing diversity in workgroups and is defined as the ratio of the standard deviation to the mean [22].

$$CV = \sigma/\mu \tag{1}$$

where:

$$\sigma = \sqrt{\sum (xi - \mu)^2/n} \tag{2}$$

where:
- n: the size of population

- *xi*: each value from the population

To calculate the distribution of markdown, we considered the number of words of each markdown as the size of it. After applying Equation 1 to the notebooks of the final dataset, we tested the results and evaluated the confidence of our outputs. For instance, the highest CV (the most unbalanced distribution) belongs to notebook P[1] and the lowest CV (the most balanced distribution) belongs to notebook Q[2].

- **Distribution of Imports**: We adopted a similar approach as the distribution of markdowns for assessing the distribution of the imports feature within Jupyter notebooks, leveraging the inverse coefficient of variation as a statistical measure.
- **Performance Tier**: This feature is present in the metadata of the Kaggle repository, which, based on specific rules, determines the level of expertise of notebook developers with one of the levels 0 to 4.
- **External API Popularity**: This metric assigns a number to each notebook that represents to what extent the APIs and libraries used in that notebook are popular [19]. The more its value, the more the APIs used in the notebook are frequently used by other developers which results in better CU based on previous studies [23]. In order to measure this criterion in notebooks: First, we counted the number of times each API was imported in our dataset and assigned a popularity score to each API based on the frequency at which they were used in the whole dataset. Then, for each notebook, we summed up the popularity score of APIs used in that notebook to obtain the External API Popularity [23] score for that notebook.

2) Script-Based Metrics: Given that each code cell in Jupyter notebooks is a regular Python script, many of the metrics for code scripts proposed by prior studies [23]–[28] are also applicable to code cells in Jupyter notebooks. Considering that a small percentage of notebooks use the concept of class and object orientation[3] and have a weaker structure, some metrics were ignored.

Script-based metrics including eight basic metrics, two complexity metrics, four Halstead metrics [29] and three readability metrics as introduced in Table I and some of them are explained here:

- **Cyclomatic Complexity**: Cyclomatic complexity assigns a number for code complexity based on code graph. It was first used by Mathias [27] in order to examine the underlying nature of code designed to study the process of program comprehension.
- **KLCID**: Kind of Line of Code Identifier Density (KLCID) is One of the complexity metrics which analyzes the cognitive complexity of program comprehension tasks. The KLCID is calculated by counting the lines of conceptually unique code and calculating the density of identifiers.

[1]P: https://www.kaggle.com/code/xiwuhan/jmtc-2nd-place-solution

[2]Q: https://www.kaggle.com/code/anokas/two-sigma-time-travel-eda

[3]Based on the statistics of our notebooks dataset, only 3.5% of notebooks defined classes to implement objects.

TABLE I
FEATURES EXTRACTED IN THIS STUDY

Feature	Abbreviation	Reference
Script-Based Metrics		
Basic Metrics		
Lines of Code	LOC	[9], [23], [24], [26]
Number of Blank Lines of Code	BLC	[23], [24], [26]
Lines of Comments	LOCom	[23], [24], [26]
Number of Comment Words	CW	[26]
Number of Statements	S	[23], [26]
Number of Parameters	P	[23], [26]
Number of User-Defined Functions	UDF	[16]
Complexity Metrics		
Cyclomatic Complexity	CyC	[16], [23], [24], [26]
Nested Block Depth	NBD	[23], [26]
Kind of Line of Code Identifier Density	KLCID	[26]
Halstead Metrics		
Number of Operands	OPRND	[23], [26], [29]
Number of Operators	OPRAT	[24], [26], [29]
Number of Unique Operands	UOPRND	[26], [29]
Number of Unique Operators	UOPRAT	[26], [29]
Readability Metrics		
Average Line Length of Code	ALLC	[24]
Number of Identifiers	ID	[24]
Average Number of Identifiers	AID	[24]
Notebook-Based Metrics		
Number of Code Cells	CC	[9], [30]
Mean Number of Lines in Code Cells	MeanLCC	[26]
Number of Imports	I	[26], [30]
Mean Number of Words in Markdown Cells	MeanWMC	[9]
Number of H1 tags in the Markdown	H1	[10], [30], [31]
Number of H2 tags in the Markdown	H2	[4], [10], [30]
Number of H3 tags in the Markdown	H3	[4], [10], [30]
Number of Markdown Cells	MC	[9], [21], [30]
Mean Number of Lines in Markdown Cells	MeanLMC	[16]
Number of Markdown Words	MW	[5], [26]
Number of Lines in Markdown Cells	LMC	[9], [23]
Distribution of Markdown Cells	DMC	[9], [30]
Distribution of Imports	DI	[9], [30]
Performance Tier	PT	[23], [32]
External API Popularity	EAP	[19], [20], [23]
Number of Visualization Data Type	VDT	[17], [33]
Number of Executed Cells	EC	[30]

B. Dataset Construction Methodology

Our journey in generating the dataset began by obtaining the paper IDs shared on Meta Kaggle API between October

2020 and October 2023. Due to Kaggle's platform policy, each machine can download limited number of notebooks per day. To overcome this limitation, we utilized multiple servers with various IP addresses, enabling the download of 293,290 notebooks from the Kaggle Platform. The response payload containing the notebooks is stored in JSON format, with a total size exceeding 300GB.

In the subsequent step, we organized the code cells and markdown cells into two distinct DataFrames (CSV files). This not only reduced the overall size of the dataset to 3GB but also enhanced its usability for future data analysis by fellow researchers. These refined DataFrames form an integral part of our final dataset, DistilKaggle. The detailed description of the columns of the Dataframes is presented in Table II.

TABLE II
COLUMNS OF THE DATAFRAMES

Column	Description
Kernel ID	This key represents the unique identifier assigned to the Kaggle project associated with the Jupyter notebook.
Cell Index	The Cell Index indicates the sequential order of the cell within the Jupyter notebook.
Source	The Source key contains the actual text written in the cell.
Output Type	The Output Type key signifies the type of output generated by the execution of a code cell in the Jupyter notebook. It can take on various values, including:stream, display_data, error, execute_result. This column only exists in code cells Dataframe.
Execution Count	The Execution Count helps in tracking the order of code execution and understanding the flow of computations within the notebook. This column only exists in code cells Dataframe.

Following a comprehensive investigation into code quality metrics utilized in prior studies to evaluate code quality and comprehensibility, we employed the markdowns and code cells dataframes to extract these metrics for each Jupyter notebook. In many instances, utilizing either the code cells or markdown cells proved sufficient for capturing these features. Certain features, such as the performance tier indicating the developer's expertise level, were extracted using the Meta Kaggle dataset [34].

Finally, as Figure 1 shows, the resulting features dataset is compact at 121MB, yet it offers invaluable insights into the code quality of the notebooks.

III. Sample Application

The sheer magnitude of this dataset renders it versatile for diverse code analysis tasks. Many prior studies have concentrated on utilizing Jupyter notebooks for statistical analysis [2], [4], [18], [20], [30] and implementing deep learning models [5], [9], [10], [21], underscoring our dataset's potential for various research applications.

To demonstrate the dataset's utility, we employed a Random Forest classifier to predict the performance tier of notebook

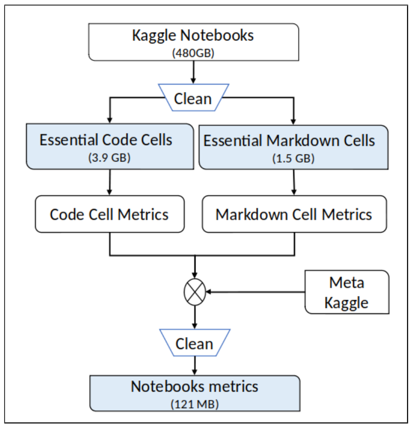

Fig. 1. The approach to creating our 3 datasets

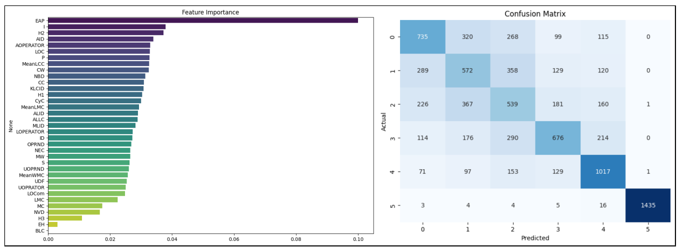

Fig. 2. (left) The influence of individual features on the classifier's prediction output (performance tier) is presented. Notably, API popularity emerged as the most decisive factor in predicting the performance tier of the notebook creator. (right) Confusion matrix results for the Random Forest Classifier trained with 100 trees, utilizing 80% of the dataset for training.

developers based on code quality features. We utilized 80% of the DistilKaggle as our training set and the rest as the test set. The achieved F1 score of 57% significantly outperformed a baseline Random predictor, which would have an accuracy of 20%. This stark contrast emphasizes distinct coding styles between experienced and novice programmers, prompting further exploration in a more targeted manner. Figure 2 presents the performance of the Random Forest classifier. Our findings open avenues for in-depth investigations into the nuances of coding practices across different expertise levels.

IV. Conclusion

In this study, we introduced DistilKaggle, an extensive dataset comprising 293,290 Jupyter notebooks, each accompanied by its corresponding static code quality metrics. Our vision for DistilKaggle is to serve as a catalyst for future research endeavors in the realm of code analysis. The substantial volume of notebooks positions it as an ideal resource for the development and evaluation of large-scale deep learning models.

Notably, we enriched DistilKaggle by incorporating code quality metrics from the KGTorrent dataset, creating a combined dataset totaling 542,051 notebooks. In this process, we extracted code quality metrics of these notebooks, amplifying the depth and richness of our dataset.

References

[1] Fotis Psallidas, Yiwen Zhu, Bojan Karlas, Jordan Henkel, Matteo Interlandi, Subru Krishnan, Brian Kroth, Venkatesh Emani, Wentao Wu, Ce Zhang, and et al. Data science through the looking glass. *ACM SIGMOD Record*, 51(2):30–37, 2022.

[2] Jiawei Wang, Li Li, and Andreas Zeller. Better code, better sharing: on the need of analyzing jupyter notebooks. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, pages 53–56, 2020.

[3] Will Epperson, April Yi Wang, Robert DeLine, and Steven M Drucker. Strategies for reuse and sharing among data scientists in software teams. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*, pages 243–252, 2022.

[4] Jiawei Wang, KUO Tzu-Yang, Li Li, and Andreas Zeller. Assessing and restoring reproducibility of jupyter notebooks. In *2020 35th IEEE/ACM international conference on automated software engineering (ASE)*, pages 138–149. IEEE, 2020.

[5] Chenyang Yang, Shurui Zhou, Jin LC Guo, and Christian Kästner. Subtle bugs everywhere: Generating documentation for data wrangling code. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 304–316. IEEE, 2021.

[6] Helen Dong, Shurui Zhou, Jin LC Guo, and Christian Kästner. Splitting, renaming, removing: A study of common cleaning activities in jupyter notebooks. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*, pages 114–119. IEEE, 2021.

[7] Jiali Liu, Nadia Boukhelifa, and James R Eagan. Understanding the role of alternatives in data analysis practices. *IEEE transactions on visualization and computer graphics*, 26(1):66–76, 2019.

[8] Adam Rule, Aurélien Tabard, and James D Hollan. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018.

[9] April Yi Wang, Dakuo Wang, Jaimie Drozdal, Michael Muller, Soya Park, Justin D Weisz, Xuye Liu, Lingfei Wu, and Casey Dugan. Documentation matters: Human-centered ai system to assist data science code documentation in computational notebooks. *ACM Transactions on Computer-Human Interaction*, 29(2):1–33, 2022.

[10] Ashwin Prasad Shivarpatna Venkatesh, Jiawei Wang, Li Li, and Eric Bodden. Enhancing comprehension and navigation in jupyter notebooks with static analysis. 2023.

[11] John Wenskovitch, Jian Zhao, Scott Carter, Matthew Cooper, and Chris North. Albireo: An interactive tool for visually summarizing computational notebook structure. In *2019 IEEE visualization in data science (VDS)*, pages 1–10. IEEE, 2019.

[12] Sergey Titov, Yaroslav Golubev, and Timofey Bryksin. Resplit: improving the structure of jupyter notebooks by re-splitting their cells. In *2022 IEEE international conference on software analysis, evolution and reengineering (SANER)*, pages 492–496. IEEE, 2022.

[13] Yuan Jiang, Christian Kästner, and Shurui Zhou. Elevating jupyter notebook maintenance tooling by identifying and extracting notebook structures. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 399–403. IEEE, 2022.

[14] Krishna Subramanian, Ilya Zubarev, Simon Völker, and Jan Borchers. Supporting data workers to perform exploratory programming. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–6, 2019.

[15] Luigi Quaranta, Fabio Calefato, and Filippo Lanubile. Kgtorrent: A dataset of python jupyter notebooks from kaggle. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 550–554. IEEE, 2021.

[16] Konstantin Grotov, Sergey Titov, Vladimir Sotnikov, Yaroslav Golubev, and Timofey Bryksin. A large-scale comparison of python code in jupyter notebooks and scripts. In *Proceedings of the 19th International Conference on Mining Software Repositories*, pages 353–364, 2022.

[17] Jorge Piazentin Ono, Juliana Freire, and Claudio T Silva. Interactive data visualization in jupyter notebooks. *Computing in Science & Engineering*, 23(2):99–106, 2021.

[18] Jiawei Wang, Li Li, and Andreas Zeller. Restoring execution environments of jupyter notebooks. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1622–1633. IEEE, 2021.

[19] Chenguang Zhu, Ripon K Saha, Mukul R Prasad, and Sarfraz Khurshid. Restoring the executability of jupyter notebooks by automatic upgrade of deprecated apis. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 240–252. IEEE, 2021.

[20] Malinda Dilhara, Ameya Ketkar, and Danny Dig. Understanding software-2.0: a study of machine learning library usage and evolution. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(4):1–42, 2021.

[21] Xuye Liu, Dakuo Wang, April Wang, Yufang Hou, and Lingfei Wu. Haconvgnn: Hierarchical attention based convolutional graph neural network for code documentation generation in jupyter notebooks. *Findings of the Association for Computational Linguistics: EMNLP 2021*, 2021.

[22] Arthur G Bedeian and Kevin W Mossholder. On the use of the coefficient of variation as a measure of diversity. *Organizational Research Methods*, 3(3):285–297, 2000.

[23] Simone Scalabrino, Gabriele Bavota, Christopher Vendome, Mario Linares-Vasquez, Denys Poshyvanyk, and Rocco Oliveto. Automatically assessing code understandability. *IEEE Transactions on Software Engineering*, 47(3):595–613, 2019.

[24] Raymond PL Buse and Westley R Weimer. Learning a metric for code readability. *IEEE Transactions on software engineering*, 36(4):546–558, 2009.

[25] Jonathan Dorn. A general software readability model. *MCS Thesis available from (https://web.eecs.umich.edu/ weimerw/students/dorn-mcs-pres.pdf)*, 5:11–14, 2012.

[26] Nadia Kasto and Jacqueline Whalley. Measuring the difficulty of code comprehension tasks using software metrics. In *Proceedings of the Fifteenth Australasian Computing Education Conference-Volume 136*, pages 59–65, 2013.

[27] Karl S Mathias, James H Cross, T Dean Hendrix, and Larry A Barowski. The role of software measures and metrics in studies of program comprehension. In *Proceedings of the 37th annual Southeast regional conference (CD-ROM)*, pages 13–es, 1999.

[28] JR Parker and Katrin Becker. Measuring effectiveness of constructivist and behaviourist assignments in cs102. *ACM SIGCSE Bulletin*, 35(3):40–44, 2003.

[29] Daryl Posnett, Abram Hindle, and Premkumar Devanbu. A simpler model of software readability. In *Proceedings of the 8th working conference on mining software repositories*, pages 73–82, 2011.

[30] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. A large-scale study about quality and reproducibility of jupyter notebooks. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 507–517. IEEE, 2019.

[31] Ashwin Prasad Shivarpatna Venkatesh and Eric Bodden. Automated cell header generator for jupyter notebooks. In *Proceedings of the 1st ACM International Workshop on AI and Software Testing/Analysis*, pages 17–20, 2021.

[32] Eman Abdullah AlOmar, Anthony Peruma, Mohamed Wiem Mkaouer, Christian D Newman, and Ali Ouni. Behind the scenes: On the relationship between developer experience and refactoring. *Journal of Software: Evolution and Process*, page e2395, 2021.

[33] Vishakha Agrawal, Yong-Han Lin, and Jinghui Cheng. Understanding the characteristics of visual contents in open source issue discussions: a case study of jupyter notebook. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022*, pages 249–254, 2022.

[34] Kaggle. Kaggle meta data, https://www.kaggle.com/datasets/kaggle/meta-kaggle, 2020.