Analyzing 4 Million Jupyter Notebooks

A follow-up to *Study 1: Analyzing 1 Million Jupyter Notebooks* from Design and Use of Computational Notebooks by Adam Rule

In July of 2017, Adam Rule "scraped and analyzed 1.23 million publicly available Jupyter Notebooks on GitHub" (Rule, 23). In a section of his PhD Dissertation, he reports statistics reflecting the role of narrative in computational notebooks. His report focuses on languages and packages used and the amount of context provided by users in terms of the amount of markdown compared to code and repository descriptions. Here, we inspect how these aspects of the use of Jupyter Notebooks has changed over the past two years. The structure of this paper follows that of section 3.2 of Adam's dissertation, discussing the results of my analysis and how they compare to his 2017 results.

The number of notebooks on GitHub has grown dramatically since Rule's research. Between July 2017 and July 2019, the number of notebooks on GitHub have over tripled, now at 5,376,454 notebooks and 414,614 users. The methods I used to query, download, and extract data from GitHub is based on Adam's research. His code can be found in Jupyter Notebooks on GitHub under activityhistory/jupyter_on_github, and my scripts can be found on GitHub under jupyter-resources/notebook-research.

I first queried all public, non-forked Jupyter notebooks, then downloaded them and their corresponding repository metadata. There are just over 5 million notebooks on GitHub, and of those, I was able to download 93%. 23% of all notebooks were a part of ipynb_checkpoints, a version control feature of Jupyter that saves the most recent manually-saved versions of notebooks. This allows users to return to that point even if the program has automatically saved the notebook since. Files in ipynb_checkpoints are either exact duplicates or very recent versions of the "real" notebook which is in the same repository. To avoid double-counting these notebooks, I dropped these files from our dataset. I was unable to download repository metadata for 6.6% notebooks. This can be because the repository was deleted, its name was edited, or it was changed to private in the time between querying and downloading, or because there was a problem with the JSON file. Finally, 1% of notebooks that were downloaded were empty and were removed. This leaves us with just over 4.2 million notebooks.

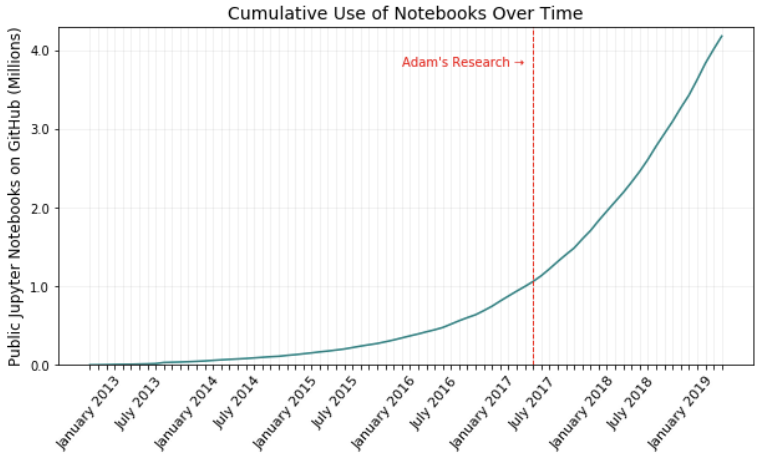

Figure 1: Use of Jupyter Notebooks over time.

Owners

In July of 2017, there were 100,503 GitHub users with publicly shared Jupyter notebooks, at the time approximately 0.4% of all owners. Further, Adam revealed that 24.5% of these owners only hosted one notebooks on GitHub. Now, there are 414,614 owners, 1.16% of the nearly 36 million users (GitHub search for users). Not far from the 2017 study, 28.7% of Jupyter users host only one notebook on GitHub.

Notebooks owners can either be individual users or organizations. Adam's dissertation doesn't mention the differences between owners and organizations, so there is no 2017 baseline to refer to. 4.72% of owners are organizations and they own 7.86% of all notebooks. The median number of notebooks per owner is 4 for both organizations and owners, but the mean number for organizations is much higher at 19.03 compared to 10.88 notebooks for users. Both distributions are skewed right, but more so for organizations. The most notebooks held by a single owner is 26,032, a record held by the organization learn-co-students.

Repositories

There are now 917,535 public repositories on GitHub with at least one Jupyter Notebook, 3.37% of the 27.2 million public repositories (GitHub search for public repositories). This is compared to 191,402 in 2017. I found that 46.9% of repositories have only one notebook and 11.25% have ten or more. In 2017, Adam calculated these as 39.1% and 14.6%, respectively.

Languages and Packages

98.3% of notebooks have a specified language, an increase from 85.1% calculated by Rule. The distribution of languages has stayed very similar, all within a percentage point of what was reported in 2017. Looking only at notebooks with specified languages, 97.62% are written in Python. The next most popular languages are Julia and R, making up 0.8% and 0.95% respectively.

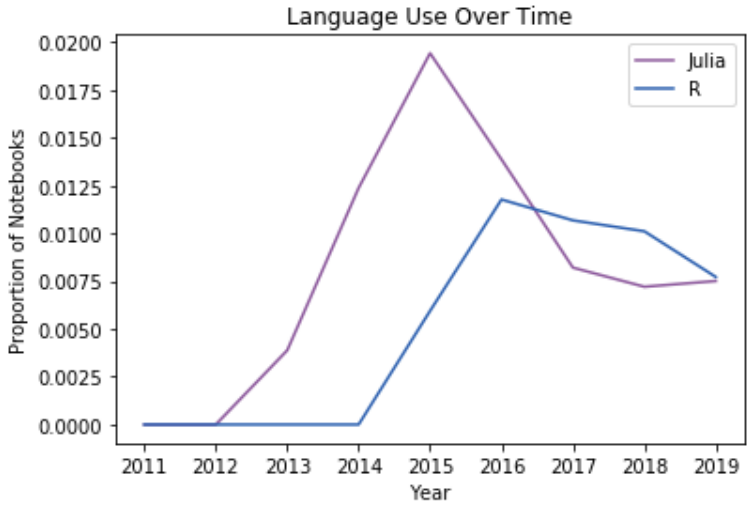

Figure 2: R vs Julia over time.

Python's popularity has stayed pretty consistent, but Julia and R have changed slightly over time (See Figure 2). Julia hit its peak popularity in 2015, used in nearly 1.5% of notebooks with a specified language, followed by a quick decline to just over 0.7% in 2019. R reached 1% in 2017 and has decreased slightly since. In years since 2017, R has been more popular than Julia and the total number of notebooks has increased significantly, meaning overall, there are more notebooks written in R than Julia. Many other languages are used (e.g. Scala, Bash, C++, Ruby), but with minimal frequency (all together, less than 0.7%).

86.94% of Python, R, or Julia notebooks have imports, compared to 89.1% from 2017. Consistent with Adam's research, the most common Python imports are `Numpy` (70.8% of notebooks with imports, compare to 67.3%), `Pandas` (47.0%, compare to 42.3%), and `MatPlotLib` (55.6%, compare to 52.1%).

I also investigated whether some packages are likely to be imported together. Adam did not investigate this, so I don't have comparisons to 2017. One frequent co-occurrence is `SkLearn` , a machine learning framework, with `Re`, the regular expression package for python frequently used to pre-process text data before modeling takes place (r = 0.51). `Time` carries time-related functions while `Datetime` has a focus on time formatting and manipulation (r = 0.61). `Numpy` is used for data manipulation and to calculate summary statistics while `Matplotlib` is for data visualization (r = 0.56). There are also moderate correlations between `SkLearn` and `Pandas` (r = 0.36), `Pandas` and `Numpy` (r = 0.31), as well as `SkLearn` and `os` (r = 0.35).

All of the combinations mentioned so far make sense because the packages that occur frequently together perform related but different functions, meaning they're used in different parts of the same process. `Matplotlib` and `Seaborn`, however, are both popular visualization packages. Theoretically anything that can be accomplished with one can also be done using the other, so it doesn't make as much sense to use these packages together as it does for the other combinations. These two are still frequently used in the same notebooks (r = 0.34).

I then looked more specifically at the imports of machine learning frameworks. The most frequent machine learning framework imported is `SkLearn` (24.1% of notebooks with imports). `Tensorflow`, `Keras`, and `Torch` are also used, but much less frequently (8.1%, 5.9%, and 2.8%, respectively). `Theano`, `MXNet`, `Caffe`, `CNTK`, and `PyTorch` are used extremely rarely (all less than 0.5%). More on machine learning framework use is discussed in the results on Data Science with Jupyter.

Notebook Size and Content

The majority of notebooks are under 20 kilobytes in size, but notebooks range from 62 bytes and 100 megabytes (GitHub's file size limit). The median number of cells per notebook is 19 (mean 28.7). 4.0% of notebooks have no code, up from 2.2% in 2017. Among notebooks with code, the median number of lines of code has remained at 86 lines (mean 145.7). 27.8% of notebooks have no markdown text, the exact same as in 2017. Among notebooks with markdown, the median number of words is 194 (mean 550.9). This has gone down since Adam's research, when a median of 218 words was calculated.

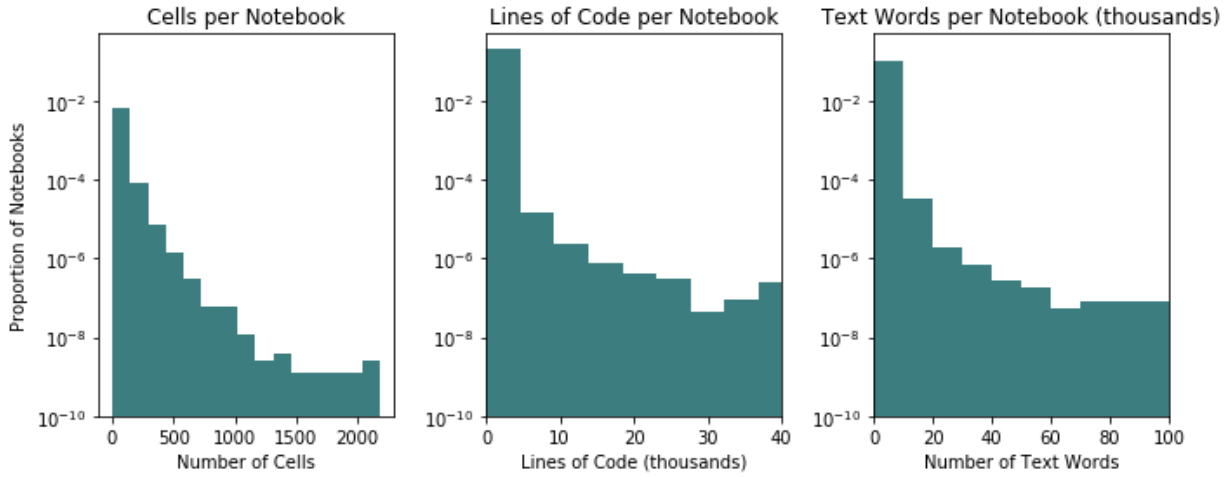

Figure 3: Notebook Content Distributions.

Organization

Adam identified the pattern of notebooks containing more markdown at the beginning and more code at the end. This pattern is still present. I calculated that while 53% of cells within the first 5% of a notebook are markdown, this drops to 26% for the last 5% of a notebook (see Figure 4).

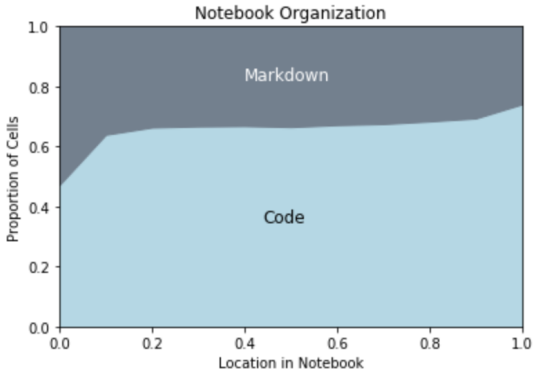

Figure 4: Markdown vs code distribution across the notebook.

Execution and Outputs

Jupyter notebooks have cells that can be run independent of the rest of the notebook, allowing users to iteratively change and re-run sections of code. Notebooks are usually run top to bottom but cells can be run out of order, which has the potential to cause confusion. At the 2018 JupyterCon, research engineer Joel Grus discussed this confusion in a presentation titled, "Why I don't Like Notebooks", showing examples of how "the ability to run code snippets in arbitrary order is *weird* and *unintuitive*" (Grus, slide 36). The execution order of cells is stored in the state of each Jupyter notebook. Inspecting this, I found that 25% of all notebooks had cells run out of order.

Non-linear execution can be especially confusing when cells have output. What is displayed will stay the same even if earlier, related cells are changed. Take, for example, these two cells. These were run in order, and the output of the second cell makes sense.

But we can change the first cell and re-run it, migrating this notebook to a non-linear execution order. Now the output of the second cell does *not* make sense; we expect to see "100".

In 2017, Adam looked specifically at the execution order of notebooks with output. I calculated that 84% (compare to 85%) of notebooks have at least one cell with output, and 29% of such notebooks were run out of order. This is a huge decrease since the 43.9% observed in 2017.

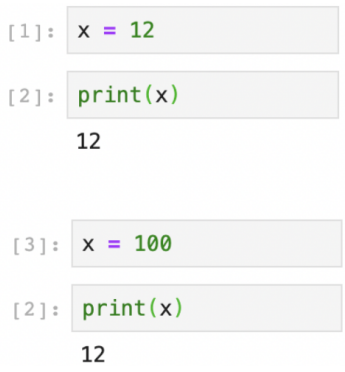

Figure 5: Execution order.

There are three types of output supported by Jupyter notebooks, "stream (e.g., print statements), executed results (e.g., numerical results), and displayed data (e.g., rich data displays such as graphs and tables)" (Rule, 27). 68.3% of notebooks have at least one stream output (compare to 68.5%). 59.3% of notebooks have at least one execute output (compare to 58.1%). Further, 44.7% of notebooks have at least one displayed data (compare to 45.5%).

Non-linear execution also gives users the ability to use functions, variables, and imports in cells above where they are initialized. Users can even *delete* the initialization and continue their work, as long as their kernel stays alive. This impedes reproducibility, as a notebook cannot be run from top to bottom without editing or rearranging code.

To investigate the order of function, variable, and package uses, I parsed the code of each notebook using the abstract syntax tree package for Python. AST only works on Python 3 code and it cannot parse Jupyter magic commands (prefixed with %), shell commands (prefixed with !), or documentation calls (prefixed with ?). After removing these lines, only 86.25% of Python 3 notebooks were able to be parsed. 5.54% of notebooks that could be parsed call user-defined functions before the function definition. Narrowing in on notebooks with at least one user-defined function, this raises to 12.53%. 2.72% of notebooks that could be parsed use variables before defining them. Finally, 1.12% use packages before importing them. Among notebooks with at least one import, this raises to 1.6%.

Repository Descriptions

54.1% of repositories have a repository description, comparable to the 58.5% Rule observed in 2017. The most commonly used words in repository descriptions are: data, learning, project, python, using, machine, analysis, code, repository, and deep. These are very similar to the words listed in Rule's dissertation, "learning, project, machine, udacity, course, deep, nanodegree, neural, kaggle, and model" (Rule, 27). One noticeable difference is that references to online learning classes and competitions ("udacity" and "kaggle") are not as frequent as they were back then. This may be a sign that notebook use is expanding to new topics beyond online learning.

Sources

1. Rule, Adam. Design and Use of Computational Notebooks. 2018. https://adamrule.com/files/dissertation_rule.pdf.
2. GitHub. activityhistory/jupyter_on_github repository. https://github.com/activityhistory/jupyter_on_github. Accessed 29 July 2019.
3. Github. jupyter-resources/notebook-research repository. https://github.com/jupyter-resources/notebook-research. Accessed 29 July 2019.
4. GitHub. Owner profile for learn-co-students. https://github.com/learn-co-students. Last updated 2019.
5. GitHub. Search for public Repositories. https://github.com/search?q=is:public. Accessed 29 July 2019.
6. GitHub. Search for users with greater than or equal to zero repositories. https://github.com/search?utf8=%E2%9C%93&q=repos%3A%3E=0&type=Users&ref=advsearch&l=&l=. Accessed 29 July 2019.
7. Grus, Joel. I don't like notebooks. https://conferences.oreilly.com/jupyter/jup-ny/public/schedule/detail/68282. 24 August 2018.