A Comparison of Machine Learning Code Quality in Python Scripts and Jupyter Notebooks

Kyle Adams (Moravian University), Aleksei Vilkomir (East Carolina University), and Mark Hills (Appalachian State University)

37th Annual CCSC:Southeastern Conference (CCSC-SE 2023)
November 3-4, 2023
Coastal Carolina University, Conway, SC

The origin story: 2022 summer REU at ECU

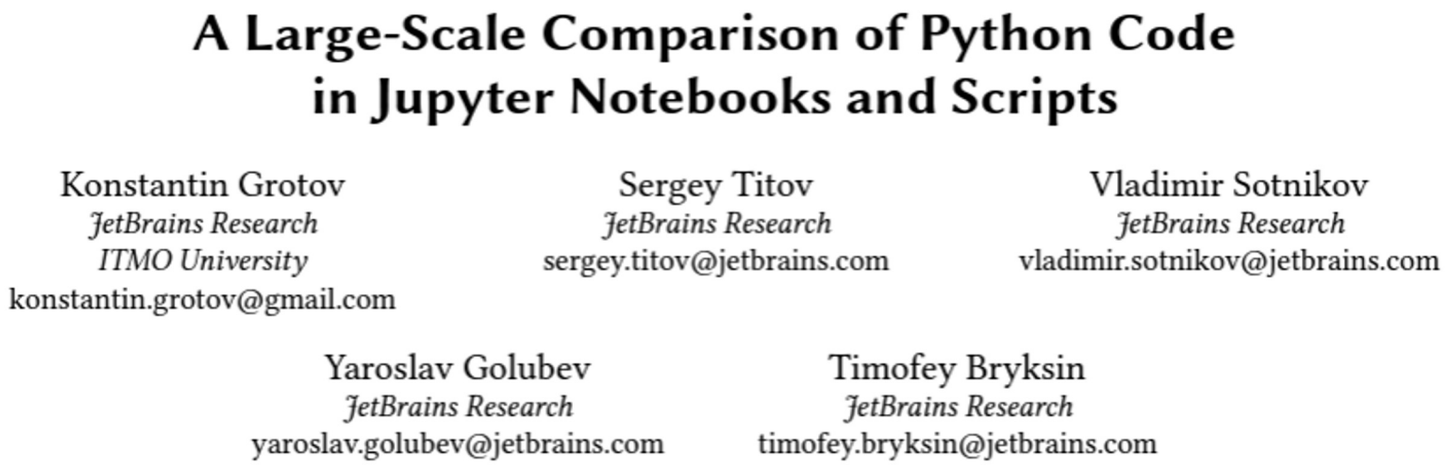
- In An Empirical Exploration of Python Machine Learning API Usage, Vilkomir explored how developers use different ML libraries in their code

- One incidental finding: code is often messy, maybe as a result of copying in examples, with repeated imports and imports of libraries that are never used

- This led to a 2022 REU project at ECU with Adams, Vilkomir, and Hills, focused on code quality in Jupyter notebooks as compared to Python scripts

- But, then something happened…

A Large-Scale Comparison of Python Code in Jupyter Notebooks and Scripts (Grotov et al., MSR, 2022)



A Large-Scale Comparison of Python Code in Jupyter Notebooks and Scripts

Konstantin Grotov
JetBrains Research
ITMO University
konstantin.grotov@gmail.com

Sergey Titov
JetBrains Research
sergey.titov@jetbrains.com

Vladimir Sotnikov
JetBrains Research
vladimir.sotnikov@jetbrains.com

Yaroslav Golubev
JetBrains Research
yaroslav.golubev@jetbrains.com

Timofey Bryksin
JetBrains Research
timofey.bryksin@jetbrains.com

- Published at MSR after the start of the REU by about 2 weeks, but found on arXiv when we started, this paper essentially studied the exact topic we had selected

- Studied a huge corpus – essentially, all Python scripts and notebooks on GitHub that were released under an open-source license

Findings from Grotov, and a new project idea

- Grotov et al. found significant differences in structural metrics and code style metrics between notebooks and scripts
    - Notebook code seems to be simpler, on average, but with higher coupling and more style errors
    - Some style errors seem to reflect normal ways of using notebooks, so notebooks may need specialized linters
- “Another interesting aspect of comparison is the *domain* where the files come from. In our work, we aim to compare notebooks and scripts in general, whereas selecting scripts solely from the same domain that Jupyter notebooks come from (**machine learning**, education, etc.) can lead to other interesting insights.”

Pivoting to machine learning, assembling the corpus

- Kaggle (https://www.kaggle.com/) hosts competitions focused on machine learning and data science

- Meta Kaggle (https://www.kaggle.com/datasets/kaggle/meta-Kaggle) includes data about public competitions, datasets, and solutions – and, it provides an API

- Starting corpus: Top 100k projects from Kaggle, based on Meta Kaggle API, sorted by popularity (stars)

- One note: we could be missing some of the best solutions if they were not marked as public

Assembling the corpus: some stats

- Using the API, we obtained 69,858 ML files
    - 12,136 Python scripts
    - 57,722 Jupyter notebooks
- Other files were present, but disregarded (e.g., data files, documentation)
- Some files from top 100k projects could not be downloaded

Collecting results

- Our methodology was intentionally similar to that of Grotov to make for an easier comparison

- Matroskin was used to compute structural metrics over Python code included in Jupyter notebooks

- Hyperstyle was used to compute style errors (PEP-8 violations) in Python scripts

- Scripts were used to convert between notebook and script formats so each tool could be run across all files in the corpus; other scripts automated running Matroskin and Hyperstyle across all files in the corpus, collecting and processing results

Challenges along the way

- Matroskin only works on notebooks, so we needed to convert scripts to notebooks – each script became a notebook with a single cell

- Hyperstyle only works on scripts, so we needed to convert notebooks to scripts – each notebook becomes a script where each cell is a function

- Some results from Matroskin could not be processed (for 385 notebooks and 154 scripts) due to errors in the result format, we are still not sure why

- Hyperstyle needed to be run for each file to get accurate results, but would randomly hang on specific files, needed to script this carefully

Exploring the results: Structural metrics (Functions)

Metric	Notebook Mean (STD)	Script Mean (STD)
API Functions Count	6.95 (6.69)	10.20 (10.06)
API Functions Uses	13.17 (20.24)	23.29 (81.93)
Defined Functions Count	3.45 (6.00)	3.39 (6.86)
Defined Functions Uses	5.83 (13.31)	4.72 (12.91)
Built In Functions Count	4.83 (3.30)	3.80 (3.20)
Built In Functions Uses	22.51 (35.88)	14.78 (25.26)
Other Functions Uses	86.02 (99.11)	28.73 (54.03)

- Function counts give unique occurrences, uses are total occurrences

- Functions can be built-in to Python (Built In), user-defined (Defined), directly imported (API), or “Other” (e.g., through implicit imports), based on results of Matroskin

Exploring the results: Structural metrics (Functions)

Metric	Notebook Mean (STD)	Script Mean (STD)
API Functions Count	6.95 (6.69)	10.20 (10.06)
API Functions Uses	13.17 (20.24)	23.29 (81.93)
Defined Functions Count	3.45 (6.00)	3.39 (6.86)
Defined Functions Uses	5.83 (13.31)	4.72 (12.91)
Built In Functions Count	4.83 (3.30)	3.80 (3.20)
Built In Functions Uses	22.51 (35.88)	14.78 (25.26)
Other Functions Uses	86.02 (99.11)	28.73 (54.03)

- Scripts tend to import and use more API functions, while notebooks use more built-in functions

- Notebooks have significantly more “other” functions, most likely from importing entire libraries of functions instead of using targeted imports (arguably, not good coding practice)

Exploring the results: Structural metrics (Other)

Metric	Notebook Mean (STD)	Script Mean (STD)
SLOC	174.48 (216.45)	105.02 (180.11)
Comments SLOC	27.93 (48.51)	26.96 (57.13)
Extended Comments LOC	75.40 (113.45)	N/A (N/A)
Blank Lines Count	30.42 (46.97)	32.53 (46.69)
Cyclomatic Complexity	6.18 (9.28)	12.00 (23.71)
NPAVG	0.71 (0.27)	1.00 (0.24)
Cell Coupling	48.44 (358.84)	N/A (N/A)
Function Coupling	10.49 (101.02)	12.59 (105.80)

- Counted lines of source code (SLOC), comments (Comments SLOC), Markdown comments (Extended Comments SLOC), and Blank Lines

- Other metrics include cyclomatic complexity, average number of arguments per function, function coupling, and cell coupling (which treats cells like functions)

Exploring the results: Structural metrics (Other)

Metric	Notebook Mean (STD)	Script Mean (STD)
SLOC	174.48 (216.45)	105.02 (180.11)
Comments SLOC	27.93 (48.51)	26.96 (57.13)
Extended Comments LOC	75.40 (113.45)	N/A (N/A)
Blank Lines Count	30.42 (46.97)	32.53 (46.69)
Cyclomatic Complexity	6.18 (9.28)	12.00 (23.71)
NPAVG	0.71 (0.27)	1.00 (0.24)
Cell Coupling	48.44 (358.84)	N/A (N/A)
Function Coupling	10.49 (101.02)	12.59 (105.80)

- Notebooks seem to have significantly more code, but lower cyclomatic complexity (so, more straight-line code)

- Calls to script functions have, on average, more parameters (but this clusters around 1 for both)

- Cell coupling in notebooks seems significant, which could make comprehension challenging

Comparing structural metrics with Grotov

- Both scripts and notebooks for ML tend to have a higher SLOC than the Grotov corpus

- In Grotov, notebooks and scripts had a similar number of uses of built-in functions, while we found more (on average, about half again as many) in notebooks

- Both scripts and notebooks for ML had higher CC

- Our interpretation: good tools for reasoning about Python code in ML notebooks and scripts, including uses of imported and built-in functions, could help ML developers

Exploring the results: Style metrics (Best Practices)

Error Code	Error Desc	Notebook %	Script %
W0611	Import module or variable is not used	41	64
W0621	Redefining name from outer scope	22	27
W0404	Re-imported module	16	13
W0612	Unused variable name	8	14
W0613	Unused argument	4	9

- Overall, notebooks have a lower # of warnings per file (average of 14.21) as compared to scripts (23.77), codes are from PEP-8

- Warnings here may be related to copying code and not removing unused parts (W0611, W0404 with duplicate imports, W0612, W0613)

- W0621 seems separate, but could be caused by copying as well

Exploring the results: Style metrics (Code Style)

Error Code	Error Desc	Notebook %	Script %
C0411	Import order not followed	38	57
C0412	Imports not grouped by package	21	19
C0305	Trailing newlines	20	15
W0301	Unnecessary semicolon	7	4
W0311	Bad indentation	5	11

- C0411 and C0412 again are problems related to imports (PEP-8 specifies rules for import ordering and grouping), and could be caused by copying in code

- Spacing issues (C0305, W0311) could also be caused by this, but (along with W0311) could also be caused by unfamiliarity with Python

Exploring the results: Style metrics (Error Proneness)

Error Code	Error Desc	Notebook %	Script %
E1101	Variable accessed for nonexistent member	47	59
E0001	Syntax error	47	3
W0104	Statement seems to have no effect	33	6
E0611	No name in module	29	54
W0106	Expression is assigned to nothing	9	1

- E0001 may mean that notebook cells that don’t run have errors, and could be an artifact of the analysis
- E0611 and E1101 relate to naming, and could either be errors or analysis limitations (names could reference library dependencies)
- W0104 and W0106 could be detecting statements and expressions that display info in cells (side-effects) but are not used directly

Comparing style metrics with Grotov

- We have an overlap with 2 warning codes: W0611 (Import module or variable is not used) and W0621 (Redefining name from outer scope)

- Other top warning codes from best practices, code style, and error proneness categories are distinct between the ML-focused and Grotov sets of code

- 13 of 15 in Grotov are more common in notebooks, while this is only true for 7 of the ones reported here

- Past studies have tended to show that notebooks have lower quality, but that isn’t obviously true for this domain (although error proneness does seem to be higher)

Threats to validity

- Internal: we are converting between script and notebook formats, this could cause issues especially with the style metrics, although we are not seeing notebooks having consistently higher numbers of style warnings so do not believe this is a significant issue
- External: it is unclear how representative Kaggle code is of general Python ML code
  - We have looked at multiple competitions and a large number of submitted solutions with the goal of having a representative corpus
  - Looking at well-maintained code on GitHub may be a better option, but Kaggle may give more insight into issues novices face

Related work

- One branch of work: studies to see how notebooks are used for collaboration

- Another: code quality (focusing on good and bad practices) and reproducibility

- More closely related: analysis of notebooks to identify dependencies in notebooks, visualize dependencies, and provide additional information on notebook cells

- Pynblint (Quaranta et al., CAIN 2022) is a linter specifically for notebooks

Related work: Grotov, Siddik & Bezemer

- Grotov already described above – this focused on comparing notebooks and scripts across all domains

- Siddik & Bezemer (SCAM 2023, presented about a month ago) focuses instead specifically on differences between ML and non-ML notebooks, but does not look at scripts

Conclusions

- We do see clear differences between ML notebooks and scripts, but need more in-depth statistical analysis to determine which of these are truly significant

- We also see differences between ML files and the general domain of Grotov, indicating that ML-focused tools may be useful for both novices and practitioners

- Many problems seem to be related to copying and tweaking code from examples, leading to naming, import, and potentially style (e.g., indentation) issues (and maybe some of the syntax errors seen for notebooks) that proper tooling could improve

Discussion

Thank you!
Any Questions?

- Alex Vilkomir: http://www.cs.ecu.edu/alex/
- Mark Hills: https://cs.appstate.edu/hillsma/
- Zenodo: https://zenodo.org/doi/10.5281/zenodo.8122384