Casual Notebooks and Rigid Scripts: Understanding Data Science Programming

Krishna Subramanian, Nur Hamdan, Jan Borchers
RWTH Aachen University
52074 Aachen, Germany
{krishna, hamdan, borchers}@cs.rwth-aachen.de

***Abstract*—Data workers are non-professional data scientists who often use scripting languages like R, Python, or MATLAB, and employ an exploratory programming workflow. Current IDEs offer them two main programming modalities: script files and computational notebooks. To understand how these modalities impact work practice, we conducted a study with 21 data workers, and a subsequent larger survey with 62 respondents. Through interviews, walkthroughs, and screen recordings, we collected information about their workflows. Our analysis shows a tension between scripts and computational notebooks. Scripts are more common, better support storage and execution of previous analyses, but hinder experimentation. Notebooks better suit the actual data science workflow, but can become easily unorganized. We discuss how this dual nature of modality usage leads to several issues that affect data workers' workflows, and discuss implications for the design of programming IDEs.**

***Index Terms*—scripting languages, exploratory programming, programming interfaces, data science, notebooks**

I. Introduction

Data scientists perform tasks, such as machine learning, significance testing, and regression analysis, to extract "information and knowledge from data" [1]. Many data scientists are **data workers,** who are usually from academia and may not be formal trained in data science or programming [2]. Scripting languages, such as R, Python, and MATLAB, are popular among data workers [3], [4]. Today, the IDEs for these languages allow data workers to write and execute source code using two main programming modalities: **scripts** and **computational notebooks**. We also investigated consoles, but found that it is not used frequently. When using these modalities, data workers employ a programming style known as **exploratory programming** [5], characterized by open-ended exploration of alternatives and the evolution of the programmer's goals during programming [6].

Our key goal with this research is to understand how these modalities support data workers' workflows. To achieve this, we collected data from 21 data workers through interviews, walkthroughs, and screen recordings. In this paper, we present and discuss our findings about how the nature of data workers' work affects how programming modalities are used in various data science tasks. We support our findings using the data collected from our participants, results of an online survey, and Green's cognitive framework for programming [7].

978-1-5386-5541-2/18/$31.00 ©2020 IEEE

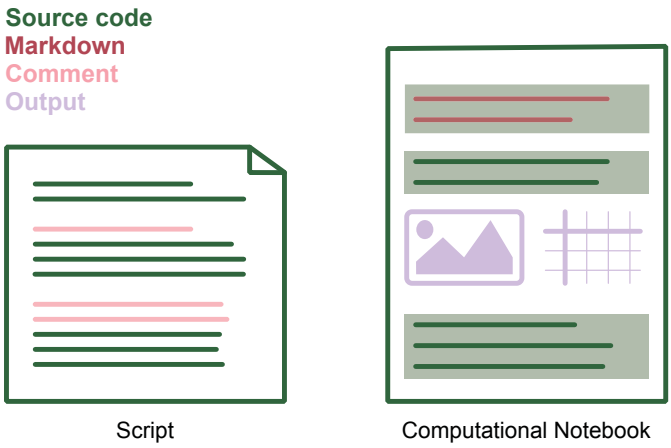

Fig. 1. Current scripting language IDEs support writing and executing code via two programming *modalities*: *scripts* (left) and *computational notebooks* (right). In this paper, we investigate how these modalities are used in data science programming.

II. Background and Related Work

We first describe how scripting language IDEs and their modalities work, and then discuss prior research about data science programming practices.

A. Scripting language IDEs

IDEs of R [8], Python [9], and MATLAB [10] are prevalent in data science [3], [4]. Most of these IDEs support both scripts and computational notebooks; some offer support via plugins. The most popular notebook environments for the three languages are RMarkdown, Jupyter, and MATLAB Live Editor. In addition to IDEs, scripts can also be written in text editors.

Scripts (Fig. 1, left) support conventional storage and execution of source code. All IDEs support execution of entire scripts, with most also allowing selective execution of snippets. When scripts are executed, text output and error messages are shown in the console, while graphic output is usually shown in a separate dedicated window.

Computational notebooks (Fig. 1, right) let users organize their code into cells. A cell typically represents one computational chunk of the data science task, and can be executed in a non-sequential order. Unlike scripts, execution output

is juxtaposed with the cell. Additionally, notebooks support narration via Markdown; these are interwoven with the code.

B. Data science and exploratory programming

Many data workers use programming for data science tasks. Prior research has studied data scientists' exploratory programming practice. We found three research works that investigated data scientists' notebook use [11]–[13]. Prominent issues that were identified include the messy nature of notebooks, difficulties using versioning control, and difficulties finding previous code.

Unlike notebooks, which have been used almost exclusively for data science, scripts were originally used for web development and automation [14]. Only with the recent performance bumps have data scientists begun using them [15]. Research by Guo [16] and Kery et al. [17] present findings about data science workflow in general, without differentiating between programming modalities. For example, Kery et al. discuss issues with the use of version control and code hoarding practices among data scientists [17]. This paper is the first to compare how the two main programming modalities are used for different data science tasks.

III. Method

We collected data from 21 data workers (9 female, median age of 27). Participants include 13 researchers from different fields such as Psychology, Human-Computer Interaction, and Electrical Engineering, and 8 graduate students. All participants reported using at least one modality in their work. After obtaining participants' background details through interviews, we asked them to walk us through their recent data science projects. This helped us understand our participants' workflow with the programming IDE. After walkthroughs, we wanted to observe participants, either remotely or in-person, perform real-world tasks. Most participants felt this was intrusive, and provided a screen recording of their work instead. We collected audio and video logs of the interview and walkthroughs, and screen capture of observations. For details of our participants and the data we collected, see https://hci.rwth-aachen.de/modality-use.

To analyze our data, we followed the Constant Comparative Method [18], [19] of the Grounded Theory methodology [20], [21]. Details of our method, including the code book can be found on https://hci.rwth-aachen.de/modality-use. In addition to interviews and observations, we also conducted an online survey with 62 data workers to substantiate our findings about modality usage for data science programming, discussed in the next section.

IV. How Data Workers Use Programming Modalities

Before looking at how data workers use scripts and notebooks, we need to understand what the different tasks in data science programming are, and what characteristics programming IDEs need to have to support these tasks. Given below are four important tasks in data science programming and the corresponding characteristics required by the modalities. These tasks are adopted from Guo's work [16] and grounded in our analysis. The characteristics are based on Green's cognitive dimensions [7]. For details, see https://hci.rwth-aachen.de/modality-use.

1) **Experimentation:** The main task of data workers during experimentation is to create and test new approaches. Since new code is typically written based on existing code [17], the interface should help users **find existing code.** Per Green's cognitive dimensions [7], this requires a high-level of **role-expressiveness**, i.e., help the user know the functionality of a piece of code.
2) **Compare execution results of approaches:** During experimentation, data workers compare their experiments and select an experiment to refine. To compare execution results, data workers need to be able to (1) locate the source code that belongs to an approach, i.e., high role-expressiveness, (2) map the output to the source code that generated it, and (3) view execution outputs and source code simultaneously. To support (2), the proximity of output to source code is important, a concept we term **'code-result distance'.** A high-distance means that the execution output is located farther away from the source code that generated it, and vice versa. (3) can be achieved with a high-level of juxtaposability [7].
3) **Curate source code:** When refining code, data workers need to understand their code so that they can remove the scratchpad code. This would require high role-expressiveness and low code-result distance. Alternatively, the programming interface could support secondary notations like tags and programmer's comments to aid this task. Removing source code should also be done without breaking any **hidden dependencies** [7].
4) **Present analysis and help reuse:** To present the analysis and results to stakeholders or in publications, the ability to add narrative and reproduce expected results from source code is desired. Reproducibility can be achieved by minimizing hidden dependencies.

Do scripts and computational notebooks co-exist in harmony, or in discord? How do data workers utilize them at various stages of data science? We address these questions below.

A. Scripts and notebooks for experimentation and comparing experiments

Among our participants, many (P01–04, P06, P07, P09, P11, P12, and P18–21) do not use computational notebooks frequently in their work. However, of the remaining participants who use both computational notebooks and scripts, most (P05, P08, P13, and P15–17) prefer notebooks for experimentation. This is not surprising as scripts are not well suited to exploratory work. They have low code-result distance and juxtaposability. When a code snippet is executed, the execution output is displayed on the console or stored in the file system; this can deter the data workers' productivity during

experimentation. Also, the console can get cluttered over a period of time, further exacerbating this problem.

Notebooks, on the other hand, better suit experimentation. Our participants provided two main reasons for this. First, notebooks allow organizing source code into manageable chunks, allowing easier experiments without having to use messy program structures like comments:

> "(On notebooks) I don't have to worry about the other code for now. I can focus on this [chunk of] code." - P15

Second, notebooks offer a low code-result distance, allowing data workers to easily map the execution output to the code cells that generated them. For smaller code cells, the juxtaposability is also improved, allowing data workers to view multiple code cell-execution output pairs at the same time.

B. Scripts and notebooks for code refinement

Our participants performed two broad tasks when refining source code: (a) refactor and migrate code, which may require pruning the scratchpad code; and (b) add documentation and narratives. Scripts were mostly used to perform (a). This is because scripts are often used in IDEs like PyCharm and Visual Studio, and therefore offer powerful code refactoring functionalities, e.g., [22]. Notebooks, on the other hand, are mostly run on the web to facilitate easier sharing, and offer little to no code refactoring functionality.

Larger notebooks have been known to get unstructured and unmanageable [11]; instead, several scripts, where each script acts as a "black box" code package could be a simpler alternative. This also helps explain why notebooks are not typically used to write software packages that contain large code base. Despite these limitations, participants found notebooks to be indispensable, especially when they need to present or share their work.

C. Scripts and notebooks for dissemination and storage

Most participants (P01-04, P07-10, P12, P20, and P21) use scripts for dissemination and storage. Scripts have better support for use in the production pipeline. E.g., P08 and P12 work on projects with a large code base that is already organized as multiple scripts; it is easier to add new source code as scripts. For some participants (P08, P15, P16, and P17), scripts are indispensable because of what they offer: Execution from the command shell, better support for file stream access, possibility of automation, etc.:

> "Compute cluster accepts only scripts. IPythons can be run as well, but it is frowned upon [by those who maintain the compute cluster] because it blocks computing resources." - P17

Computational notebooks were used by some participants (P05, P11, P13, P16, and P17) mainly for dissemination, e.g., stakeholders, a colleague, or blog readers. However, none of our participants reported using computational notebooks exclusively.

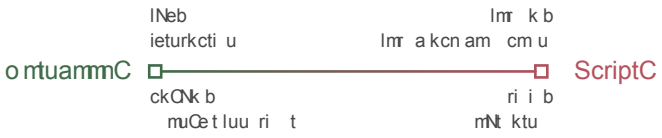

Fig. 2. Terms our participants used to refer to computational notebooks and scripts. Notebooks were considered to be interactive and fun, but too casual, whereas scripts were considered formal but too rigid.

Many data workers kick-start their analysis by basing it on existing code [16]. Several participants (P08, P13, P14, P15, and P17) reported using scratchpads notebooks, i.e., notebooks used for typically unstructured, exploratory work, for this purpose. These notebooks can also help data workers get reacquainted with API usage and programming language syntax, Some participants (P08, P14, and P15) considered these scratchpad notebooks to be even more important than the source code files that contain the production code, as it helps them reason about the execution output better.

D. Data workers' perceptions of notebooks and scripts

Participants associated scripts with terms like **"black box," "formal,"** and **"reliable,",** but also **"rigid"** and **"outdated"** as shown in Fig. 2. Notebooks, on the other hand, were associated with **"interactive"** and **"fun,"** but also **"too casual"** and **"doesn't feel right".** As we discussed earlier, scripts were considered the main modality to store production code for dissemination:

> "For me, when I do analysis and [write production] code, I want to make it formal. [...] it makes sense to have my source code in scripts." - P15

Scripts, unlike computational notebooks, allow data workers to view source code output without having to step-through the source code cell-by-cell. This high code-result distance or the ability to decouple the results away from the code can be desirable if the data worker just wants to obtain the results without having to step through the code, one chunk at a time. Conversely, the higher-level of interactivity offered by notebooks via low code-result distance and cell-by-cell execution acts as an inconvenience when reusing code:

> "I have tried writing the final version of my code blocks in Python notebooks, but it just doesn't feel right...Once something is a black box, it should belong in scripts. I don't want to run it in a notebook anymore because it runs through line-by-line and I don't want that. It doesn't feel clean." - P17

Also, support for computational notebooks in production pipeline is still a work in progress. E.g., for P17, a data worker who builds neural networks, a standard practice in his field is to outsource computationally-intensive executions to external GPUs. These GPUs support scripts in the standard *.py file format, and not Python notebooks, making it difficult for P17 to adopt computational notebooks.

There is thus a **tension** between the two main modalities, computational notebooks and scripts (Fig. 2). The interactivity of computational notebooks is great for experimentation, but scripts' are a more reliable medium for reuse and storage.

E. Prevalence of computational notebooks and scripts

Since a significant number of our participants do not use notebooks, we wanted to extrapolate the usage statistics of notebooks and scripts to a larger sample. To do so, we conducted an online survey with 62 data workers from various domains like machine learning and significance testing. Respondents were recruited via word of mouth and social media. Respondents self-reported an average expertise of 3.51 (1 = novice, 5 = expert). Combining the survey results with data from our interview participants (n = 21), we were able to gather modality usage statistics for 83 data workers as shown in Fig. 3.

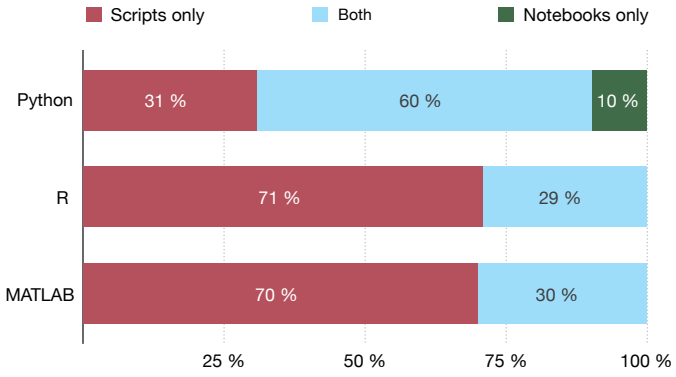

Fig. 3. Results of our online survey where respondents chose the programming modalities they use for data science programming. Scripts are the most commonly used modality across all programming languages, with many respondents using a combination of scripts and computational notebooks.

Scripts are the most commonly used programming modality across R, Python, and MATLAB. Only five respondents use notebooks exclusively, all for Python programming. Conversely, 40 respondents use scripts exclusively. Most respondents (n = 41) use **both** computational notebooks and scripts. Computational notebooks are more popular among Python programmers (36 out of 53 respondents; 67.9%) than MATLAB (3 out of 11 respondents; 27.3%) and R programmers (7 out of 25 respondents; 28%). This indicates that Python notebooks are more popular than the MATLAB Live Editor and RMarkdown notebooks, but there may be other explanations. E.g., notebooks are more popular in the machine learning community, where Python is commonly used.

To investigate why scripts are more common than computational notebooks, we returned to our interview transcripts. Twelve participants (P01–04, P06, P07, P09, P11, P12, and P18–21) who do not use computational notebooks regularly had reasons that varied from not being aware of their existence to finding them unnecessary. Some reported that they had tried to use notebooks earlier, but did not gain much benefit. There is no clear monopoly of modalities; scripts are still the most common modality in data science, but many data workers use notebooks, often in combination with scripts.

TABLE I
TRADE-OFFS BETWEEN THE TWO COMMON MODALITIES OF DATA SCIENCE PROGRAMMING. ✓ INDICATES THE PREFERENCE OF MOST OF OUR PARTICIPANTS.

Data science task	Notebooks	Scripts
Experimentation	✓	–
Refactor code	–	✓
Large data science project	–	✓
Present code	✓	–
Share code	✓	–
Execute from command line or GPU	–	✓
Store code	–	✓
Re-run past code	–	✓

V. OPPORTUNITIES FOR DESIGN AND CONCLUSION

Our findings from the previous section show trade-offs between scripts and notebooks. Despite their share of issues, notebooks are gaining traction among data workers, and researchers are constantly working to improve their interaction design, e.g., with [13], [23] and [24]. It is also reasonable to expect computational notebooks to have better support for use in production pipeline in future.

Here are some ways to redesign notebooks to bridge the difference to scripts: a) **lock** code cells so that they are immune to further changes once the data worker finishes experimentation, b) **merge** code cells after they reach a level of maturity to help avoid executing small code cells one at a time, and c) allow users to **switch** between experimentation and refinement modes; in experimentation mode, cell-based programming is active, but in the refinement mode, cells of flattened into scripts—users can switch between the modes as desired. However, given how prevalent scripts currently are, these are long-term solutions. For design recommendations that can resolve problems in the short-term, and our design recommendations for other issues discussed in this paper, see http://hci.rwth-aachen.de/modality-use.

In summary, good programmers find notebooks "too casual" since it promotes writing unstructured, less-modular code; existing notebook users find scripts rigid and unyielding for experimentation. Issues arise when data workers try to repurpose notebooks to write and store production code, and use scripts for experimentation, both of which are not ideal. Given this dual nature of modality use, how can we (a) design notebooks to support storage and use in production pipeline, (b) help data workers who are also good programmers "ease in" to notebooks, and c) design an environment where data workers can seamlessly switch between experimentation, code refinement, and presentation? We hope that the insights presented in this paper can inform future designs and research in data science IDEs.

ACKNOWLEDGMENT

We thank the reviewers for their valuable insights; our study participants for their time and input; and Oliver Nowak and Sarah Suleri for their feedback on a draft of this paper. This research is funded in part by the German B-IT foundation.

REFERENCES

[1] F. Provost and T. Fawcett, "Data Science and Its Relationship to Big Data and Data-Driven Decision Making," *Big Data*, vol. 1, pp. 51–59, 2013. [Online]. Available: https://doi.org/10.1089/big.2013.1508

[2] N. Boukhelifa, M.-E. Perrin, S. Huron, and J. Eagan, "How Data Workers Cope with Uncertainty: A Task Characterisation Study," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, ser. CHI 17. New York, NY, USA: Association for Computing Machinery, 2017, p. 36453656. [Online]. Available: https://doi.org/10.1145/3025453.3025738

[3] N. Dopico. (2017, May) Data Science Tools - A Survey: Post 1. [Online]. Available: https://www.keedio.org/data-science-tools-a-survey-post-1/

[4] P. Prabhu, T. B. Jablin, A. Raman, Y. Zhang, J. Huang, H. Kim, N. P. Johnson, F. Liu, S. Ghosh, S. Beard, T. Oh, M. Zoufaly, D. Walker, and D. I. August, "A survey of the practice of computational science," in *State of the Practice Reports*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 19:1–19:12. [Online]. Available: http://doi.acm.org/10.1145/2063348.2063374

[5] B. Sheil, "Environments for Exploratory Programming," *Datamation*, vol. 29, no. 7, pp. 131–144, 1983.

[6] M. B. Kery and B. A. Myers, "Exploring Exploratory Programming," in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, ser. VL/HCC '17. IEEE, Oct 2017, pp. 25–29. [Online]. Available: https://doi.org/10.1109/VLHCC.2017.8103446

[7] T. R. G. Green and M. Petre, "Usability Analysis of Visual Programming Environments: A Cognitive Dimensions Framework," *Journal of Visual Languages & Computing*, vol. 7, no. 2, pp. 131–174, 1996.

[8] R. Ihaka and R. Gentleman, "R: A Language for Data Analysis and Graphics," *Journal of Computational and Graphical Statistics*, vol. 5, no. 3, pp. 299–314, 1996.

[9] H. P. Langtangen, Ed., *Python Scripting for Computational Science*, ser. Texts in Computational Science and Engineering. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, vol. 3.

[10] C. B. Moler, *Numerical Computing With MATLAB: Revised Reprint*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2008, vol. 87.

[11] M. B. Kery, M. Radensky, M. Arya, B. E. John, and B. A. Myers, "The Story in the Notebook: Exploratory Data Science Using a Literate Programming Tool," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, ser. CHI '18. New York, NY, USA: ACM, 2018, pp. 174:1–174:11. [Online]. Available: http://doi.acm.org/10.1145/3173574.3173748

[12] A. Rule, A. Tabard, and J. D. Hollan, "Exploration and explanation in computational notebooks," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, ser. CHI '18. New York, NY, USA: ACM, 2018, pp. 32:1–32:12. [Online]. Available: http://doi.acm.org/10.1145/3173574.3173606

[13] A. Head, F. Hohman, T. Barik, S. M. Drucker, and R. DeLine, "Managing messes in computational notebooks," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19. New York, NY, USA: ACM, 2019, pp. 270:1–270:12. [Online]. Available: http://doi.acm.org/10.1145/3290605.3300500

[14] J. Vitek, "Of Scripts and Programs: Tall Tales, Urban Legends, and Future Prospects," *SIGPLAN Not.*, vol. 44, no. 12, p. 12, Oct. 2009. [Online]. Available: https://doi.org/10.1145/1837513.1640136

[15] H. P. Langtangen, "Texts in Computational Science and Engineering," 2006.

[16] P. J. Guo, "Software Tools to Facilitate Research Programming," Ph.D. dissertation, Stanford, CA, USA, 2012.

[17] M. B. Kery, A. Horvath, and B. Myers, "Variolite: Supporting exploratory programming by data scientists," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, ser. CHI '17. New York, NY, USA: ACM, 2017, pp. 1265–1276. [Online]. Available: http://doi.acm.org/10.1145/3025453.3025626

[18] H. Boeije, "A Purposeful Approach to the Constant Comparative Method in the Analysis of Qualitative Interviews," *Quality and Quantity*, vol. 36, no. 4, pp. 391–409, Nov 2002. [Online]. Available: https://doi.org/10.1023/A:1020909529486

[19] B. G. Glaser and A. L. Strauss, *Discovery of Grounded Theory: Strategies for Qualitative Research*. USA: Routledge, 2017.

[20] J. A. Smith, R. Harre, and L. v. Langenhove, *Rethinking Methods in Psychology*. London, UK: SAGE Publications Ltd, 1995.

[21] A. Strauss and J. Corbin, "Grounded Theory Methodology: An Overview," *Handbook of Qualitative Research*, vol. 17, pp. 273–85, 1994.

[22] D. Filippov. (2015, February) Feature Spotlight: Refactoring Python Code. [Online]. Available: https://blog.jetbrains.com/pycharm/2015/02/feature-spotlight-refactoring-python-code/

[23] M. B. Kery, B. E. John, P. O'Flaherty, A. Horvath, and B. A. Myers, "Towards Effective Foraging by Data Scientists to Find Past Analysis Choices," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, 2019, p. 92. [Online]. Available: https://doi.org/10.1145/3290605.3300322

[24] A. Rule, I. Drosos, A. Tabard, and J. D. Hollan, "Aiding Collaborative Reuse of Computational Notebooks with Annotated Cell Folding," *Proc. ACM Hum.-Comput. Interact.*, vol. 2, no. CSCW, Nov. 2018. [Online]. Available: https://doi.org/10.1145/3274419