Data Science Through the Looking Glass: Analysis of Millions of GitHub Notebooks and ML.NET Pipelines

Fotis Psallidas, Yiwen Zhu, Bojan Karlaš[⋈], Jordan Henkel,
Matteo Interlandi, Subru Krishnan, Brian Kroth, Venkatesh Emani, Wentao Wu,
Ce Zhang[⋈], Markus Weimer, Avrilia Floratou, Carlo Curino, Konstantinos Karanasos
name.surname@microsoft.com [⋈]name.surname@inf.ethz.ch

ABSTRACT

The recent success of machine learning (ML) has led to an explosive growth of systems and applications built by an ever-growing community of system builders and data science (DS) practitioners. This quickly shifting panorama, however, is challenging for system builders and practitioners alike to follow. In this paper, we set out to capture this panorama through a wide-angle lens, performing the largest analysis of DS projects to date, focusing on questions that can advance our understanding of the field and determine investments. Specifically, we download and analyze (a) over 8M notebooks publicly available on GITHUB and (b) over 2M enterprise ML pipelines developed within Microsoft. Our analysis includes coarse-grained statistical characterizations, fine-grained analysis of libraries and pipelines, and comparative studies across datasets and time. We report a large number of measurements for our readers to interpret and draw actionable conclusions on (a) what system builders should focus on to better serve practitioners and (b) what technologies should practitioners rely on.

1. INTRODUCTION

The ascent of machine learning (ML) to mainstream technology is in full swing: from academic curiosity in the 80s and 90s to core technology enabling large-scale Web applications in the 90s and 2000s to ubiquitous technology for the masses today. We expect that in the next decade, most applications will be "ML-infused" [2]. This massive commercial success and academic interest are powering an unprecedented amount of engineering and research efforts—in the last five years alone, we have seen over 75K papers in a leading public archive (https://bit.ly/3KZFla5) and millions of publicly shared data science (DS) notebooks corresponding to billions of dollars of development cost (per COCOMO model [5]).

As our team at Microsoft's Gray Systems Lab began to invest heavily both in building systems to support DS and leveraging DS to build applications, we were faced with many open questions due to the speed of evolution of ML. As system builders, we were uncertain about what DS practitioners needed (e.g., are practitioners shifting to using only DNNs?). As DS practitioners, we were equally puzzled on which technologies to learn and build upon (e.g., shall we use TENSORFLOW [1, 34] or PYTORCH [24]?). Discussing with experts in the field led to rather inconsistent views, too.

We thus embarked on a (costly) fact-finding mission, consisting of large data collection and analysis, to better understand this shifting panorama. In particular, we analyzed (a) 8M notebooks, publicly shared on GITHUB—representative of OSS, educational, and self-learning activities; and (b) 2M ML pipelines professionally authored in ML.NET within Microsoft. DS encompasses a wide range of operations (e.g., wrangling, visualization, ML, collaboration), and collecting datasets representative of them all is a herculean task. As we will see, the datasets we use here are a first step towards this end, and skewed towards ML, visualization, and data pre-processing.

Over the past few years, we have used the results of this analysis to educate several decisions across Microsoft in a data-driven fashion (§6). Given the significant internal impact, we realized that this knowledge could serve the community at large. Hence, we summarize our key results in this paper. To our knowledge, this is the largest analysis of DS projects made public.

The diversity and sheer size of these datasets enable multiple dimensions of analysis. In this paper, we focus on extracting insights from dimensions that are most urgent for the development of systems for DS and for practitioners to interpret adoption and support trends:

- *Landscape* (§3) provides a bird's-eye view on the volume, shape, and authors of DS code.
- *Library analysis* (§4) provides a finer-grained view of this landscape. As such, it (a) sheds light on the functionality that data scientists rely on and systems for DS should focus on, and (b) informs prioritization of efforts based on the relative usage of libraries.
- *Pipeline analysis* (§5) provides an even finer-grained view by analyzing operators (e.g., learners, transformers) and the shape (e.g., #operators) of ML pipelines, both explicit (i.e., SCIKIT-LEARN and ML.NET pipelines) and implicit ones (i.e., pipelines defined using one or more libraries).

In addition to reporting objective measures in isolation, throughout the paper we perform comparative analysis to better understand trends of usage. In particular, we compare: (a) statistics for Python notebooks, libraries, and DS pipelines across four years in GITHUB datasets;

and (b) ML pipelines from GITHUB with ML.NET ones. Furthermore, throughout the paper we draw actionable interpretations of our results to better inform practitioners and system builders.

Finally, to better highlight the importance of DS code analytics in practice, we conclude our discussion with scenarios from multiple teams within Microsoft and maintainers of DS libraries on how they have used the results of our analysis over the years (§6).

2. CORPORA AND CODE ANALYTICS

For our analysis, we leverage all publicly available notebooks from GITHUB and a large dataset of ML.NET pipelines from within Microsoft. These datasets cover a broad spectrum of users—from inexperienced to somewhat experienced to experts. Although the majority of the notebooks include ML operations, at least 34% of them involve data processing and visualization but no ML operations. Next, we briefly describe each data source and our purpose-built system for code analytics.

GITHUB. In our analysis, we use three corpora of publicly available notebooks on GITHUB spanning four years (2017 to 2020): GH17; GH19; and GH20, indicating the corresponding download year. Each consists of notebooks available at the HEAD of the default (e.g., main or master) branch of all public repositories at the time of download: 1.2M notebooks (0.3TB compressed) in July 2017 for GH17 [26], 4.6M notebooks (1.5TB compressed) in July 2019 for GH19, and 8.7M notebooks (3.2TB compressed) in July 2020 for GH20. Based on our analysis, this dataset is skewed towards inexperienced or somewhat-experienced users.

ML.NET. The underlying system, ML.NET [3], has been used in production for over a decade. We obtained access to a telemetry database from 2015 to 2019, containing over 88M events. While many users opted out of reporting telemetry, this sample is representative of ML activities within Microsoft, providing an enterprise-centric vantage point. This dataset is also representative of an ML community within Microsoft that supports mostly large-scale enterprise applications.

Code Analysis System. Analyzing DS code at the scale outlined above involves multiple challenges (e.g., distributed downloading, parsing notebooks and ML.NET pipelines to extract meaningful information, analyzing DS code efficiently, supporting extensible interfaces for an ever-increasing need for analytics). To address these challenges, we designed and implemented a purpose-built system—here, we only summarize the main steps for analyzing code. We employ a distributed crawler for downloading GITHUB notebooks based on [26, 27]. Upon downloading, we parse notebooks based on the nbformat format, discard malformed notebooks, and upload extracted information from valid ones to a backend database. We include all metadata (kernel, language, and nbformat versions) and cell-level information (type, order, source code). Along with notebooks, we also keep track of related information provided by GITHUB (e.g., repository and owner). ML.NET pipelines are similarly processed. We then perform several extraction passes using sophisticated extraction subsystems (e.g., extract and semantically annotate data flows to identify training pipelines), and store results back in the same database for composition purposes (e.g., count #users that train a SCIKIT-LEARN learner using remote CSVs).

Table 1: Overall statistics for GITHUB. Average yearly growth is $\sim 2\times$ for most metrics (except "Other languages" growing at $1.4\times$). Percentages for each metric are computed over the closer "Total" upward in the table, and remain relatively similar across years.

Dimension	Metric	GH17	GH19	GH20
Notebooks	Total	1.23M	4.6M	8.7M
	Deduped	66.0%	65.5%	65.7%
	Linear	26.4%	29.1%	30.3%
	Completely Linear	21.2%	23.3%	24.6%
Languages	Python	81.7%	91.7%	91.1%
	Other	18.3%	8.3%	8.9%
Cells	Total	34.6M	143.1M	261.2M
Code Cells	Total	64.5%	66.4%	66.9%
	Deduped	41.0%	38.6%	38.5%
	Linear	72.1%	80.2%	79.3%
	Completely Linear	68.3%	76.1%	75.6%
Users	Total	100K	400K	697K

3. LANDSCAPE

We start our analysis with a bird's-eye view of the landscape of DS through coarse-grained statistics from GITHUB notebooks. In particular, our analysis in this section aims to reveal the volume of (a) notebooks and cells; (b) languages associated with them; (c) users; and (d) characteristics of code shape. Table 1 presents the overall statistics and drives our discussion.

Notebooks and Cells. We make three main observations from the statistics in Table 1 regarding notebooks and cells. First, most metrics roughly double on average every year. Along with the large volume of individual metrics (e.g., 8.7M notebooks and 261.2M cells in GH20), this indicates the wide and fast-paced adoption of notebooks as a programming medium. Second, the volume of duplicate notebooks and code cells is considerable. Sources of duplication include git forking and notebook checkpointing (users often upload both the base notebook and checkpoints), besides regular duplication of best practices and code reuse. Interestingly, however, duplication does not alter metrics and insights of our analysis (and we omit analysis on deduped code).

Languages. Looking at languages, we confirm one of our main hypotheses: Python grew its lead from 81% in GH17 to 91% in GH19 and GH20. Notebooks in

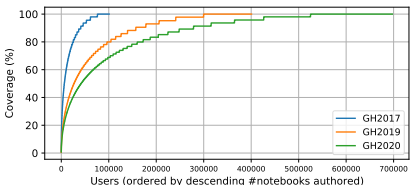


Figure 1: Users' coverage on #notebooks No spikes on the left suggest notebook authoring is not dominated by a few users.

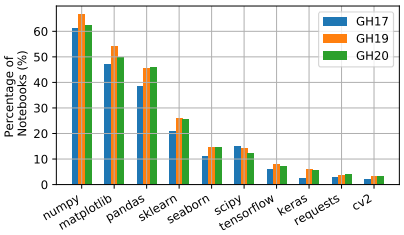


Figure 2: Top-10 used libraries.

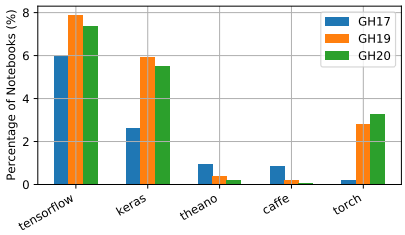


Figure 3: DL libraries usage percentages.

all other languages combined grew by roughly 1.4× on average, below Python's 2×; growth is increasing in recent years, but absolute volumes remain relatively low. These results suggest that Python is emerging as a de-facto standard for DS; system builders and practitioners should focus primarily on Python.

Users. User growth is similarly paced at ∼2× average yearly growth, reaching an impressive 697K unique users in GH20. The top-100 most prolific users have each authored at least 432 notebooks in GH17 (693 in GH19, 1052 in GH20), yet no individual user is a "heavy-hitter." This is best-seen in Figure 1, which shows the coverage of users on notebooks: note the absence of spikes on the left-hand side of the figure. Interestingly, the average number of notebooks per user has remained roughly the same across years (i.e., 12.1 per user per year).

Code shape. To better understand the code complexity of Python notebooks, we focused on how many notebooks or cells are *linear* (contain no conditional statements) or *completely linear* (contain neither conditional statements nor classes or functions). To do so, we analyzed every code cell by first parsing it to its AST representation using parso [8]. This resulted in an immense 12.9B AST nodes for GH20. At the notebook level for GH20, 25% are completely linear while 30% are linear. At the cell level, however, 76% are completely linear while 80% are linear. (Observations are similar for GH17 and GH19). Overall, this analysis shows that DS code is a mostly linear orchestration of libraries. It is thus amenable to transformation into declarative dataflows, leveraging compiler/database optimizations (e.g., lazy evaluation or cross-optimization between SQL and ML [13, 3, 2]).

Takeaways: Notebooks are emerging as a widely adopted programming medium, with a roughly ∼2× yearly growth across our metrics (see Table 1). Python is emerging as a de-facto standard language for authoring notebooks, with ∼91% of GitHub notebooks being authored in Python today. Moreover, Python code in notebooks appears mostly as a linear orchestration of libraries (80% of notebook cells are linear).

4. LIBRARIES

We continue our analysis on GitHub notebooks by focusing on which libraries are used more prominently and in which combinations. This will implicitly help us characterize what practitioners code for in such notebooks (e.g., data processing, DNNs, or classical ML).

We look at this through Python notebooks (due to their dominance across years in the GitHub datasets.) and the lens of import statements (i.e., import... or from...import...). Through imports, we observe a large number of unique libraries: 41.3K, 116.4K, and 175.6K unique libraries in GH17, GH19, and GH20, respectively (1.5× yearly growth).

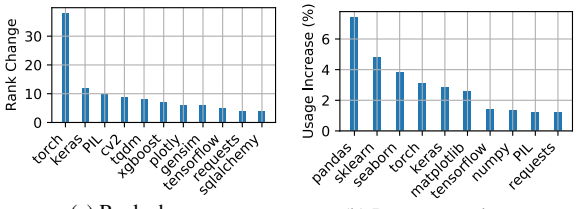
In our analysis, next, we identify important libraries (§4.1), analyze the coverage of libraries on Python notebooks (§4.2), and conclude with statistical correlations (positive and negative) between libraries (§4.3).

4.1 Important Libraries

We now focus our analysis on identifying what we informally refer to as "important" libraries.

Most used libraries. Figure 2 shows the top-10 libraries according to the percentage of Python notebooks that each library is imported. We first confirm a key assumption of our study: notebooks are used primarily for DS activities—the top-10 libraries focus on common DS tasks (e.g., communicating with external sources, processing data, ML modeling, exploring and visualizing datasets, and scientific computations). Second, we verify our intuition that NumPy [18], Matplotlib [15], Pandas [20], and Scikit-Learn [32] are quite popular. However, their frequencies exceed our expectations (e.g., NumPy is used in >60% of the notebooks). Third, by comparing usage across years, we observe that "big" (i.e., most used) libraries are becoming "bigger", with several libraries losing in popularity (e.g., SciPy [31]); indicating a consolidation around a core set of libraries. Overall, we believe these results suggest systems builders can focus efforts on a few frequently used libraries (e.g., NumPy or Pandas), but must also provide mechanisms to support a growing tail of less frequently used libraries.

Highest ranking differentials in usage. Comparing GH17 to GH20, the ranking in terms of usage changed substantially for a few libraries. Figure 4a shows libraries that increased their usage ranking the most over the last three years. We observe a popularity increase of ML frameworks (e.g., +38 positions for PyTorch [24]; Keras, XGBoost [37], and TensorFlow are in top-



(a) Rank change (b) Percentage change

Figure 4: Top-10 libraries with the most increased usage.

10). Furthermore, we observe an interest increase in image and text processing (PILLOW [21], OPENCV [19], and GENSIM [10]), while the increase of PLOTLY [22] indicates an interest increase for (interactive) visualization. Finally, the increase of TQDM [36] indicates a growing interest for showing progress bars (which, in turn, indicates long-running computations), while the increase for SQLALCHEMY [33] and REQUESTS [25] suggest a need to access data stored in databases and other endpoints.

Most increased in usage. We complement the ranking differential analysis with the percentage increase in absolute terms. The top-10 libraries by highest percentage increase are shown in Figure 4b. Interestingly, we observe that "big" libraries are getting "bigger" at a faster rate than average. We observe a similar pattern for libraries related to deep learning (e.g., KERAS, PYTORCH, and TENSORFLOW), that we analyze in more detail next.

Usage among deep learning libraries. Figure 3 shows the percentage of notebooks that use TENSORFLOW, KERAS, THEANO [35], CAFFE [7], and PYTORCH. We observe that PYTORCH has increased the most, followed by TENSORFLOW and KERAS (with the usage of the latter two slightly decreasing in GH20). Furthermore, for both THEANO and CAFFE the usage rates have dropped considerably. Overall, deep learning is becoming more popular, yet accounts for less than 20% of DS today.

Overall, we believe that the above library usage changes can be mainly attributed to (a) shifts in user interests and operations (e.g., increased interest in image processing) and (b) social and community trends (e.g., companies or classes focusing on specific toolkits). The latter cannot be unveiled solely by data.

4.2 Coverage

How to prioritize implementation efforts and what is the impact of supporting a library are important questions during the development of systems for DS. In this direction, we perform a coverage analysis of libraries on notebooks (i.e., if we only support K libraries, how many notebooks would be fully covered?).

Figure 5 shows the cumulative percentage of notebooks covered (y-axis) while varying the number of libraries (x-axis). We sort libraries from the most to the least used and pick a prefix of size K. Our main observation is that by including just the top-10 most used

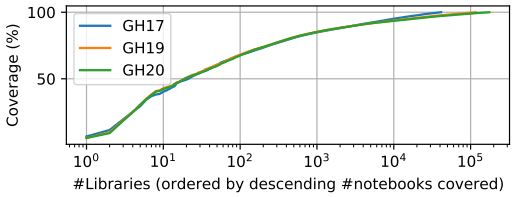
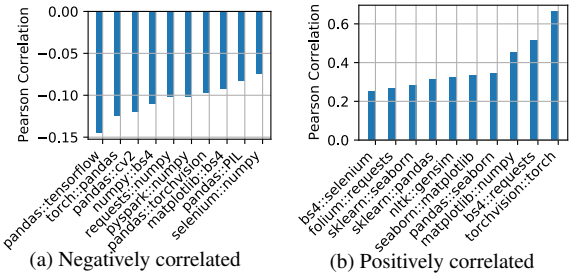


Figure 5: Percentage of notebooks to be covered.



(a) Negatively correlated (b) Positively correlated

Figure 6: Top-10 correlated library pairs.

libraries (i.e., the ones shown in Figure 2), we can reach a coverage of ~40% across years, while a coverage of 75% can be achieved by including the top-100 most used libraries. The increase in coverage, however, is diminishing as less used libraries are added in. More interestingly, a coverage of 100% is much harder to achieve in GH20 and GH19 than in GH17—suggesting that the DS field is expanding both in size and tail complexity.

4.3 Correlation

We conclude this section with an analysis of the co-occurrence (or correlation) of libraries. Figure 6a projects top-10 positively and negatively Pearson correlated libraries. (We defer a detailed discussion on individual correlations to [23]). Our main observation is that negative correlations highlight incompatibilities between supported data types and focus areas, whereas positive correlations validate common wisdom and indicate (a) to practitioners the need for expertise in certain combinations of libraries, and (b) to system builders which libraries to focus on co-optimizing.

Takeaways: Python notebooks on GITHUB emerge as large collections of DS-related activities, with all top-10 most used libraries focusing on DS. In particular, we observe an increased consolidation around a core set of highly used libraries: NUMPY, MATPLOTLIB, PANDAS, SCIKIT-LEARN are used in more than 23% of notebooks in GH20. The adoption of DL is increasing, yet it accounts for less than 20% of DS. At the same time, an increase in #used libraries indicates that DS is still an expanding field. Finally, our correlation analysis ranks library connections (Figure 6) to help system builders and practitioners decide what to co-optimize or get expertise on.

Table 2: #Extracted pipelines and distinct operators.

		GH17	GH19	GH20	ML.NET
#Pipelines	Implicit	164K	415K	1.4M	N/A
	Explicit	10K	129K	252K	29.7M
#Distinct Ops	Implicit	668K	1.8M	2.6M	N/A
	Explicit	584	3.4K	5.5K	23.5K

5. PIPELINES

So far, we have focused on understanding DS projects based on the libraries they are using (Section 4). In this section, we dive deeper into (primarily training) ML pipelines to provide an even finer-grained view of DS logic, that is also optimizable and manageable [2, 29].

Pipelines. We focus analysis on *explicit* and *implicit* pipelines. We refer to the former as pipelines defined declaratively: for our analysis, we use pipelines constructed using "sklearn.pipeline" [30] in notebooks, and the method chaining pattern in ML.NET. We refer to the latter as pipelines defined without such constructs but rather imperatively, using functions from different toolkits. To enable comparisons among explicit and implicit pipelines, we require implicit ones to be used for training, and have operators semantically close to SCIKIT-LEARN ones (which are close to ML.NET ones). As such, and given that PANDAS is the predominant way to read data (Section 4), we model implicit pipelines as data flows in notebooks with PANDAS reads and SCIKIT-LEARN learners being source and sink nodes, respectively.

Volumes. Table 2 outlines the overall volumes of extracted pipelines across datasets. Regarding GITHUB, we observe a steady increase (>1.9× growth), for both implicit and explicit pipelines. Furthermore, explicit pipelines have started increasing in size and stabilizing their growth. Moreover, implicit pipelines are larger in volume than explicit ones (e.g., 5× larger in GH20). Note, however, that from our extraction process, explicit SCIKIT-LEARN pipelines are subsets of implicit ones because a SCIKIT-LEARN pipeline can be a subpipeline of an implicit pipeline. Finally, from ML.NET we extracted 29.7M pipelines (2M unique after dedup), highlighting the overall importance of explicit pipelines in enterprises.

5.1 Pipeline Length

Figures 7a and 7b show the #pipelines per pipeline length, for both explicit and implicit pipelines, as a proxy for the complexity of problems tackled by different pipelines. (Intuitively, problems requiring more steps correspond to pipelines with more operators).

Explicit pipelines are right-skewed, with most having a length of 1 to 4. Among them, SCIKIT-LEARN pipelines have a smaller length than the ML.NET ones. Given that the operators used in both are of similar expressiveness, we believe that this is related to the type of users behind each of them. As also discussed in Section 2, expert users (represented by ML.NET) tend to address more complicated problems that require more steps, resulting in longer pipelines. Note, however, that SCIKIT-LEARN pipelines are increasing in length over the years, which might indicate that corresponding users are addressing increasingly more complex problems. Implicit pipelines are also right-skewed, but their length is much larger than the one of explicit pipelines: most implicit pipelines have a length of 4 to 100, reaching a max length of 4K in GH20. We believe this is because explicit pipelines tend to encompass a much narrower functionality than the implicit ones (e.g., data cleaning and visualization steps are uncommon in explicit pipelines), and enable expressing operations in a more succinct way.

Table 3: Top learners and transformers in explicit pipelines.

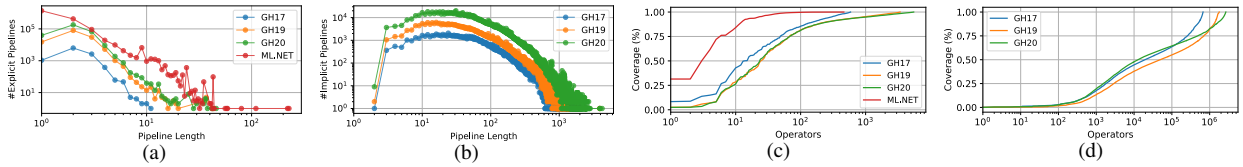
	Top Transformers	Top Learners
SKLEARN	StandardScaler CountVectorizer TfidfTransformer PolynomialFeatures	LogisticRegression MultinomialNB SVC LinearRegression RandomForestClassifier
ML.NET	OneHotEncoding FeaturizeText ReplaceMissingValues TokenizeIntoWords ProduceWordBags	Fast Tree Fast Forest SdcaLogisticRegression LbfgsPoissonRegression AveragedPerceptron

5.2 Operators

To study the type of operations that users perform using ML pipelines, we discuss #distinct operators and then we rank individual operators by frequency to identify important ones. Trends on operator frequencies are similar across years (we omit drill-downs to avoid duplication).

Table 2 shows that the #distinct pipeline operators across datasets has increased substantially for both implicit and explicit pipelines. Furthermore, it is evident that practitioners need the flexibility for introducing their own functionality: #operators of explicit SCIKIT-LEARN pipelines has increased due to user-defined transformers and learners, ML.NET contains 23K user-defined operators (and 536 system-defined ones), and #operators in implicit pipelines is much higher than the explicit ones due to the unconstrained way in encoding user logic.

Explicit Pipelines. Table 3 shows the top learners and transformers in explicit pipelines. An interesting observation is that normalizers are not within the top transformers for ML.NET while they are popular in SCIKIT-LEARN pipelines. This is because ML.NET adds these automatically based on needs of downstream operators or because data is normalized beforehand. Furthermore, regarding learners, Gradient Boosting and Random Forest are more popular in ML.NET than SCIKIT-LEARN. We believe this is due to their relative quality and the tasks observed in Microsoft. Finally, the importance of Poisson Regression in ML.NET indicates that data scientists in enterprises routinely deal with count data, and data like that is rarely released and made available publicly.



(a) (b) (c) (d)

Figure 7: Number of pipelines and coverage of operators on explicit and implicit ML pipelines.

Implicit Pipelines. The trends on transformers and learners of explicit pipelines follow the implicit ones (modulo hyperparameter tuning being more prominent in implicit ones). Importantly, however, implicit pipelines are primarily dominated by operators for data wrangling (e.g., project, select, group by, concat, aggregations, and merge) and data visualization (e.g., print, plot, and head).

5.3 Coverage

We conclude our discussion on pipelines with a coverage analysis of operators on pipelines (to better help system builders realize prioritization opportunities). Figures 7c and 7d show the coverage of pipelines while increasing operators in descending order of operator frequency. Regarding explicit pipelines, top-100 and top-10 operators in SCIKIT-LEARN and ML.NET pipelines, respectively, can cover more than 80% of corresponding pipelines, highlighting optimization opportunities. In contrast, implicit pipelines are much harder to cover in full, with >50% coverage requiring $>10^4$ operators.

Takeaways: Explicit pipelines are gaining considerable traction in GITHUB notebooks (>1.9× yearly growth), and they are an established practice in an enterprise setting. Nevertheless, implicit pipelines continue to be the dominant way (by 5× in GH20) to specify DS logic in GITHUB notebooks. While explicit pipelines appear to focus more on feature transformations, implicit ones contain more data preprocessing and visualization operations. Finally, explicit pipelines contain a core set of operators that system builders can prioritize support for, and practitioners casually introduce their own operators indicating a need for additional functionality.

6. INTERNAL IMPACT OF ANALYSIS

The analysis we presented here has been used to guide decisions on the DS space throughout Microsoft. Its impact has exceeded our expectations, leading to a continuous stream of drill-down asks from several teams.

Operationally, insights have already been used to (a) inform decision making on resource allocation and feature enhancements (e.g., what libraries to support and optimize in Azure services); and (b) verify the correctness of prior decisions (e.g., first class support of notebooks in Synapse, VSCode, and Azure Data Studio). Furthermore, this analysis has motivated decisions on several projects: (a) holistic optimization of DS pipelines [13]; (b) constructing KBs of Python libraries for data-flow analysis [17]; (c) tracking provenance from DS scripts [17]; and (d) overall shape visions and inform research agendas in the space [2]. Finally, ongoing collaborations triggered by this line of work include furthering the analysis of notebooks with analysis of context (i.e., output and markdown cells), supporting finer-grained analysis of DS code (e.g., understanding dataset types), and constructing ML models for code auto-completion and synthesis.

7. RELATED WORK

Understanding DS-related activities is crucial as most applications are becoming ML-infused [2]. Our work pushes the envelope on the topic by performing an extensive analysis of millions of Python notebooks and ML.NET pipelines. Other studies reveal interesting insights on complementary DS aspects through discussions with data scientists about their engineering and collaboration practices [4, 12, 16, 39, 28, 11, 14]. Furthermore, the work in [6] presents a coarse-grained analysis on PYPI. Our work targets the Python language as well but with a special focus on DS projects. The work in [9] compares the package dependency graphs of CRAN, PYPI, and NPM. This work targets various languages and thus does not contain a detailed analysis of Python libraries. The study in [26] performs an analysis of GITHUB notebooks with a focus on interactions between exploration and explanation of results. Our work incorporates the dataset used in this study (GH17) but focuses on analyzing the DS code structure instead. Finally, the study in [38] performs dynamic analysis of notebooks (to recommend data preprocessing operators) and provides interesting insights on DS operators, albeit on limited #notebooks due to the requirement on executing notebooks.

8. CONCLUSION

In this work, we amassed and analyzed approximately 40M DS projects, both publicly available at GITHUB and Microsoft-internal ones. With this paper, we share the key findings of our analysis with the community, provide actionable interpretations, and share how this analysis has been used internally at Microsoft to inform decisions in the DS space. Finally, we believe this analysis is pragmatically useful to better inform investments by practitioners and systems builders—and a step towards a thorough, data-driven understanding of DS as a field.

9. REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 265–283, 2016.

[2] A. Agrawal, R. Chatterjee, C. Curino, A. Floratou, N. Gowdal, M. Interlandi, A. Jindal, K. Karanasos, S. Krishnan, B. Kroth, J. Leeka, K. Park, H. Patel, O. Poppe, F. Psallidas, R. Ramakrishnan, A. Roy, K. Saur, R. Sen, M. Weimer, T. Wright, and Y. Zhu. Cloudy with high chance of DBMS: A 10-year prediction for Enterprise-Grade ML. In *Conference on Innovative Data Systems Research (CIDR)*, 2019.

[3] Z. Ahmed, S. Amizadeh, M. Bilenko, R. Carr, W.-S. Chin, Y. Dekel, X. Dupre, V. Eksarevskiy, S. Filipi, T. Finley, et al. Machine learning at microsoft with ml. net. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2448–2458, 2019.

[4] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann. Software engineering for machine learning: A case study. In *ICSE*, 2019.

[5] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece. *Software Cost Estimation with COCOMO II*. Prentice Hall Press, 2009.

[6] E. Bommarito and M. Bommarito. An Empirical Analysis of the Python Package Index (PyPI). *CoRR*, abs/1907.11073, 2019.

[7] Caffe. https://caffe.berkeleyvision.org.

[8] David Halter et al. https://parso.readthedocs.io.

[9] A. Decan, T. Mens, and M. Claes. On the Topology of Package Dependency Networks: A Comparison of Three Programming Language Ecosystems. In *ECSAW*, pages 21:1–21:4, 2016.

[10] Gensim. https://radimrehurek.com/gensim.

[11] C. Hill, R. Bellamy, T. Erickson, and M. Burnett. Trials and tribulations of developers of intelligent systems: A field study. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 162–170. IEEE, 2016.

[12] Kaggle. The State of Data Science & Machine Learning, 2019. https://www.kaggle.com/kaggle-survey-2019.

[13] K. Karanasos, M. Interlandi, D. Xin, F. Psallidas, R. Sen, K. Park, I. Popivanov, S. Nakandal, S. Krishnan, M. Weimer, Y. Yu, R. Ramakrishnan, and C. Curino. Extending relational query processing with ml inference. In *Conference on Innovative Data Systems Research (CIDR)*, 2019.

[14] M. B. Kery, A. Horvath, and B. A. Myers. Variolite: Supporting exploratory programming by data scientists. In *CHI*, volume 10, pages 3025453–3025626, 2017.

[15] Matplotlib. https://matplotlib.org.

[16] M. Muller, I. Lange, D. Wang, D. Piorkowski, J. Tsay, Q. V. Liao, C. Dugan, and T. Erickson. 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*, pages 1–15, 2019.

[17] M. H. Namaki, A. Floratou, F. Psallidas, S. Krishnan, A. Agrawal, Y. Wu, Y. Zhu, and M. Weimer. Vamsa: Automated provenance tracking in data science scripts. In *The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1542–1551, 2020.

[18] NumPy. https://numpy.org.

[19] OpenCV. https://opencv.org.

[20] Pandas. https://pandas.pydata.org.

[21] Pillow. https://pillow.readthedocs.io.

[22] Plotly. https://plotly.com/.

[23] F. Psallidas, Y. Zhu, B. Karlas, M. Interlandi, A. Floratou, K. Karanasos, W. Wu, C. Zhang, S. Krishnan, C. Curino, et al. Data science through the looking glass and what we found there. *arXiv preprint arXiv:1912.09536*, 2019.

[24] PyTorch. https://pytorch.org.

[25] Requests. https://requests.readthedocs.io.

[26] A. Rule, A. Tabard, and J. D. Hollan. Data from: Exploration and Explanation in Computational Notebooks. UC San Diego Library Digital Collections, 2017. https://doi.org/10.6075/J0JW8C39.

[27] A. Rule, A. Tabard, and J. D. Hollan. Exploration and explanation in computational notebooks. In *CHI*, 2018.

[28] N. Sambasivan, S. Kapania, H. Highfill, D. Akrong, P. Paritosh, and L. M. Aroyo. "everyone wants to do the model work, not the data work": Data cascades in high-stakes ai. In *proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–15, 2021.

[29] S. Schelter, J.-H. Böse, J. Kirschnick, T. Klein, and S. Seufert. Automatically tracking metadata and provenance of machine learning experiments. In *NIPS*, 2017.

[30] Scikit-Learn. https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html/.

[31] SciPy. https://www.scipy.org.

[32] Scikit-Learn. https://scikit-learn.org.

[33] SQLAlchemy. https://www.sqlalchemy.org.

[34] Tensorflow. https://www.tensorflow.org.
[35] Theano. http://deeplearning.net/software/theano.
[36] Tqdm. https://tqdm.github.io/.
[37] XGBoost. https://xgboost.readthedocs.io.
[38] C. Yan and Y. He. Auto-suggest: Learning-to-recommend data preparation steps using data science notebooks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, page 1539–1554, 2020.
[39] A. X. Zhang, M. Muller, and D. Wang. How do data science workers collaborate? roles, workflows, and tools. *Proceedings of the ACM on Human-Computer Interaction*, 4(CSCW1):1–23, 2020.