What's Wrong with Computational Notebooks? Pain Points, Needs, and Design Opportunities

Souti Chattopadhyay[1], **Ishita Prasad**[2], **Austin Z. Henley**[3], **Anita Sarma**[1], **Titus Barik**[2]
Oregon State University[1], Microsoft[2], University of Tennessee-Knoxville[3]
{chattops, anita.sarma}@oregonstate.edu, {ishita.prasad, titus.barik}@microsoft.com, azh@utk.edu

ABSTRACT

Computational notebooks—such as Azure, Databricks, and Jupyter—are a popular, interactive paradigm for data scientists to author code, analyze data, and interleave visualizations, all within a single document. Nevertheless, as data scientists incorporate more of their activities into notebooks, they encounter unexpected difficulties, or pain points, that impact their productivity and disrupt their workflow. Through a systematic, mixed-methods study using semi-structured interviews ($n = 20$) and survey ($n = 156$) with data scientists, we catalog nine pain points when working with notebooks. Our findings suggest that data scientists face numerous pain points throughout the entire workflow—from setting up notebooks to deploying to production—across many notebook environments. Our data scientists report essential notebook requirements, such as supporting data exploration and visualization. The results of our study inform and inspire the design of computational notebooks.

Author Keywords

Computational notebooks; challenges; data science; interviews; pain points; survey

CCS Concepts

•Human-centered computing → Interactive systems and tools; Empirical studies in HCI; •Software and its engineering → *Development frameworks and environments;*

INTRODUCTION

Computational notebooks are an interactive paradigm for combining code, data, visualizations, and other artifacts, all within a single document [21, 36, 32, 30]. This interface, essentially, is organized as a collection of input and output *cells*. For example, a data scientist might write Python code in an input code cell, whose result renders a plot in an output cell. Although these cells are linearly arranged, they can be reorganized or executed in any order. The code executes in a *kernel*—the computational engine behind the notebook.

This interactive paradigm has made notebooks an appealing choice for data scientists, and this demand has sparked multiple open source and commercial implementations, including Azure,[1] Databricks,[2] Colab,[3] Jupyter,[4] and nteract.[5] While originally intended for exploring and constructing computational narratives [29, 31], data scientists are now increasingly orchestrating more of their activities within this paradigm [33]: through long-running statistical models, transforming data at scale, collaborating with others, and executing notebooks directly in production pipelines. But as data scientists try to do so, they encounter unexpected difficulties—pain points—from limitations in affordances and features in the notebooks, which impact their productivity and disrupt their workflow.

To investigate the pain points and needs of data scientists who work in computational notebooks, across multiple notebook environments, we conducted a systematic mixed-method study using field observations, semi-structured interviews, and a confirmation survey with data science practitioners. While prior work has studied specific facets of difficulties in notebooks [24, 17], such as versioning [18, 19] or cleaning unused code [13, 34], the central contribution of this paper is a taxonomy of validated pain points across data scientists' notebook activities.

Our findings identify that data scientists face considerable pain points through the entire analytics workflow—from setting up the notebook to deploying to production—across many notebook environments. While our participants reported workarounds, these were ad hoc, required manual interventions, and were prone to errors. Our data scientist report their key needs are support for deploying notebooks to production and scheduling time-consuming batch executions as well as under-the-hood software engineering support for managing code and history. Our findings further our understanding of requirements for supporting data scientists' day-to-day activities, and suggest design opportunities for researchers and toolsmiths to improve computational notebooks and streamline data science workflows.

STUDY DESIGN

Our investigation consisted of two studies. Study 1, a mix of complementary field observations and interviews, investigates the difficulties that data scientists face in their day-to-day activities. Study 2 confirms our findings from Study 1 through a survey of 156 data scientists.

CHI '20, April 25–30, 2020, Honolulu, HI, USA.

ACM ISBN 978-1-4503-6708-0/20/04 ...$15.00.
http://dx.doi.org/10.1145/3313831.3376729

[1] https://notebooks.azure.com/
[2] https://databricks.com/product/
[3] https://colab.research.google.com/
[4] https://jupyter.org/
[5] https://nteract.io/

Table 1: Field Study and Interview Participants

FIELD STUDY PARTICIPANTS					
ID	ROLE	INDUSTRY	EXP. (YRS)	NOTEBOOKS	LANGUAGES
FP1	Data Scientist	Advertising	5	Jupyter, RStudio	Python, R
FP2	Data Scientist	Cloud Computing	3	Jupyter, VS Code	Python
FP3	Data Scientist	Machine Learning	15	Jupyter	Python
FP4	Data Engineer	Machine Learning	3	Jupyter	Python
FP5	Data Engineer	Data Services	2	Jupyter, AzureML	Python
INTERVIEW PARTICIPANTS					
ID	ROLE	INDUSTRY	EXP. (YRS)	NOTEBOOKS	LANGUAGES
IP1	Cloud Soln. Architect	Cloud Computing	4	Jupyter, Zeppelin	Python
IP2	Data Scientist	Business Analytics	3	Jupyter, Databricks	Python
IP3	Data Scientist	Cloud Computing	4	Jupyter, Databricks	Python
IP4	Data Scientist	Security	10	Jupyter	Python
IP5	Data Scientist	Cloud Computing	4	Jupyter	Python
IP6	Soln. Architect	Development Tools	4	Jupyter, Colab	Python
IP7	Database Architect	Environmental Consulting	6	Jupyter	Python, Julia
IP8	Data Analyst	Entertainment	7	Jupyter, iPython	Python
IP9	Software Engineer	Manufacturing	8	Mupad	C#
IP10	Data Analyst	Finance	5	Proprietary	Python
IP11	Consultant	Finance	9	Jupyter, Databricks	Python
IP12	Consultant	Finance	15	Jupyter	Python
IP13	Data Scientist	Security	10	Jupyter, Colab	Python, R
IP14	Software Engineer	Cloud Computing	9	Databricks	Python
IP15	Data Scientist	Development Tools	5	Jupyter, Databricks	Python, R

Study 1: Field Observations and Exploratory Interviews

To understand when and why data scientists experience difficulties with notebooks, we conducted field observations to observe data scientists in situ. We complemented these observations with interviews with professional data scientists to get a broader picture.

Recruitment. For our field observations, we recruited five professional data scientists from within Microsoft—a large, multinational, data-driven organization—using our internal address book to sample data scientists with the title of "Senior" (or higher) and having at least two years of experience in the organization. For our interviews, we recruited 15 data scientists having at least two years of data science experience from multiple companies and industries.

Participants. Participants reported working in a variety of industries and roles, including Data Analyst, Data Scientist, and Data Engineer (Table 1). Participants reported 6.6 years of experience ($sd = 3.7$). Participants primarily reported using Python with Jupyter notebooks.

Field study protocol. We observed data scientists who primarily work in computational notebooks as they performed their regular data science activities. All sessions were conducted with a single observer and a single data scientist in the data scientist's office. Sessions were scheduled for one hour, with 45 minutes of observation and 15 minutes of retrospective interviews. During the session, we recorded their screens and audio through screen capture software and a hand-held audio recorder. We asked participants to think-aloud as they worked, and the observer took in-situ field notes as the data scientists conducted their work. During the retrospective interview, the observer used these notes to probe further about difficulties observed or mentioned during the session.

Interview study protocol. We conducted semi-structured interviews remotely through online communication software, and recorded these interviews. The questions roughly followed this organization: brief questions regarding what they do as a data scientist and what activities they conduct using notebooks, followed by more detailed conversation about why they prefer to use notebooks as well as any difficulties when using the notebooks. Interviews were about 30-45 minutes long.

Informed consent. In both field observations and interviews, participants signed a consent form prior to conducting the study, in accordance with our institutional ethics board. Participation in the study was voluntary and participants received no compensation.

Analysis. We transcribed the audio for the field observations and interviews. The first and last author collaboratively analyzed these transcripts through an inductive, open coding process using the ATLAS.ti qualitative analysis software. First, we segmented the transcripts and applied descriptive codes [35], that is, assigning short phrases or labels, to these segments. We merged and split descriptive codes as necessary, looking for similarity in challenging *activities* that data scientists experienced when working in the notebook. Next, we performed axial coding, grouping similar codes and analyzing them to identify higher-level and cross-cutting themes,

which we termed *pain points*. The collaborators met frequently over several weeks to discuss, examine, and refine codes and themes.

Validity of qualitative coding. To support interpretive validity, we recruited two external raters, both data scientists. We randomly sampled five statements from each of the nine pain points as identified in our inductive coding process. Using Table 2, we asked the raters to independently categorize each statement and assign it to a single pain point that *best* reflected that statement. We achieved a Cohen's Kappa just below 0.8, with disagreement from our external raters primarily because some statements concern multiple pain points.

Study 2: Confirmatory Surveys

To triangulate, validate, and increase the credibility of our qualitative field observations and interviews, we conducted a survey with a broader population of data scientists.

Survey protocol. Our survey consisted of demographic questions about the respondents' number of years of data science experience, the computational notebooks they use regularly, the programming languages they use regularly within these notebooks (multi-option question, with "other" as an answer choice), and how frequently they use computational notebooks for their data science activities (5-point Likert-type item frequency scale, "More than 5 times a week," "4-5 times a week," "2-3 times a week," "once a week," and "less than once a week."

Using the findings of our qualitative analysis of field observations and interviews, 20 activities were drawn and presented as a series of questions using a 5-point Likert-type scale for difficulty and importance. All questions in the survey were optional.

To evaluate if our field observations and interviews reached theoretical saturation, we asked respondents to indicate if there were any other difficulties with using computational notebooks that we missed.

Informed consent. Our survey included instructions for informed consent, in accordance with our institutional policies.

Recruitment. We recruited participants by e-mail through internal address book contacts across multiple organizations, through social media such as Twitter and LinkedIn, and through data science mailing lists. We also asked respondents to forward the survey to other data scientists. Respondents did not receive compensation for completing the survey.

Respondents. 156 data scientists from various companies responded to our survey, after discarding eight blank responses. Respondents had an average of 5.3 years of experience ($sd = 3.9$). 53% of our respondents used notebooks "more than 5 times a week" while 16.6% of them used notebooks once or less than once a week. 98% of our respondents primarily used Python in notebooks, 30% used R, and 14.7% used Scala. Respondents also reported using Java, JavaScript, Spark, and SQL. 84% used Jupyter notebooks and 33% used Jupyter Labs. 36% reported using Databricks and 28% use Azure Notebooks.

Analysis. We computed descriptive statistics and plotted the survey responses for difficulty and importance. We manually inspected the responses for other challenging activities that we missed. Respondents rarely populated this response. In all cases, the answers were additional details for already-covered activities ("restructuring the code" for the activity of "refactoring" and "using loops to create multiple plots" for "visualizing data and models") or confirmational responses ("looks like you've covered everything").

PAIN POINTS OF USING NOTEBOOKS

We identified nine categories of pain points in computational notebooks, across the data scientists' workflow (Table 2).

Setup

Difficulties with notebooks happen as soon as the data scientist creates a new notebook.

Loading data. To explore data, it first has to be pulled into the notebook. That's not always easy, especially when the data needs to be shuffled back and forth between multiple sources and platforms (IP10, IP11, IP14, IP15, FP1). This process quickly becomes a tortuous, multi-step adventure that requires repeatedly "going to separate cloud instances to bring down the data locally, taking that to a local file, and uploading it back to the cloud" (IP15). Although some data libraries exist (for example, `psycopg2`)[6], they are nontrivial to use, and data scientists must be aware that they exist and remember how to use them. Unsurprisingly, some of our data scientists relied on others for help—IP10 had a developer build *magic commands* in the notebook that "triggered functions behind the notebook" on their behalf and provided them with an easier-to-use interface to connect to their commonly-used data sources.

Sometimes, when working with large data sets, the "notebook tends to crash a lot; the kernel dies and that causes frustration" (IP13). In contrast to IDEs, the client/server nature of notebooks complicates setup, and the difficulties of working with large data within a web browser are amplified. In such cases, this often "requires getting a lot of data engineering resources just to be able to run something that's supposed to be a daily job. The stack to do something pretty simple is pretty heavy" (IP15). But not all data scientists have dedicated developers or data engineers to help them.

Cleaning data. While clean data makes "a big difference in the overall model output" (IP15), it's seldom readily available. Efforts to clean data are mostly clerical, and there's "no mystery—it's just time consuming" (IP1). To avoid repetitive cleaning data scientists create "a bunch of routines" (IP11). But these routines still require modification and manual copying-and-pasting across notebooks (IP4, IP9, IP15)—an error-prone process.

Explore and Analyze

Although notebooks profess to allow "quick and dirty work and exploration" (IP1, IP3, IP4, IP6, IP8, IP11 , IP12, IP13, IP14), data scientists tell us that this isn't always the case.

[6] http://initd.org/psycopg/

Table 2: Summary of Pain Points in Computational Notebooks

PAIN POINT	DESCRIPTION	EXAMPLE
Setup	Loading and cleaning data from multiple sources and platforms is a tortuous, multi-step, manual process.	"If you do a lot of data loading and pre-processing always re-loading the data is time consuming" (IP2).
Explore and Analyze	An unending cycle of copy-paste and tweaking bits of code made worse by feedback latency and kernel crashes.	"I need immediate feedback, like when I am testing slight changes in the model. I don't want to execute everything again" (IP1).
Manage Code	Managing code without software engineering support results in "dependency hell" with ad hoc workarounds that only go so far.	"Debugging is a horrible experience, copying the code over to do the debugging outside [in the IDE], and copying it back" (IP8).
Reliability	Scaling to large datasets is unsupported, causing kernel crashes and inconsistent data.	"Disconnects between browser-server or server-kernel introduce all sorts of lack-of-reliability problems" (IP6).
Archival	Preserving the history of changes and states *within* and *between* notebooks is unsupported, leading to unnecessary rework.	"The thing is using any kind of versioning mechanism for notebooks is just a complete and utter failure" (IP2).
Security	Maintaining data confidentiality and access control is an ad hoc, manual process where errors can leak private client data.	"We are missing a more private way of handling credentials. I don't want client credentials be visible to others" (IP13).
Share and Collaborate	Sharing data or parts of the notebook interactively and at different levels—demo/reports, review/comment, collaborative editing—is generally unsupported.	"There are cases where somebody is asking you to review/comment, while other times to go collaborate" (IP6).
Reproduce and Reuse	Replicating results or reusing parts of code is infeasible because of high levels of customization and environment dependencies.	"The fact that somebody could run a notebook on organization A's service but not on organization B's is a serious problem" (IP6).
Notebooks as Products	Deploying to production requires significant cleanup and packaging of libraries—DevOps skills that are outside the core skill set of data scientists.	"Once the code gets a certain level of maturity, it's very difficult to transition that to production code. Everything has to translate to functions and classes" (IP15).

Modeling. Building models take time. Not only is "[having the system] learn the model itself very time consuming" (IP7), it also "involves a lot of complexity" just to build them (IP3). Getting to the right models require many iterations, but data scientists don't get "immediate feedback" so that they can quickly make adjustments (IP1, IP2). Instead, data scientists like IP1 have to wait a long time to check if the execution was successful, and they can't interrupt the process to evaluate alternatives in the meantime. Worse, if their model produces an error, they have to "execute everything again" (IP1).

Visualizing. Data scientists use visualizations—primarily plots—to quickly see how their code refinements modify their data (IP1, IP7, IP8, IP6, IP12, IP13). But it's hard to customize the plots when the data scientist isn't happy with the result; these frustrations led our data scientists to continuously copy-paste or tweak bits of code (IP4, IP12, IP15) to tailor the visualizations to their needs. In some cases, the output cells themselves are a hindrance—"there's a lot of things just limited by the footprint of the notebook. Everything is in a cell, and the chart is limited by the boundaries and real estate of the notebook" (IP15). In these situations, data scientists end up manually exporting their code and data and redoing the work in exploratory data analysis tools outside of the notebook—"integration with tools like Tableau, or even API level access to them, would reduce all this copying and pasting" (IP8). Since notebooks are intended for facilitating data exploration, it is unfortunate that visualization continues to be difficult within some notebook environments.

Iterating. Having to iterate between modeling and visualization, that is, "changing some methods slightly and trying different things" (IP1), is the norm. Of course, supporting iteration is one of the core purposes of computational notebooks. Notebooks should be an ideal environment for iterating, but like we saw in setup, churning through code introduces many of the same difficulties when code assistance isn't available: the data scientist ends up "having to go through the same ceremony to do even the most basic modeling task" (IP8), finding relevant packages, and deleting now-unused code (IP2, IP8, IP9, IP11, FP3). One option is to switch to an IDE, but

this requires constantly shuffling between the IDE and the notebook.

Manage Code

Although managing and working with code is a fundamental activity in the computational notebook paradigm, data scientists told us about code-related activities they found to be challenging.

Writing code. Having to write code—particularly due to lack of code assistance—is something that IP7 "hated the most" about working in notebooks. To be efficient, they had "to know all the function names and class names correctly and have another browser open to search for help and documentation" (IP7, FP2). Coding in notebooks is even more difficult using new libraries since it's not possible "to explore the API and functions" from within the notebook (IP8). Practically, IP8 argues, "anyone who tries to use notebooks has to start off with an IDE and then graduate into a notebook."

Managing dependencies. Having to manage packages and library dependencies within the notebook is, to put it mildly, a "dependency hell" (IP7). Notebooks provide little-to-no-support for finding, removing, updating, or identifying deprecated packages (IP3, IP7, IP9, IP11, IP12, IP13, IP15). Often, discovering what packages are even installed isn't accessible from the notebook environment, requiring data scientists to plod over to their command-line terminal and use commands like `conda` and `pip` to manage their environment (IP3).

Debugging. Feedback about debugging in notebook was mixed. Some data scientists applauded the notebook's ability to quickly diagnose errors (IP2, IP5, IP14), while others maintained that it's "a horrible experience" (IP6, IP8, IP15).

On one hand, the cell-oriented structures make debugging "fairly instantaneous and straightforward" because it allows "splitting up functions into different cells and then slowly stitching them together until you get something that works right" (IP2). In the case of errors, notebooks at least usually retain the output states of previously executed cells. On the other hand, the only way to debug in most notebooks is through the use of `print` statements—many computational notebook don't let data scientists "peek inside variables and change them" (IP7). Due to out-of-order execution, typical IDE affordances like breakpoints would make it difficult to "follow the code flow" (IP2), and notebooks likely require different affordances to support debugging in this exploratory context (for example, data introspection).

Testing. Tests are another mechanism to troubleshoot issues in notebooks, but because there's no standard way to test notebooks, different data scientists end up following different approaches (IP1, IP3, IP4, IP5, IP6, IP9, IP13, FP1). For example, while IP13 and FP1 wrote test cases within the same notebook, IP3 and IP4 used dedicated test notebooks to validate their functionality.

Reliability

One problem with computational notebooks is that executing them "isn't particularly reliable" (IP6). The other is that they don't scale to big data.

Executing notebooks. If the notebook kernel crashes during the middle of an operation, or the data scientists gets disconnected—this can result in putting the notebook or the data in an inconsistent state. For example, when inserting many records into a database, getting disconnected from the notebook can result in only partial records being written to a data store (IP10, FP4). Inconsistent state can be hard to detect because there's "no transparency in terms of understanding how the process is being executed on the kernel" (IP2). And some notebooks "get very large due to people abusing them; as notebooks get larger, the reliability falls" (IP6). Sometimes it's easiest to just restart and run the whole notebook again (IP8, IP10, IP15).

Scaling to big data. A limitation to iterating is that notebooks "can only handle so much data in the notebooks" (IP8, IP10). "Although notebooks could be used for lightweight extracting, transforming, and loading data (ETL), for heavyweight ETL we still have to rely on the Java pipelines. The data is way too huge for notebooks to handle" (IP6). Notebooks introduce a tension between balancing the needs of quick iteration and working with large data (IP4).

A key reason was that reliable kernel connections were hard to maintain and resulted in kernel crashes (IP1, IP2, IP6, IP10, IP13). These crashes were often a result of the notebooks' limited processing power, which can't handle large notebooks or big data loads.

Archival

While some exploratory notebooks have a relatively short shelf life as "playgrounds" (IP5), other notebooks have longer lifetimes. For the latter, data scientists need support for versioning and searching notebooks.

Versioning. There's "a lot of room for improvement when we want to check notebooks into source control, such as being able to visualize the differences between the last version and the new version" (IP3). Using traditional versioning mechanisms intended for source code are "just a complete and utter failure" (IP2) when versioning notebooks. IP2 continues, "because all the outputs are saved within the notebook, there's a lot of state that's bundled in the file." In traditional source control systems, all of these changes appear as spurious differences, making it difficult to identify the actual changes between the notebooks—"there's just a a lot of mess" (IP2, FP2). To be effective, version control systems need special-handling for computational notebooks. Moreover, since "the history and execution order of the notebook cannot be tracked by version control systems" (IP2, IP15), committing the notebook to version control doesn't mean that you'll be able to successfully run the notebook.

Searching. Data scientists create a lot of notebooks, and these notebooks are "rarely maintained or given useful names, which make it hard to know what's saved in these notebooks" (IP9). Since the "number of notebooks grows quickly" (IP7), folders and files names "become disorganized very fast. It's hard to remember what is saved in these things, so they're all just `Untitled-1` or `Untitled-2`" (IP9). All of which make "finding and navigating to the intended file difficult" (IP14).

Security

Securing notebook was difficult along two dimensions: avoiding unintentional data leaks and managing access to the notebooks.

Securing sensitive data. Securing data required our data scientists to perform "additional suppression and transformation operations," which is both time consuming and computationally costly (IP11). Adding security into the mix "slows down the whole [data science] process" (IP11). In some cases, it's the transformation code itself that needs to masked as it can reveal patterns of the underlying suppression mechanisms. But notebooks don't provide a way to "lock the functionality of a particular cell so that it's not visible to others," while still allowing them to execute the code (IP13). For certain types of analysis, the transformed data had to be again "unmasked," which introduced even more steps into the data science process. Finally, copying or exporting desensitized data over to external platforms can be "dangerous, because if you're copying data and screw up something," this data can be inadvertently leaked (IP8).

Controlling access. In some notebook environments, it's not possible to finely control access to the notebooks, and "the only way to share notebooks securely is to upload notebooks to secure team drives" (IP6, IP10, IP11, FP5). For example, data scientists aren't able to specify the types of activities that others can perform within a notebook, such as making the notebook "read-only or only allowing comments" (IP12) or restricting notebook to "only run but not modify cells," or modify cells but not run (IP10, IP13).

Share and Collaborate

Sharing and collaboration is "messy" (IP5) and "not straightforward" (IP2) but "very often requested" (IP6).

Sharing supporting artifacts. "Sharing a notebook is kind of useless without the underlying data" (IP9). But doing so means that the recipient has to be able to have access to the database, and the correct data within it (IP9, IP15). In addition, data scientists also need to "match the environment settings" of the notebook (IP3, IP15), for example, by following manually-written instructions. Alternatively, the sender can create separate "requirements files" which can be read by package installation programs to automatically install the necessary dependencies (IP15). Unfortunately, this approach breaks the "single file" metaphor of notebooks, because information needed to run the notebook is decoupled from the notebook itself.

Sharing with non-data scientists. Data scientists need to present their notebooks to customers and other non-data scientists. They do so in three ways: interactively (where the data scientists execute the notebook directly to explain the results to the audience), statically (for the audience to look at the analysis and results at a later time/offline), or as a read-only notebook (for non-data scientists to leave comments or notes on the notebook for the data scientist to review at a later time).

When interactively sharing notebooks through live demos, data scientists desired to execute the notebook interactively without exposing code, since the code isn't meaningful for most of the audience. Data scientists wanted the ability to temporarily "collapse code and show only the visualizations and markdown" (IP3, IP7, IP15).

Data scientists also shared their results by exporting the content of notebooks to other formats, such as presentations or printable reports. But because visualizations and data in notebooks "spit out a lot of information, it's hard to create nice looking reports" (IP4).

Finally, some data scientists wanted stakeholders to be able to leave comments on the notebooks without being able to run the cells. "Giving the non-technical users an opportunity to provide feedback on the notebook itself by leaving comments can be extremely valuable" (IP3, IP6).

Editing collaboratively. "Collaborating in real-time with multiple users on a single notebook is super powerful" (IP6). However, collaboration is not possible in many notebook environments, which makes it challenging to "develop side-by-side with somebody, especially remotely" (IP3).

Reproduce and Reuse

Data scientists reproduce notebooks to verify how a result was computed or obtained. They also reuse and adapt code from existing notebooks to save time.

Reproducing. "Reproducibility is really ensuring that the set of software you used to produce a result is exactly the same the next time you try to reproduce the same result" (IP6). "The only way another person can run the notebook is if they're able to match all the environment settings" (IP15). Another challenge to reproducibility is that data scientists customize notebooks, for example, by using extensions that they install on their local notebook environment. Depending on the extension, these notebooks can fail to run if they are not available in the other notebook environment (IP2, FP5). Crucially, "there's no compatibility at all between extensions in different notebook implementations. There's absolutely no standard for how these extensions work. So chances are it's not gonna work. That's a serious problem" (IP6).

Reusing and adapting. Data scientists reuse notebooks, for example, in situations where "the data source can change, but the underlying logic would be the same" (IP11). Unfortunately, even seemingly simple reuse can become more complicated than expected, such as when the "earlier notebook uses absolute paths" (IP5), when the "cells have no designated format or function" (IP1) and can't be easily isolated, and when there are complex dependencies to bring into the new notebook (IP3, IP11, FP4).

Notebooks as Products

Notebooks shine when performing quick, interactive, exploratory analysis, but these features also encourage sloppy coding practices that make it difficult to transition notebooks to production.

There are two common pathways to production. First, the data scientist can run the notebook directly, but with larger data and with longer-running computations. Second, they can package the code in the notebook into a standalone artifact

that is decoupled from the original notebook. Neither option is particularly pleasant.

Scheduling long-running computations through the notebook. Running long-running computations in notebooks provides no feedback on progress; while the computation is running, data scientists lose all interactivity in their notebook (IP1, IP8). Data scientists would prefer that, "when the process is done, it automatically creates a notification" (IP15). At least this way, the data scientist knows that they can go back to the notebook (IP4, IP5, IP14). While this option is more convenient, data scientists reported reliability issues when scaling to big data.

Packaging as a standalone artifact. Because "notebooks encourage coding in cells, it's not software-engineering friendly" (IP15). During exploration and analysis, we found that data scientists often created quick and dirty code, but "code in production software is very different from code in the notebook" (IP15). To make matters worse, cells in the notebook don't have to execute linearly, and the desired execution order isn't stored explicitly with the notebook (IP6, IP15, FP2). For these reasons, "it's very difficult and painful to transition notebooks to production code without doing a full rewrite" (IP4, IP14, IP15).

SURVEY RESULTS

Our qualitative results from field observations and interviews show that data scientists' work is rife with pain points. But, which of the activities within these pain points are critical and obstruct notebook usage? Figure 1 and Figure 2 show a distribution of responses for importance and difficulty, respectively. The distribution of importance shows that all activities are at least moderately important (that is, median of responses were above 3) to data scientists.

Since all activities are challenging for some data scientists, we focus on *high-impact* activities when discussing design opportunities in the next section. These are activities that respondents marked (by median) as at least "difficult" and "important," across multiple types of notebooks. The four activities (and the associated pain points) under this criteria are: viewing and exploring history (archival), refactoring code in notebooks (manage code), scheduling long-running processes for automatic execution (notebooks as products), and easily deploying notebooks as products (notebooks as products).

Another lens through which we can prioritize activities is to look at *low-impact* activities. These are activities that are at least "important" and "easy" or "very easy." In other words, these activities are already well supported in computational notebooks, so notebook developers should avoid impacting these activities as they introduce new capabilities in their notebooks. These activities (and pain points) are: load (setup), explore (explore and analyze), and visualize data (explore and analyze), debug (manage). It's also notable that data scientists find both local and remote notebook environments important to their day-to-day activities.

DESIGN OPPORTUNITIES

The pain points in our taxonomy suggest both understudied research areas and design opportunities for improving computational notebooks for data scientists. We first discuss design opportunities for high-impact activities, followed by design opportunities in more niche data science scenarios. Encouragingly, notebook developers are already targeting some of these design opportunities—but support thus far is piecemeal, with data scientists having to shuffle across different computational notebook environments.

Refactoring

When compared with traditional integrated development environments, such as Visual Studio Code[7] or PyCharm[8], our data scientists report that proper coding assistance within notebooks is almost non-existent. Familiar features like autocompletion, refactoring tools, and live templates are often missing or do not function properly. Consequently, data scientists spend substantial time using online resources like Stack Overflow or documentation to help them code. Even when code snippets were available online, data scientists had to manipulate and wrangle these snippets to fit within their existing code, introducing unnecessary clerical errors in the process. A short-term pathway is for toolsmiths to incorporate the familiar coding assistance features of the integrated development environment and make them available within the notebook. A longer-term pathway is to leverage affordances unique to the computational notebook paradigm: the interplay between input code cells and graphical output cells enable new mixed-initiative experiences like programming-by-example [9, 25] and programming-by-demonstration [4, 26, 11, 16, 8] within the notebook environment. For example, consider a data scientist who writes some initial plotting code in an input cell that renders a plot in the output cell. With programming-by-demonstration, the data scientist would be able to directly manipulate the plot through graphical affordances, and these manipulations could automatically update the corresponding code.

In many ways, coding assistance is possibly even more important to data scientists than software engineers: for data scientists, coding is primarily a means to answer questions about their data, and not the core activity of interest. Languages such as R and Python are popular with data scientists because of their "swiss-army knife' capabilities, but the flexibility of these languages—for example, dynamic typing and reflection—make them difficult for static analysis tools to reason about [6, 7]. For researchers, it's important that we support the languages data scientists actually use, not the languages we *wish* they would use.

Deploying to Production

Deploying and adapting exploratory notebooks for production environments, such as customer-facing applications, requires data scientists to acquire expertise in a skill set that is well outside of their core, day-to-day responsibilities. While larger organizations have data engineering or software engineering teams to assist data scientists in deployment, in smaller organizations data scientists must take on these responsibilities themselves [1].

[7] https://code.visualstudio.com/
[8] https://www.jetbrains.com/pycharm/

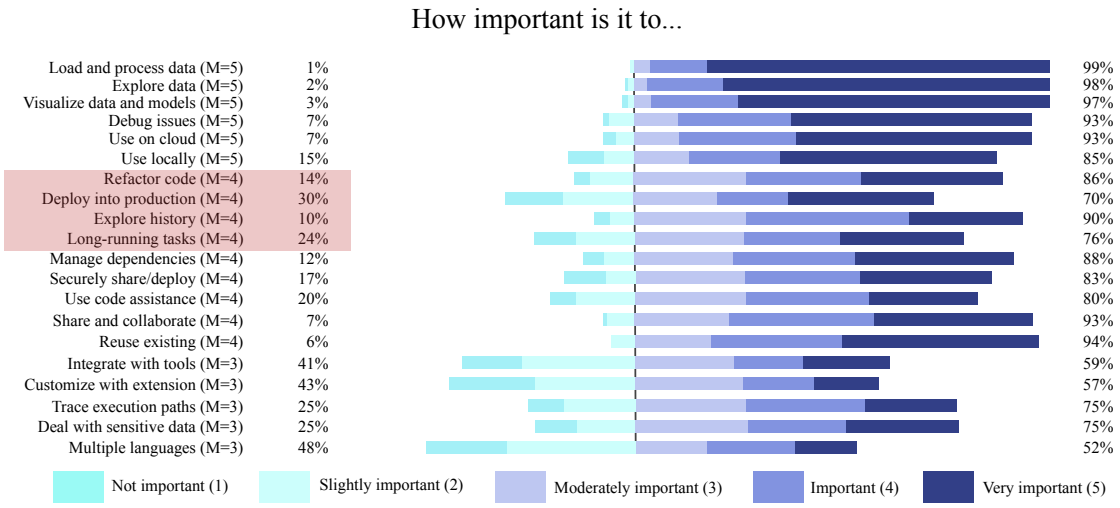

Figure 1: Distribution of importance for activities, measured from 1 (Not important) to 5 (Very important). The median importance for each activity is reported as M=median. Activities highlighted in red boxes are high-impact activities (median importance=4, median difficulty=4).

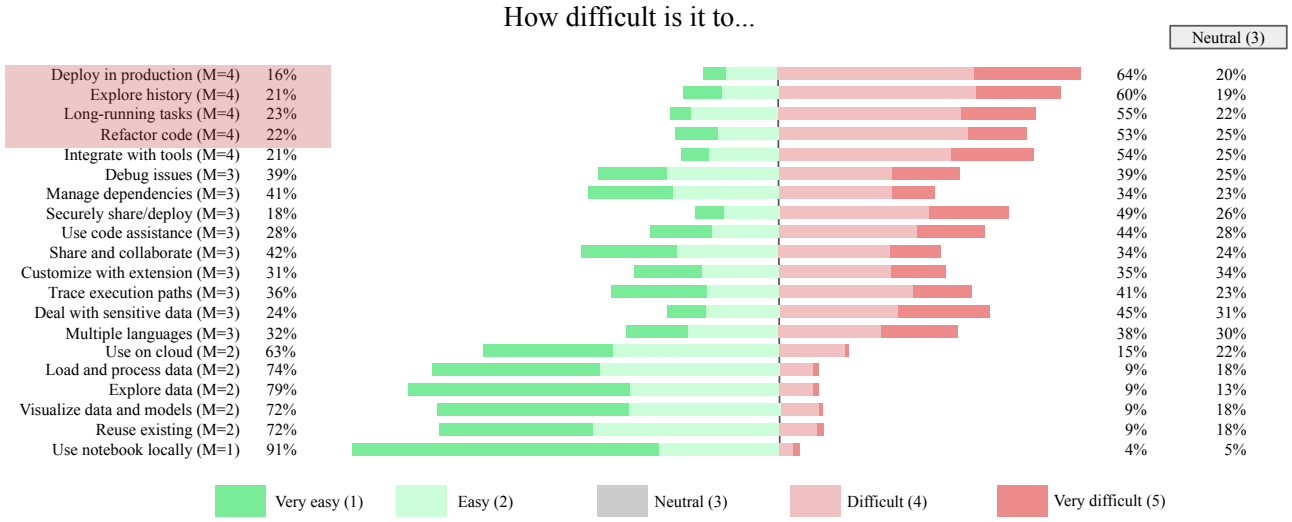

Figure 2: Net stacked distribution of difficulty for activities, measured from 1 (Very easy) to 5 (Very difficult). The net stacked distribution removes the neutral option, and shows the skew between Difficult (red bars) and Easy (green bars). The median difficulty for each activity is reported as M=median. Activities highlighted in red boxes are high-impact activities (median importance=4, median difficulty=4).

There are several opportunities to improve the data scientists' user experience and productivity. First, the programming languages for data science—such as Python or R—are not often the same programming languages that are used in production environments—such as C++ or C#. Automated translation tools or runtime environments are one opportunity to help mitigate these challenges. In circumstances when using the data science programming language is suitable, the data scientist must still clean up their notebook to remove unnecessary dependencies, dead-ends, and unused code in their exploration before exporting the notebook to a standalone script for execution in the production environments [14, 10]. Tools such as Papermill [27] are a step in the right direction: Papermill allows data scientists to directly deploy their notebook to a production environment. The tool enables "parameters" in the notebook; essentially, these parameters allow production systems to pass configuration options to the notebook, for example, the name of the data source to analyze. Still other promising directions can be found in computational notebook environments such as Databricks and Zeppelin. These environments provide higher-level abstractions and make default decisions about complex configuration options on behalf of

the data scientist, simplifying the user experience for them. In short, data scientists desire push-button approaches to productionize their notebooks.

History

Due to the affordance of out-of-order execution in computational notebooks, it can quickly—both unintentionally and *intentionally*—have an internal state that is different from the linear order of the notebook presented to the data scientist. This is one form of history-related pain point *within* a notebook. Without history support within the notebook, data scientists must painfully debug how their notebook entered the current state. As data scientists conduct exploration, build models, and evaluate multiple design options, they also introduce multiple notebooks into their environment, which they often manage in an ad-hoc way (for example, `experiment_current.ipynb` and `experiment_old.ipynb`). This is the other form of history-related pain point *between* notebooks. Without support for easily finding the right information across notebooks, data scientists end up reimplementing functionality or using stale data because they lose track of their notebooks.

In contrast to version control typically found in software engineering systems (`git`), preserving the history of artifacts is also important to data scientists. Novel innovations in the notebook space are required to support these unique requirements. For example, Head and colleagues [13] found that data scientists do not want to spend up-front effort in managing code versions or organizing their code, and that data scientists prefer automated approaches to version control management. Some notebook environments, such as Nextjournal[9], attempt to address this need by automatically and periodically versioning the notebook, data, and full runtime environment.

As computational notebooks use both text and visual medium, advanced tools and techniques are required that can easily differentiate changes (both text and visual) between different versions of notebooks. For example, Kery and colleagues [18] developed an experimental tool that complements existing version control systems and allows the data scientist to visualize the history of individual cells, code snippets, markdown, and outputs.

Finally, supporting annotations and comments in computational notebooks allows data scientists to catalog their thoughts and comments for decisions they've made in the notebook, essentially, a form of inline journaling.

Long-running Computations

The scale of data and computation increases as data scientists incorporate more demanding activities into their computational notebooks. Activities previously offloaded to standalone scripts or jobs, like working with large datasets, are now routinely conducted within the notebook itself. These activities are enabled by big data libraries that are directly accessible in Python, such as tensorflow and keras, as well as through data connectors that give data scientists access to large data repositories from within the notebook.

But large scale data means long-running computations, which often block operations in the underlying language kernel, creating tensions with the interactivity that data scientists expect from notebooks. For example, many data science libraries are synchronous, meaning that they block the data scientist from any other operations. And with streaming data—that is, data that is continuously generated by a data source—the program runs forever.

To support these activities, notebook developers must support scalable computations as a first-class design goal in notebooks. In other words, computational notebooks should maintain the benefits of exploration, interactivity, debugging, and visualization, irrespective of the size of the data.

Potential opportunities include *transaction* support, which allows the data scientist to abort long-running cells and revert to a safe state prior to executing the problematic cell. For streaming data, *reactive* notebooks such as BeakerX[10] and Tempe[11] automatically update computations as new data becomes available in the notebook. However, enabling interactivity and scaling requires careful design and introduces new challenges—in particular, these notebook models can exaggerate confusion data scientists already have with out-of-order execution.

RELATED WORK

How Data Scientists Use Notebooks

We found two studies categorizing how data scientists use notebooks. Kery and colleagues [20] studied coding behaviors in computational notebooks, focusing on how data scientists kept track of variations they explored. Rule and colleagues [34] looked at notebook structure, findings tensions between exploration and explanation in notebooks, and identifying reasons academic data scientists use notebooks—tracking provenance, reusing code, enabling replication, and presenting results. Our study differs in that we focus on *pain points* that professional data scientists have when working with computational notebooks.

The Jupyter organization conducted a survey to identify what respondents liked about Jupyter notebooks, and what needs remain unaddressed. Our study confirmed some of their findings [15] but disconfirmed others. For example, we also found that our respondents, for the most part, found it easy to explore and visualize in notebooks. However, our respondents did not find lack of debugging tools to be a substantial pain point. Our data scientists also reported that traditional version control solutions (like git) are *not* appropriate for computational notebooks, in contrast to the Jupyter survey. Moreover, our study systematically focuses on *why* certain activities are challenging across multiple computational notebook environments, using field observations and interviews.

Data Scientist Practices

Several studies investigated work practices of data scientists, although not specifically in computational notebooks. Kandel

[9] https://nextjournal.com/

[10] http://beakerx.com/

[11] https://www.microsoft.com/en-us/research/project/tempe/

and colleagues [17] described recurring pain points, outstanding challenges, and barriers to adoption for visual analytic tools. Muller and colleagues [24] described how scientists, scholars, engineers, and others work with their data. Wilson and colleagues [38] recommended several general practices for working in computational notebooks, such as modularizing code, but the exploratory nature of computational notebooks disincentivize several of these practices. Hannay and colleagues [12] found that most scientists learn to develop and use software informally from their peers. Kery and colleagues [2] described exploratory programming—such as in computational notebooks—as a key practice for data scientists; they explored further the behaviors and characteristics of this work. Wang and colleagues [37] proposed design implications to better support collaborative editing, as current synchronous editing features result in interference and imbalanced participation. And Kross and colleagues [22] studied how to teach data science in industry and academia.

Tools for Computational Notebooks

Researchers have developed some tools to improve the experience of computational notebooks, mostly around version control and history [18, 23, 19], cleaning and annotating code [34, 13], working with streaming data [5], and extending the notebook model to make visualizations more central [39]. Our study finds several additional opportunities for improving the user experience in notebooks, not yet addressed by existing research.

LIMITATIONS

Any empirical study has its strengths and weaknesses. Field studies excel in capturing how participants actually work, but this data has to be collected opportunistically. Researchers need to be present at the time a participant is engaging in the relevant activity, in our case, working on computational notebooks. Interviews, by contrast, can be scheduled a priori and are easier to perform. However, they depend on participants' memory and (sometimes) what participants *think* they should do and not necessarily what they *actually* do. Survey data is the easiest to collect, but it doesn't offer the richness of the other two empirical methods. A mixed-method study, like ours, helps balance the strengths and weaknesses of different methods.

Our study focused primarily on professional data scientists who use notebooks in their day-to-day work activities. Consequently, our findings may not generalize to other types of notebook users, such as machine learning enthusiasts, educators, end-user programmers, or infrequent notebook users. In addition, most of our participants primarily used Jupyter notebooks. Thus, some of the pain points may not apply to other notebooks, such as R Notebooks and Spyder, which have different architectures.

The importance and difficulty of our identified pain points are dependent on job roles, data scientist experience, and types of notebooks the data scientists primarily use. Our reported findings aggregated these factors, and any individual data scientist may have different priorities in how they perceive the importance and difficulty of these activities. In our survey, self-selection bias may have also influenced our results [3]: our respondents may feel more strongly about pain points in notebooks than the general population of data scientists.

Establishing validity in qualitative research is challenging due to several potential biases, including researcher bias, confirmation bias, and interpretive validity [28]. To reduce these issues, we employed three techniques. First, we calculated IRR for external raters who categorized participant statements to our pain points. Second, we conducted a survey to triangulate our interview and field study results. Within the survey, we asked participants in a short text response if there were any pain points we had missed: no new pain points were identified through these responses. Third, at the conclusion of our study, we conducted an informal focus group with both data scientists ($n = 3$) and computational notebook developers ($n = 2$, Jupyter and nteract) to identify any potential credibility issues.

The focus group identified some additional limitations in our study. Participants suggested that our pain points may be biased towards tabular ("rectangular") data, and that data scientists who work with images, audio, or natural language text may be underrepresented. For these domains, loading, processing, and exploring data may be far more difficult than our results suggest. Developers of notebooks identified that accessibility issues did not appear as a pain point in our study, although they routinely receive bug reports about these issues. They suggested that lack of keyboard shortcuts, text-to-speech support, and high-contrast themes, may impact productivity for some data scientists. Overall, however, the pain points identified in our study resonated with the focus group.

CONCLUSION

In this paper, we conducted a mixed-method study with data scientists using field observations, interviews, and a survey. In field studies and interviews, our data scientists reported a variety of difficulties when working with notebooks, and we synthesized these difficulties into a taxonomy of pain points. We validated the challenging activities that contribute to these pain points through a survey and found that supporting all of the activities were at least moderately important to data scientists, and that four activities—refactoring code, deploying to production, managing and working with history, and executing long-running tasks—were both broadly difficult as well as important to address, turning them into high-impact activities. Our findings suggest several design opportunities for researchers and computational notebook developers. Addressing these pain points can substantially improve the usefulness, productivity, and user experience for data scientists who work with computational notebooks.

Acknowledgements

We thank Audrey Au, Devesh Desai, Sumit Gulwani, John Lam, Natalia Morales, and Adriana Moscatelli for their help and feedback. This work is partially supported by the National Science Foundation under Grant Nos. 1560526 and 1815486.

REFERENCES

[1] Titus Barik, Robert DeLine, Steven Drucker, and Danyel Fisher. 2016. The bones of the system: A case study of logging and telemetry at Microsoft. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 92–101.

[2] M. Beth Kery and B. A. Myers. 2017. Exploring exploratory programming. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 25–29. DOI: http://dx.doi.org/10.1109/VLHCC.2017.8103446

[3] Jelke Bethlehem. 2010. Selection bias in web surveys. *International Statistical Review* 78, 2 (2010), 161–188.

[4] Allen Cypher and Daniel Conrad Halbert. 1993. *Watch What I Do: Programming by Demonstration*. MIT Press.

[5] Robert DeLine, Danyel Fisher, Badrish Chandramouli, Jonathan Goldstein, Michael Barnett, James F Terwilliger, and John Wernsing. 2015. Tempe: Live scripting for live data. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 137–141.

[6] Olivier Flückiger, Guido Chari, Jan Ječmen, Ming-Ho Yee, Jakob Hain, and Jan Vitek. 2019. R melts brains: An IR for first-class environments and lazy effectful arguments. In *Proceedings of the 15th ACM SIGPLAN International Symposium on Dynamic Languages*. ACM, 55–66. DOI:http://dx.doi.org/10.1145/3359619.3359744

[7] Aviral Goel and Jan Vitek. 2019. On the design, implementation, and use of laziness in R. *Proceedings of the ACM on Programming Languages* 3, OOPSLA (2019), 1–27.

[8] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. In *ACM SIGPLAN Notices*, Vol. 46. ACM, 317–330.

[9] Sumit Gulwani. 2016. Programming by examples. *Dependable Software Systems Engineering* 45, 137 (2016), 3–15.

[10] Philip J. Guo. 2012. BURRITO: Wrapping your lab Notebook in computational infrastructure. In *4th USENIX Workshop on the Theory and Practice of Provenance (TaPP)*. USENIX.

[11] Philip J. Guo, Sean Kandel, Joseph M. Hellerstein, and Jeffrey Heer. 2011. Proactive wrangling: Mixed-initiative end-user programming of data transformation scripts. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST)*. ACM, 65–74. DOI: http://dx.doi.org/10.1145/2047196.2047205

[12] Jo Erskine Hannay, Carolyn MacLeod, Janice Singer, Hans Petter Langtangen, Dietmar Pfahl, and Greg Wilson. 2009. How do scientists develop and use scientific software?. In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering (SECSE)*. IEEE, 1–8. DOI: http://dx.doi.org/10.1109/SECSE.2009.5069155

[13] Andrew Head, Fred Hohman, Titus Barik, Steven M. Drucker, and Robert DeLine. 2019. Managing messes in computational notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI)*. ACM, Article 270. DOI: http://dx.doi.org/10.1145/3290605.3300500

[14] Eric Horton and Chris Parnin. 2019. DockerizeMe: Automatic inference of environment dependencies for Python code snippets. In *Proceedings of the 41st International Conference on Software Engineering (ICSE)*. IEEE, 328–338.

[15] Jupyter. 2015. Jupyter Notebook UX Survey. (2015). https://github.com/jupyter/surveys/blob/master/surveys/2015-12-notebook-ux/analysis/report_dashboard.ipynb

[16] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 3363–3372.

[17] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer. 2012. Enterprise data analysis and visualization: An interview study. *IEEE Transactions on Visualization and Computer Graphics* 18, 12 (Dec 2012), 2917–2926. DOI:http://dx.doi.org/10.1109/TVCG.2012.219

[18] Mary Beth Kery, Bonnie E. John, Patrick O'Flaherty, Amber Horvath, and Brad A. Myers. 2019. Towards effective foraging by data scientists to find past analysis choices. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 92. DOI:http://dx.doi.org/10.1145/3290605.3300322

[19] M. B. Kery and B. A. Myers. 2018. Interactions for untangling messy history in a computational notebook. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 147–155. DOI: http://dx.doi.org/10.1109/VLHCC.2018.8506576

[20] Mary Beth Kery, Marissa Radensky, Mahima Arya, Bonnie E. John, and Brad A. Myers. 2018. The story in the notebook: Exploratory data science using a literate programming tool. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI)*. ACM, Article 174, 11 pages. DOI: http://dx.doi.org/10.1145/3173574.3173748

[21] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, and others. 2016. Jupyter Notebooks: A publishing format for reproducible computational workflows. In *ELPUB*. 87–90.

[22] Sean Kross and Philip J. Guo. 2019. Practitioners teaching data science in industry and academia: Expectations, workflows, and challenges. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 1–14. DOI: http://dx.doi.org/10.1145/3290605.3300493

[23] Hiroaki Mikami, Daisuke Sakamoto, and Takeo Igarashi. 2017. Micro-versioning tool to support experimentation in exploratory programming. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 6208–6219. DOI: http://dx.doi.org/10.1145/3025453.3025597

[24] Michael Muller, Ingrid Lange, Dakuo Wang, David Piorkowski, Jason Tsay, Q. Vera Liao, Casey Dugan, and Thomas Erickson. 2019. How data science workers work with data: Discovery, capture, curation, design, creation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI)*. ACM, Article 126, 15 pages. DOI: http://dx.doi.org/10.1145/3290605.3300356

[25] B. A. Myers. 1986. Visual programming, programming by example, and program visualization: A taxonomy. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 59–66. DOI:http://dx.doi.org/10.1145/22627.22349

[26] Brad A. Myers. 1998. Scripting graphical applications by demonstration. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, 534–541. DOI: http://dx.doi.org/10.1145/274644.274716

[27] Netflix. 2018. Part 2: Scheduling Notebooks at Netflix. (2018). https://medium.com/netflix-techblog/scheduling-notebooks-348e6c14cfd6

[28] Anthony J. Onwuegbuzie and Nancy L. Leech. 2007. Validity and qualitative research: An oxymoron? *Quality & Quantity* 41, 2 (01 April 2007), 233–249. DOI:http://dx.doi.org/10.1007/s11135-006-9000-3

[29] F. Perez and B. E. Granger. 2007. IPython: A system for interactive scientific computing. *Computing in Science Engineering* 9, 3 (May 2007), 21–29. DOI: http://dx.doi.org/10.1109/MCSE.2007.53

[30] Fernando Perez and Brian E Granger. 2015. Project Jupyter: Computational narratives as the engine of collaborative data science. *Retrieved September* 11, 207 (2015), 108.

[31] F. Perez, B. E. Granger, and J. D. Hunter. 2011. Python: An ecosystem for scientific Computing. *Computing in Science Engineering* 13, 2 (March 2011), 13–21. DOI: http://dx.doi.org/10.1109/MCSE.2010.119

[32] M. Ragan-Kelley, F. Perez, B. Granger, T. Kluyver, P. Ivanov, J. Frederic, and M. Bussonnier. 2014. The Jupyter/IPython architecture: A unified view of computational research, from interactive exploration to communication and publication. *AGU Fall Meeting Abstracts* (Dec. 2014), H44D–07.

[33] B. M. Randles, I. V. Pasquetto, M. S. Golshan, and C. L. Borgman. 2017. Using the Jupyter Notebook as a tool for open science: An empirical study. In *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. 1–2. DOI: http://dx.doi.org/10.1109/JCDL.2017.7991618

[34] Adam Rule, Aurélien Tabard, and James D. Hollan. 2018. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI)*. ACM, Article 32, 12 pages. DOI: http://dx.doi.org/10.1145/3173574.3173606

[35] Johnny Saldaña. 2009. *The Coding Manual for Qualitative Researchers*. SAGE Publications.

[36] Helen Shen. 2014. Interactive notebooks: Sharing the code. *Nature News* 515, 7525 (2014), 151.

[37] April Yi Wang, Anant Mittal, Christopher Brooks, and Steve Oney. 2019. How data scientists use computational notebooks for real-time collaboration. *Proceedings of the ACM on Human-Computer Interaction (CSCW)* 3 (2019), 1–30.

[38] Aruliah D.A. Brown C.T. Hong N.P.C. Davis M. Guy R.T. Haddock S.H. Huff K.D. Mitchell I.M. Plumbley M.D. Wilson, G. and B. Waugh. 2014. Best practices for scientific computing. *PLoS Biology* 12, 1 (2014), e1001745.

[39] Jo Wood, Alexander Kachkaev, and Jason Dykes. 2018. Design exposition with literate visualization. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2018), 759–768.