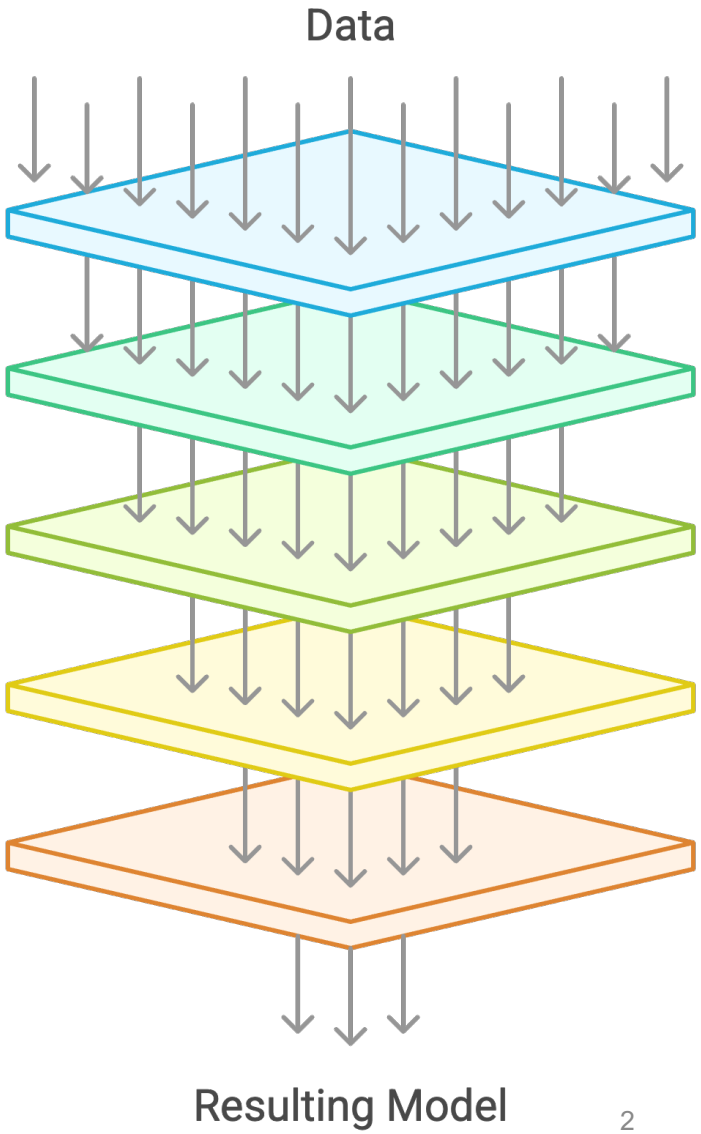
Analysing Python Machine Learning Notebooks with Moose

Marius Mignard[1]
Steven Costiou[1]
Nicolas Anquetil[1]
Anne Etien[1]

1. Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Machine learning (ML) workflow

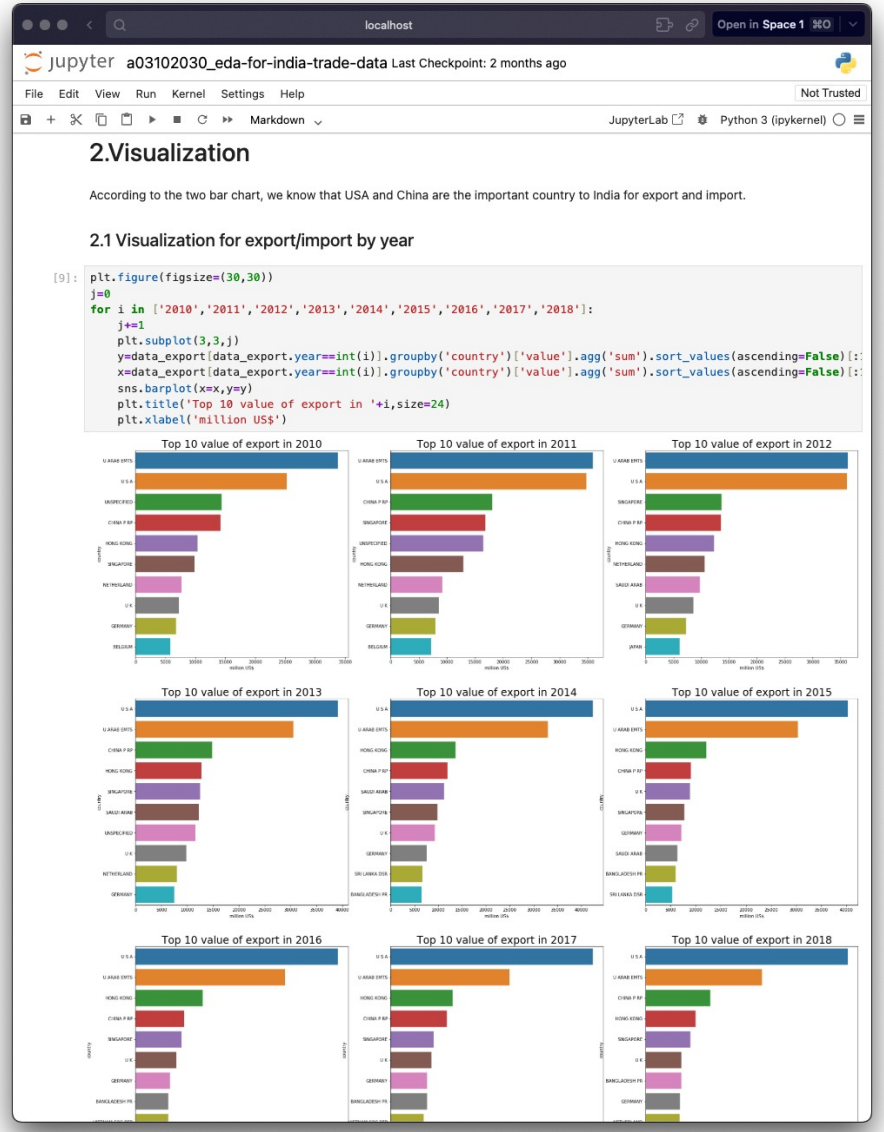


Notebooks

Markdown text

Python code

Code execution outputs



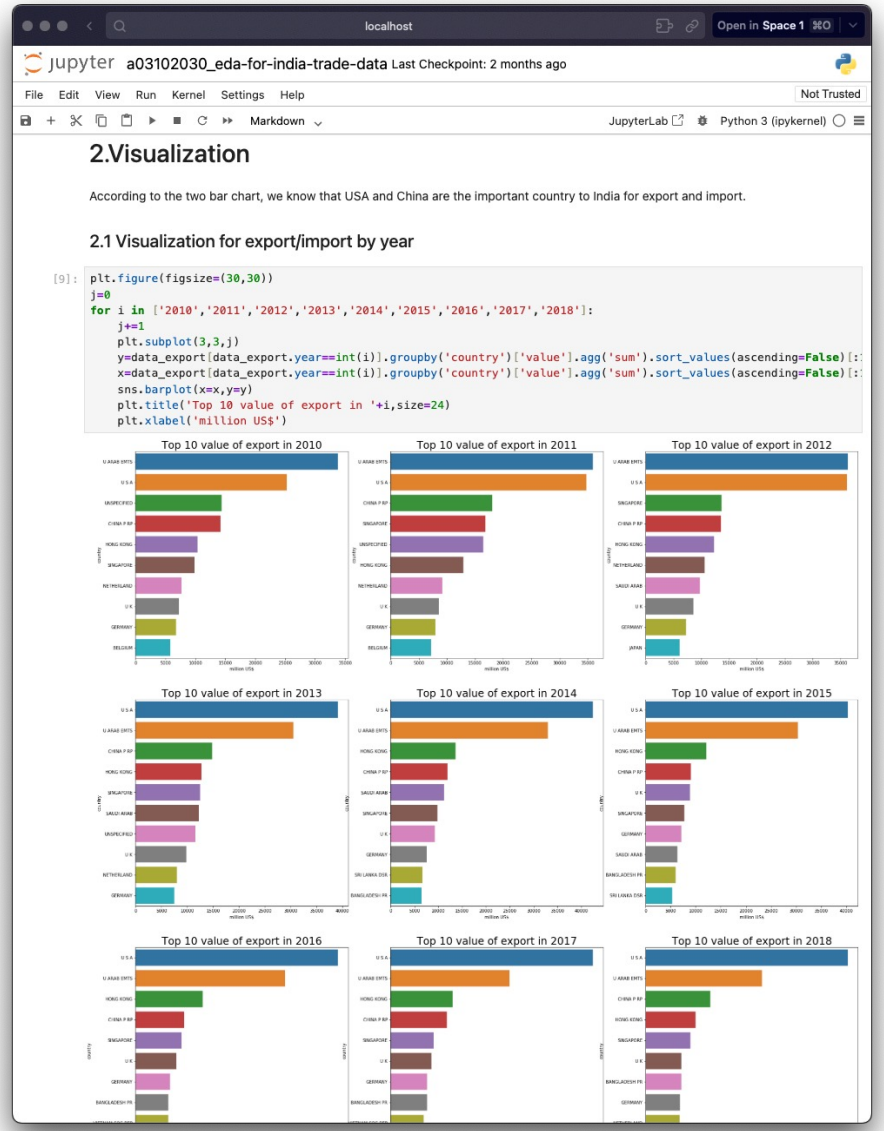
Notebooks benefits

- Accessible

- Default platform for ML development

- Centralise information

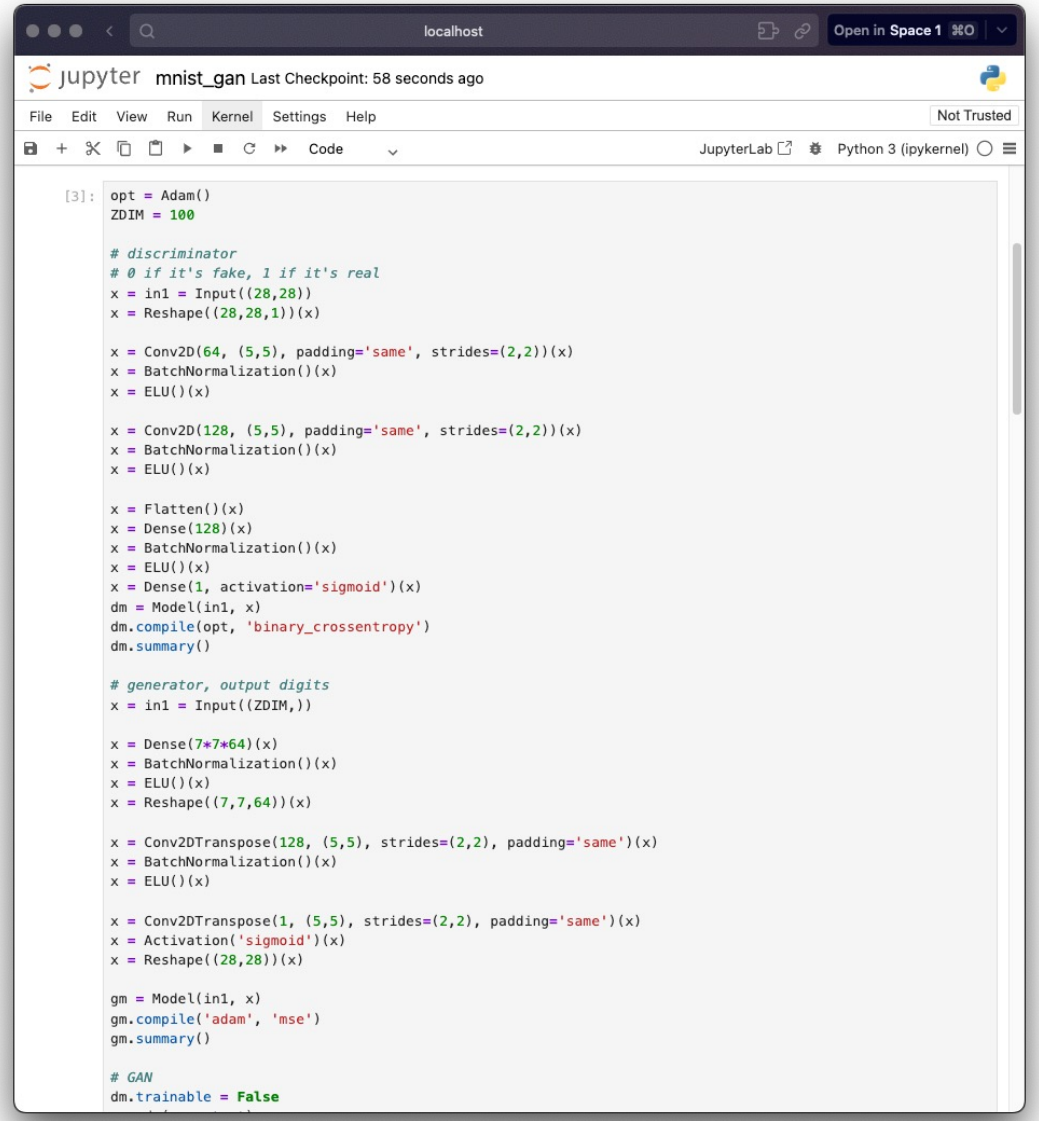
- Can be reused in whole or in part



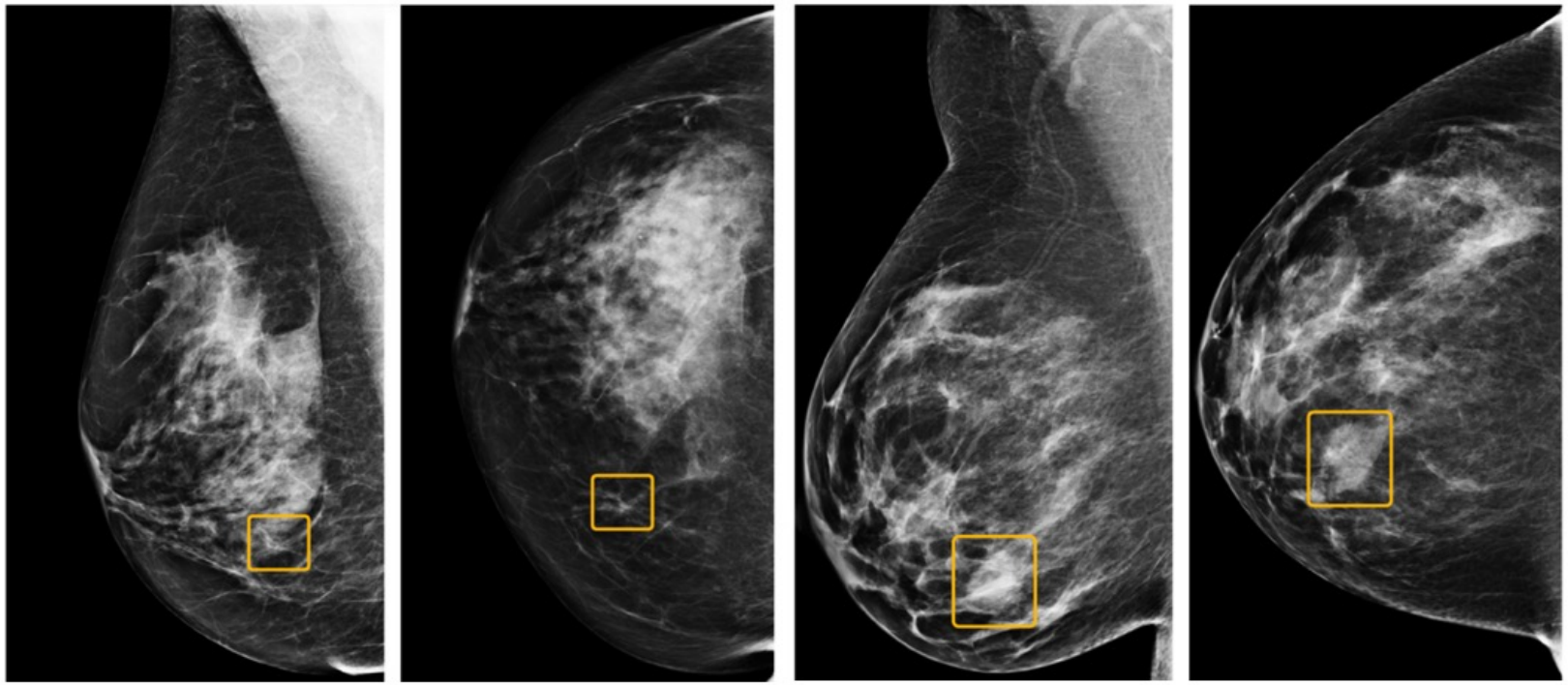
Notebooks drawbacks

- Used by people without Software Engineering knowledge

- Lack of understanding the underlying mechanisms



Machine learning (ML) usage



(A) A sample cancer case that was missed by all six readers in the US reader study, but correctly identified by the AI system

(B) A sample cancer case that was caught by all six readers in the US reader study, but missed by the AI system.

McKinney SM, Sieniek M, Godbole V, Godwin J, Antropova N, Ashrafian H, Back T, Chesus M, Corrado GC, Darzi A, Etemadi M, Garcia-Vicente F, Gilbert FJ, Halling-Brown M, Hassabis D, Jansen S, Karthikesalingam A, Kelly CJ, King D, Ledsam JR, Melnick D, Mostofi H, Peng L, Reicher JJ, Romera-Paredes B, Sidebottom R, Suleyman M, Tse D, Young KC, De Fauw J, Shetty S. International evaluation of an AI system for breast cancer screening. Nature. 2020 Jan;577(7788):89-94.

Multi-level need

1 **Python Level**

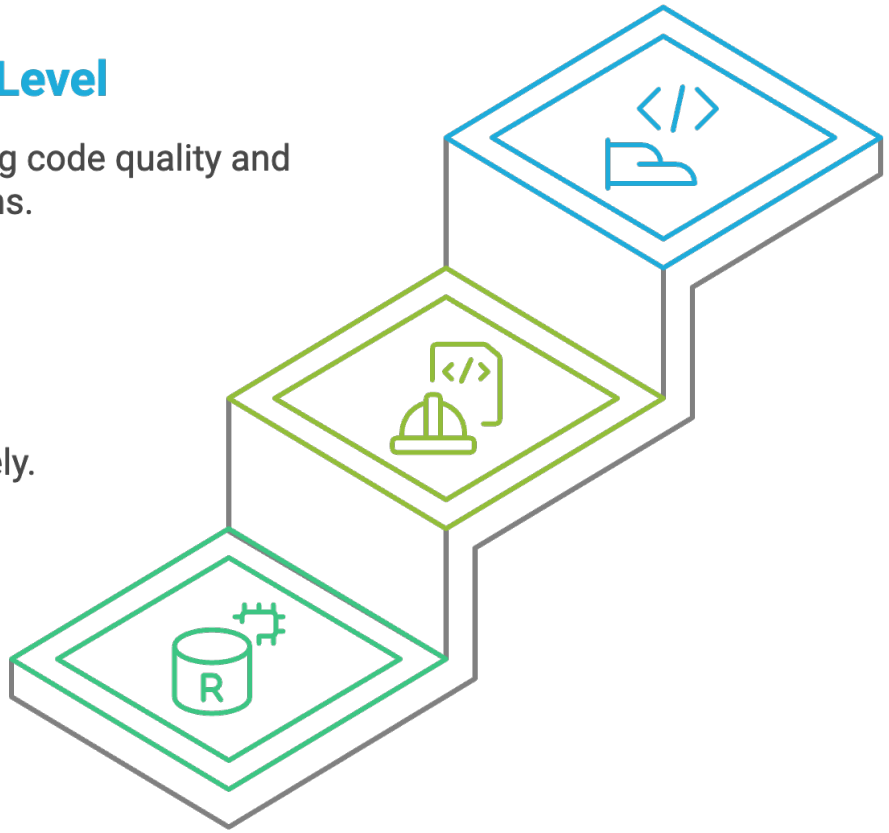
Maintaining code quality and conventions.

2 **Structural Level**

Organizing code and documentation effectively.

3 **Usage Practices**

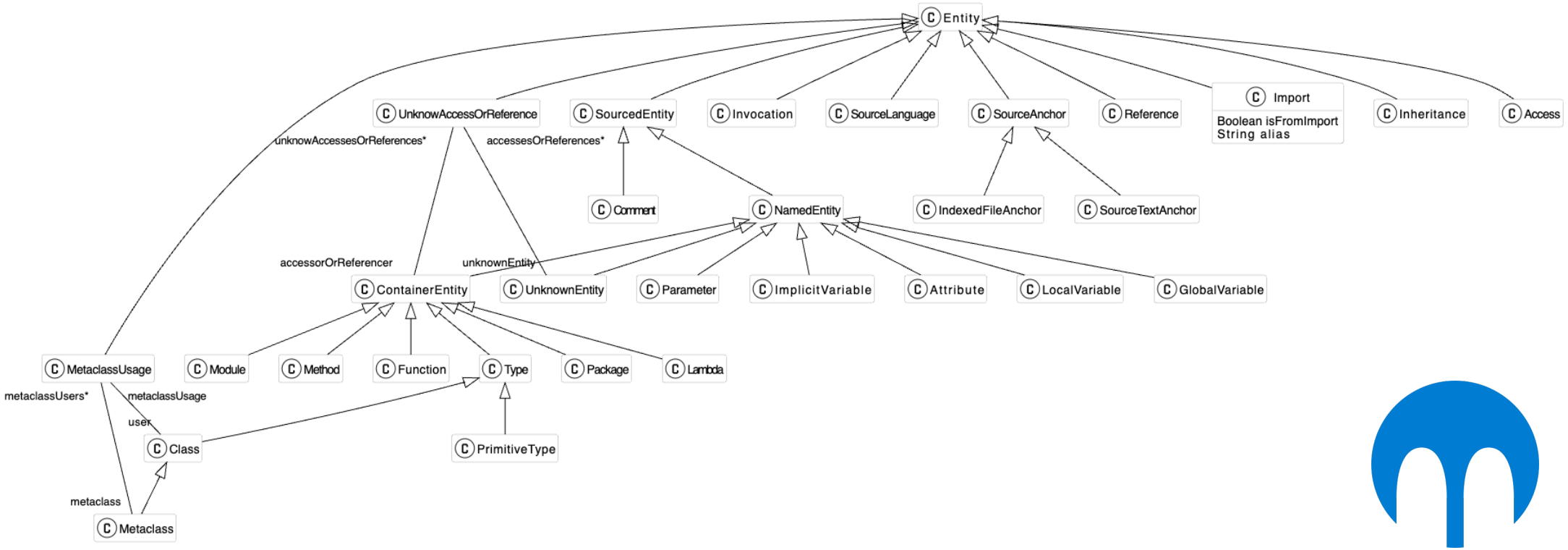
Enforce best practices for reproducibility and model specification.



Multi-level need – Existing tools

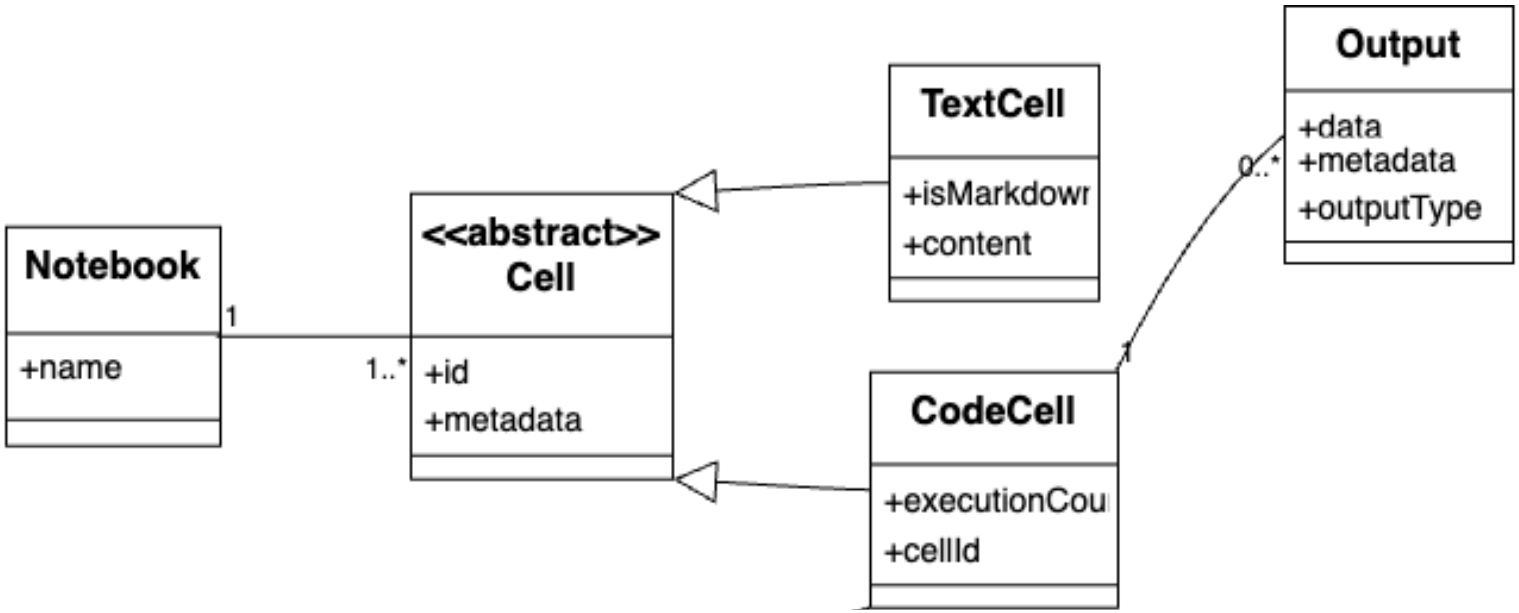
Tool	Focus	Nb support	Open-source
PyLint and Eq	Python	–	✓
SonarQube	Python/ ML	–	–
mllint	Project	✓	✓
Pynblint	Notebook	✓	✓
NBLyzer	Notebook	✓	✓
Pandera	Data	–	✓
ML Lint	Project	–	✓
Data Linter	Data	–	✓
Datalab	Data	–	–
Deepchecks	Model	–	–
Great Expect.	Data	–	–
DVC	Data	–	–
Pandas Prof.	Data	–	✓
Vespucci Linter	**All aspects**	✓	✓

Python metamodel

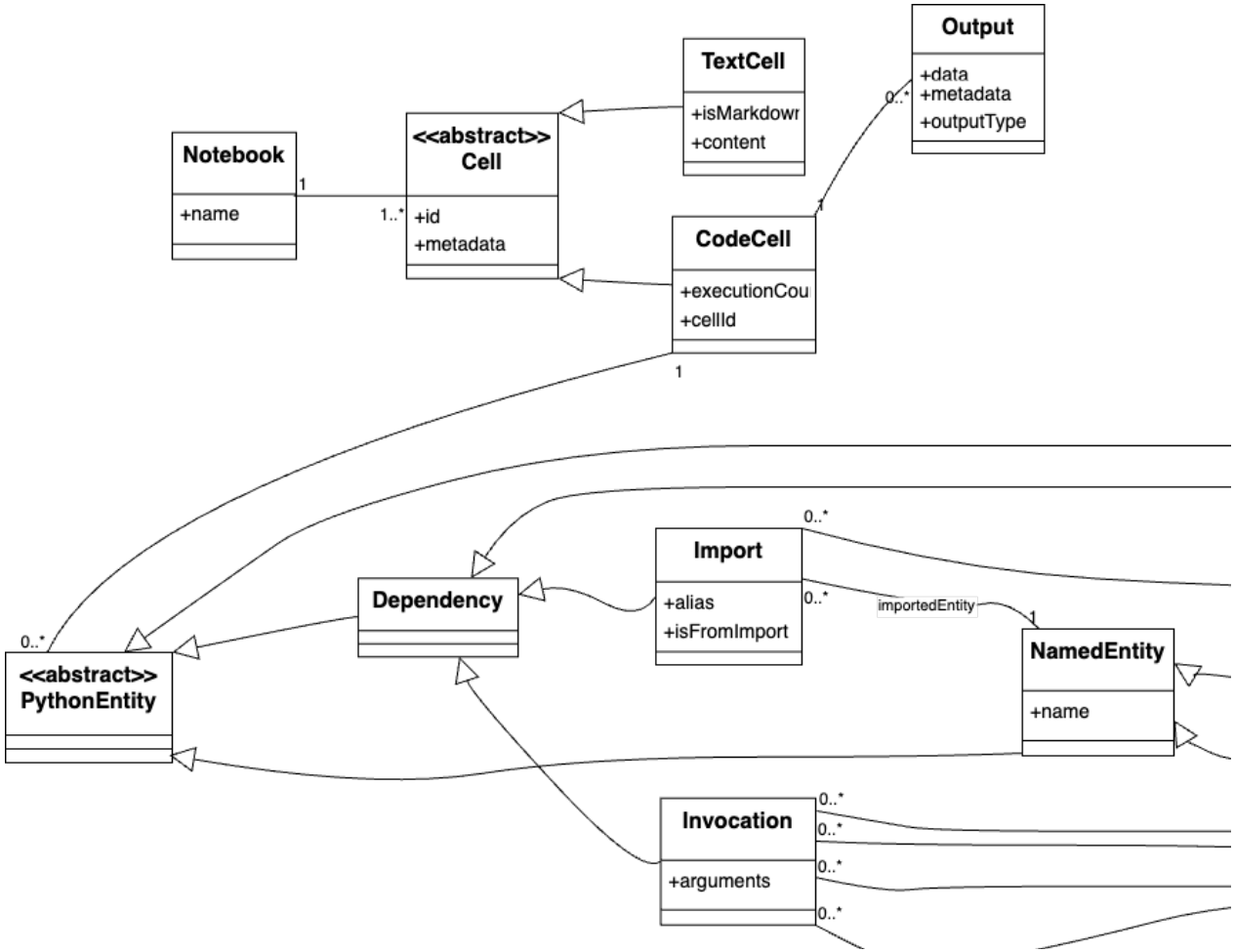


Famix

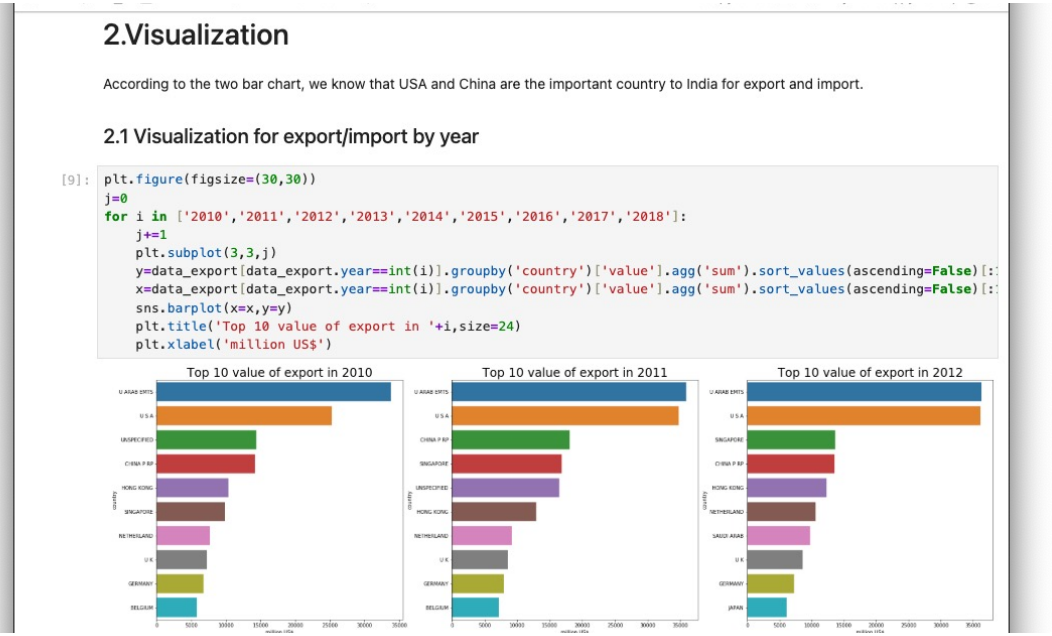
Notebook metamodel



Notebook metamodel

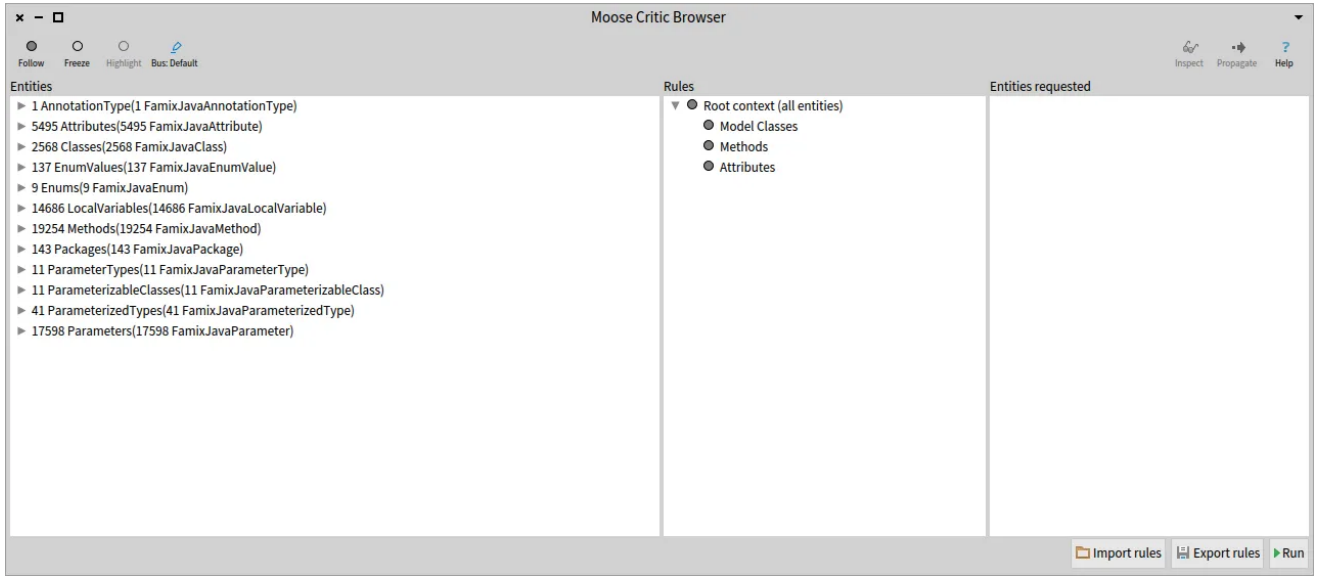


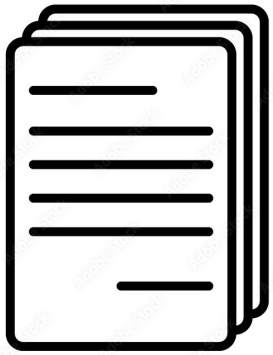
Notebook metamodel

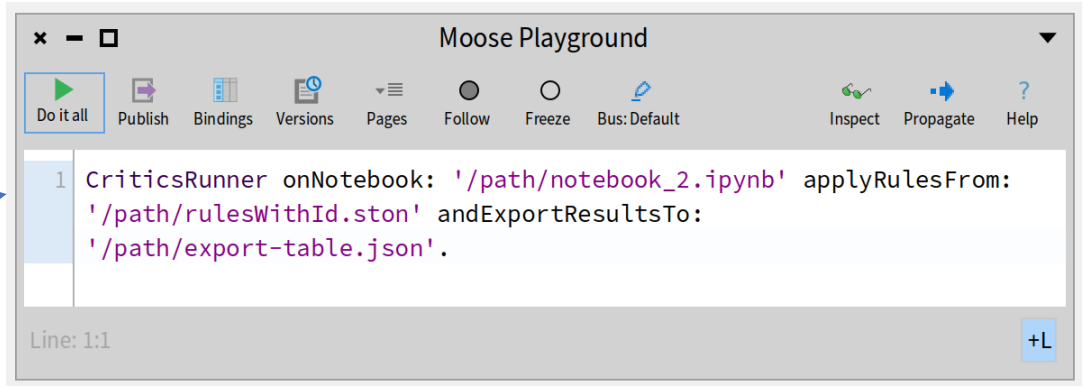


Rule engine

Moose Critic







Rules example

Python – Keep the code clean

Context :

All code cells;
All imports

Condition :

is reimported

Notebook – Enforce a modular design

Context :

All cells;

Condition :

Lines < 50

ML – Type inference error

Context :

All code cells;
read_csv() invocations

Condition :

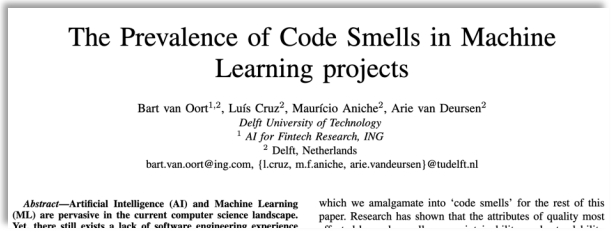
Presence of required parameters

Python rules – Literature extraction

Most common ML code violations detected by Pylint

- Error
- Convention
- Warning
- Refactor

Code	Description
C0103	invalid-name
C0303	trailing-whitespace
C0301	line-too-long
C0413	wrong-import-position
C0116	missing-function-docstring
W0611	unused-import
W0621	redefined-outer-name
W0104	pointless-statement
W0311	bad-indentation
W0404	reimported
R1705	no-else-return
R0913	too-many-arguments
R0914	too-many-locals
R0402	consider-using-from-import
R1732	consider-using-with

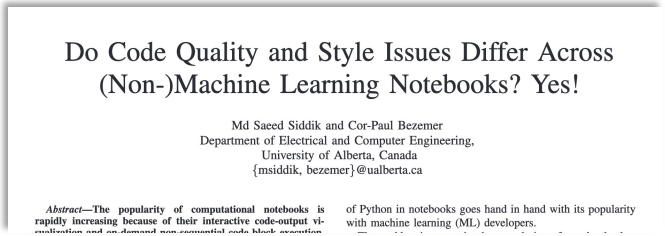


The Prevalence of Code Smells in Machine Learning projects

Bart van Oort[1,2], Luís Cruz[2], Maurício Aniche[2], Arie van Deursen[2]
Delft University of Technology
[1] *AI for Fintech Research, ING*
[2] Delft, Netherlands
bart.van.oort@ing.com, {l.cruz, m.f.aniche, arie.vandeursen}@tudelft.nl

***Abstract*—Artificial Intelligence (AI) and Machine Learning (ML) are pervasive in the current computer science landscape. Yet, there still exists a lack of software engineering experience**

which we amalgamate into 'code smells' for the rest of this paper. Research has shown that the attributes of quality most



Do Code Quality and Style Issues Differ Across (Non-)Machine Learning Notebooks? Yes!

Md Saeed Siddik and Cor-Paul Bezemer
Department of Electrical and Computer Engineering,
University of Alberta, Canada
{msiddik, bezemer}@ualberta.ca

***Abstract*—The popularity of computational notebooks is rapidly increasing because of their interactive code-output vi-**

of Python in notebooks goes hand in hand with its popularity with machine learning (ML) developers.

Python rules – Literature mint

	Code	Description	
Convention	C0103	invalid-name	←
	C0303	trailing-whitespace	
	C0301	line-too-long	
	C0413	wrong-import-position	
	C0116	missing-function-docstring	
Warning	W0611	unused-import	←
	W0621	redefined-outer-name	←
	W0104	pointless-statement	
	W0311	bad-indentation	
	W0404	reimported	←
Refactor	R1705	no-else-return	
	R0913	too-many-arguments	←
	R0914	too-many-locals	←
	R0402	consider-using-from-import	←
	R1732	consider-using-with	

Siddik, M. S., & Bezemer, C. P. (2023, October).
Do Code Quality and Style Issues Differ Across (Non-) Machine Learning Notebooks? Yes!.
In *2023 IEEE 23rd International Working Conference on Source Code Analysis and Manipulation (SCAM)* (pp. 72-83). IEEE.

Van Oort, B., Cruz, L., Aniche, M., & Van Deursen, A. (2021, May).
The prevalence of code smells in machine learning projects.
In *2021 IEEE/ACM 1st workshop on AI engineering-software engineering for AI (WAIN)* (pp. 1-8). IEEE.

Python rules – Literature mint

	Code	Description	
Convention	C0103	invalid-name	
	C0303	trailing-whitespace	←
	C0301	line-too-long	←
	C0413	wrong-import-position	←
	C0116	missing-function-docstring	←
Warning	W0611	unused-import	
	W0621	redefined-outer-name	
	W0104	pointless-statement	
	W0311	bad-indentation	←
	W0404	reimported	
Refactor	R1705	no-else-return	
	R0913	too-many-arguments	
	R0914	too-many-locals	
	R0402	consider-using-from-import	
	R1732	consider-using-with	

Siddik, M. S., & Bezemer, C. P. (2023, October).
Do Code Quality and Style Issues Differ Across (Non-) Machine Learning Notebooks? Yes!.
In *2023 IEEE 23rd International Working Conference on Source Code Analysis and Manipulation (SCAM)* (pp. 72-83). IEEE.

Van Oort, B., Cruz, L., Aniche, M., & Van Deursen, A. (2021, May).
The prevalence of code smells in machine learning projects.
In *2021 IEEE/ACM 1st workshop on AI engineering-software engineering for AI (WAIN)* (pp. 1-8). IEEE.

Python rules – Literature mint

	Code	Description
Convention	C0103	invalid-name
	C0303	trailing-whitespace
	C0301	line-too-long
	C0413	wrong-import-position
	C0116	missing-function-docstring
Warning	W0611	unused-import
	W0621	redefined-outer-name
	W0104	pointless-statement
	W0311	bad-indentation
	W0404	reimported
Refactor	R1705	no-else-return
	R0913	too-many-arguments
	R0914	too-many-locals
	R0402	consider-using-from-import
	R1732	consider-using-with

Siddik, M. S., & Bezemer, C. P. (2023, October).
Do Code Quality and Style Issues Differ Across (Non-) Machine Learning Notebooks? Yes!.
In *2023 IEEE 23rd International Working Conference on Source Code Analysis and Manipulation (SCAM)* (pp. 72-83). IEEE.

Van Oort, B., Cruz, L., Aniche, M., & Van Deursen, A. (2021, May).
The prevalence of code smells in machine learning projects.
In *2021 IEEE/ACM 1st workshop on AI engineering-software engineering for AI (WAIN)* (pp. 1-8). IEEE.

Python rules

Script

```
a = 1 + 1
a
```

pointless-statement / W0104

Notebook

```
[1]:
a = 1 + 1
a

[1]:
2
```

PyLint rule used when a statement doesn't have (or at least seems to) any effect.

Python rules

Table 1
General Python Linting Rules Implemented in Vespucci

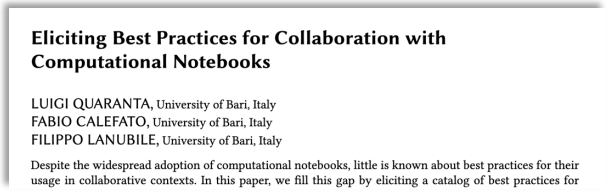
Rule Name	Description	Motivation
P1 - Unused variables	Identifies variables that are assigned but never used.	Helps remove dead code and clarify the intent.
P2 - Variables reassignments	Detects repeated use of the same variable name after initial assignment.	Prevents confusion and potential bugs in notebooks.
P3 - Variables naming	Enforces a minimum variable name length (more than 3 characters, except for ML conventions like X, y).	Promotes readability and avoids ambiguous identifiers.
P4 - Too many parameters	Detects functions with more than 5 parameters.	Encourages simpler, modular function design.
P5 - Consider using from import	Recommends using `from module import x` instead of importing the full module.	Improves clarity and avoid namespace pollution
P6 - Unused import	Identifies unused imports with an exception for the star import (`from module import *`) where we can not statically determine which symbols are actually used in the code.	Improves clarity.
P7 - Reimported	Identifies redundant imports of the same module.	Reduces code clutter and redundancy.
P8 - Too many local	Detects functions that declare more than 11 local variables.	Indicates high complexity.
P9 - Global variables in function	Warns against the use of global variables within functions.	Prevents hidden dependencies and side effects.

Notebook rules

Derived from

best practices

existing tools



Eliciting Best Practices for Collaboration with Computational Notebooks

LUIGI QUARANTA, University of Bari, Italy
FABIO CALEFATO, University of Bari, Italy
FILIPPO LANUBILE, University of Bari, Italy

Despite the widespread adoption of computational notebooks, little is known about best practices for their usage in collaborative contexts. In this paper, we fill this gap by eliciting a catalog of best practices for

PLOS COMPUTATIONAL BIOLOGY

EDITORIAL

Ten simple rules for writing and sharing computational analyses in Jupyter Notebooks

Adam Rule[1], Amanda Birmingham[2], Cristal Zuniga[3], Ilkay Altintas[4], Shih-Cheng Huang[5*], Rob Knight[3,5], Niema Moshiri[6], Mai H. Nguyen[6], Sara Brin Rosenthal[7], Fernando Pérez[8], Peter W. Rose[4*]

1 Design Lab, UC San Diego, La Jolla, California, United States of America, 2 Center for Computational

Table 2
Notebook-Level Linting Rules Implemented in Vespucci

Rule Name	Description	Motivation
N1 - Notebook naming	Enforces the use of filenames longer than 2 characters (*N1.1*) with only alphanumeric characters, hyphens, or underscores (*N1.2*). Detect default names containing `Untitled` (*N1.3*).	Improves clarity, navigation, and compatibility across systems. Prevents confusion during sharing or version control.
N2 - Version control	Requires displaying library versioning information with `package.__version__` or using `watermark`[15].	Ensures reproducibility and helps diagnose variances across executions or environments.
N3 - Imports at the top	Enforces grouping all import statements in the first code cell.	Enhances readability and makes dependencies explicit for reproducibility.
N4 - Long code cells	Detect cells exceeding 30 lines of code.	Encourages modular design, improves readability.
N5 - Empty code cells	Detects code cells that contain no code.	Avoids unnecessary visual clutter and potential confusion.
N6 - Missing text cells	Checks for absence of markdown cells providing context or explanation.	Promotes documentation and guides readers through the workflow.
N7 - Notebook too long	Detect notebooks with more than 50 code cells.	Notebooks should remain reasonably short and focused.
N8 - Non-linear execution	Detects out-of-order execution based on cell execution counts (if present).	Ensuring a linear execution order helps maintain a clear, predictable workflow and supports reliable re-execution of the notebook from top to bottom.

ML rules

Hyperparameter not Explicitly Set

[1] :

```
### Scikit-Learn
from sklearn.cluster import KMeans
- kmeans = KMeans()
+ kmeans = KMeans(n_clusters=8, random_state=0)
```

Randomness Uncontrolled

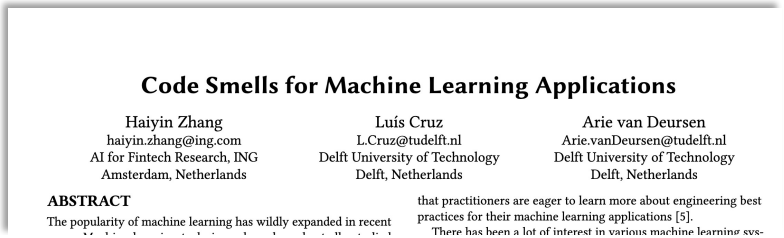
[1] :

```
### Scikit-Learn
from sklearn.model_selection import KFold
+ rng = 0
- kf = KFold(random_state=None)
+ kf = KFold(random_state=rng)
```

ML rules

Table 3
ML-Specific Linting Rules Implemented in Vespucci

Rule Name	Description	Motivation
M1 - Uncontrolled randomness	Detects missing random seed parameters in ML-related functions (e.g., train/test split, model initialization).	Ensures reproducibility of results and consistency across runs.
M2 - In-place API misuse	Warns when functions like `dropna()` are used without assigning the result.	Prevents logic errors due to misunderstanding between in-place modification and returning copies.
M3 - Implicit hyperparameters	Detects ML model instantiations where key hyperparameters are not explicitly set.	Enhances transparency, reproducibility, and adaptability to future library changes.
M4 - Columns and dtypes not set	Detects when columns (*M4.1*) and datatypes (*M4.2*) are not explicitly specified during data loading.	Improves data integrity, parsing performance, and avoids unintended type inference.
M5 - Merge without explicit parameters	Detects DataFrame merges where parameters `on`, `how` (*M5.1*), or `validate` (*M5.2*) are not specified.	Reduces risk of ambiguous or incorrect merges and improves code clarity.



Code Smells for Machine Learning Applications

Haiyin Zhang
haiyin.zhang@ing.com
AI for Fintech Research, ING
Amsterdam, Netherlands

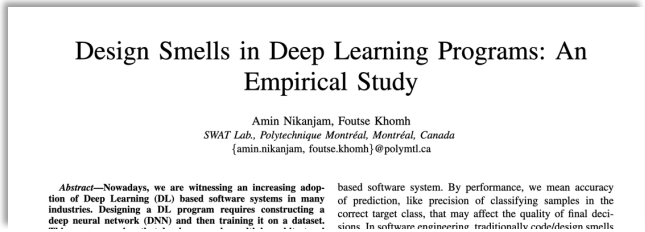
Luís Cruz
L.Cruz@tudelft.nl
Delft University of Technology
Delft, Netherlands

Arie van Deursen
Arie.vanDeursen@tudelft.nl
Delft University of Technology
Delft, Netherlands

ABSTRACT

The popularity of machine learning has wildly expanded in recent

that practitioners are eager to learn more about engineering best practices for their machine learning applications [5].



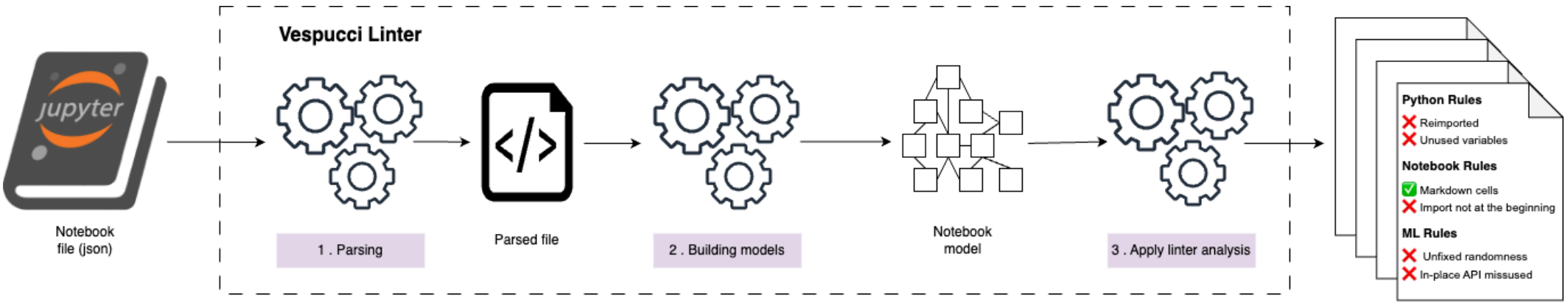
Design Smells in Deep Learning Programs: An Empirical Study

Amin Nikanjam, Foutse Khomh
SWAT Lab., Polytechnique Montréal, Montréal, Canada
{amin.nikanjam, foutse.khomh}@polymtl.ca

***Abstract*—Nowadays, we are witnessing an increasing adoption of Deep Learning (DL) based software systems in many industries. Designing a DL program requires constructing a deep neural network (DNN) and then training it on a dataset.**

based software system. By performance, we mean accuracy of prediction, like precision of classifying samples in the correct target class, that may affect the quality of final deci-

Vespucci linter process overview



Results

- 24 rules

- 3 levels : Python, Notebook, ML

- Analysis on 5000 notebooks

kaggle™

rule_name	level	num_violations	num_notebooks	pct_notebooks	violations_per_affected_nb
N3 - Imports at the top	Notebook	15527	1931	38.6 %	8.04
P2 - Variables reassignments	Python	14767	2874	57.5 %	5.14
P6 - Unused import	Python	8476	2987	59.7 %	2.84
M4.1 - read_csv missing usecols param	ML	6275	3330	66.6 %	1.88
N4 - Long code cells	Notebook	6235	1949	39.0 %	3.20
M4.2 - read_csv missing dtype param	ML	6226	3313	66.3 %	1.88
P1 - Unused variables	Python	5180	2492	49.8 %	2.08
N2 - Version control	Notebook	4951	4951	99.0 %	1.00
P3 - Variables naming	Python	4681	1772	35.4 %	2.64
M1 - Uncontrolled randomness	ML	3343	1002	20.0 %	3.34
P7 - Reimported	Python	2746	872	17.4 %	3.15
M3 - Implicit hyperparameters	ML	2552	1014	20.3 %	2.52
M2 - In-place API misuse	ML	2511	646	12.9 %	3.89
N5 - Empty code cells	Notebook	1968	1004	20.1 %	1.96
P9 - Global variables in function	Python	1103	580	11.6 %	1.90
N6 - Missing text cells	Notebook	904	904	18.1 %	1.00
M5.2 - Merge parameter validate should be explicit	ML	689	218	4.4 %	3.16
M5.1 - Merge parameters should be explicit	ML	299	129	2.6 %	2.32
N7 - Notebook too long	Notebook	251	251	5.0 %	1.00
P4 - Too many parameters	Python	176	125	2.5 %	1.41
P5 - Consider using from import	Python	159	130	2.6 %	1.22
N8 - Non-linear execution	Notebook	154	154	3.1 %	1.00
P8 - Too many local	Python	147	115	2.3 %	1.28
N1.2 - Non portable chars in nb name	Notebook	1	1	0.0 %	1.00

Results

- 24 rules

- 3 levels : Python, Notebook, ML

- Analysis on 5000 notebooks

kaggle™

rule_name	level	num_violations	num_notebooks	pct_notebooks	violations_per_affected_nb
N3 - Imports at the top	Notebook	15527	1931	38.6 %	8.04
P2 - Variables reassignments	Python	14767	2874	57.5 %	5.14
P6 - Unused import	Python	8476	2987	59.7 %	2.84
M4.1 - read_csv missing usecols param	ML	6275	3330	66.6 %	1.88
N4 - Long code cells	Notebook	6235	1949	39.0 %	3.20
M4.2 - read_csv missing dtype param	ML	6226	3313	66.3 %	1.88
P1 - Unused variables	Python	5180	2492	49.8 %	2.08
N2 - Version control	Notebook	4951	4951	99.0 %	1.00
P3 - Variables naming	Python	4681	1772	35.4 %	2.64
M1 - Uncontrolled randomness	ML	3343	1002	20.0 %	3.34
P7 - Reimported	Python	2746	872	17.4 %	3.15
M3 - Implicit hyperparameters	ML	2552	1014	20.3 %	2.52
M2 - In-place API misuse	ML	2511	646	12.9 %	3.89
N5 - Empty code cells	Notebook	1968	1004	20.1 %	1.96
P9 - Global variables in function	Python	1103	580	11.6 %	1.90
N6 - Missing text cells	Notebook	904	904	18.1 %	1.00
M5.2 - Merge parameter validate should be explicit	ML	689	218	4.4 %	3.16
M5.1 - Merge parameters should be explicit	ML	299	129	2.6 %	2.32
N7 - Notebook too long	Notebook	251	251	5.0 %	1.00
P4 - Too many parameters	Python	176	125	2.5 %	1.41
P5 - Consider using from import	Python	159	130	2.6 %	1.22
N8 - Non-linear execution	Notebook	154	154	3.1 %	1.00
P8 - Too many local	Python	147	115	2.3 %	1.28
N1.2 - Non portable chars in nb name	Notebook	1	1	0.0 %	1.00

Results

- 24 rules

- 3 levels : Python, Notebook, ML

- Analysis on 5000 notebooks

kaggle

rule_name	level	num_violations	num_notebooks	pct_notebooks	violations_per_affected_nb
N3 - Imports at the top	Notebook	15527	1931	38.6 %	8.04
P2 - Variables reassignments	Python	14767	2874	57.5 %	5.14
P6 - Unused import	Python	8476	2987	59.7 %	2.84
M4.1 - read_csv missing usecols param	ML	6275	3330	66.6 %	1.88
N4 - Long code cells	Notebook	6235	1949	39.0 %	3.20
M4.2 - read_csv missing dtype param	ML	6226	3313	66.3 %	1.88
P1 - Unused variables	Python	5180	2492	49.8 %	2.08
N2 - Version control	Notebook	4951	4951	99.0 %	1.00
P3 - Variables naming	Python	4681	1772	35.4 %	2.64
M1 - Uncontrolled randomness	ML	3343	1002	20.0 %	3.34
P7 - Reimported	Python	2746	872	17.4 %	3.15
M3 - Implicit hyperparameters	ML	2552	1014	20.3 %	2.52
M2 - In-place API misuse	ML	2511	646	12.9 %	3.89
N5 - Empty code cells	Notebook	1968	1004	20.1 %	1.96
P9 - Global variables in function	Python	1103	580	11.6 %	1.90
N6 - Missing text cells	Notebook	904	904	18.1 %	1.00
M5.2 - Merge parameter validate should be explicit	ML	689	218	4.4 %	3.16
M5.1 - Merge parameters should be explicit	ML	299	129	2.6 %	2.32
N7 - Notebook too long	Notebook	251	251	5.0 %	1.00
P4 - Too many parameters	Python	176	125	2.5 %	1.41
P5 - Consider using from import	Python	159	130	2.6 %	1.22
N8 - Non-linear execution	Notebook	154	154	3.1 %	1.00
P8 - Too many local	Python	147	115	2.3 %	1.28
N1.2 - Non portable chars in nb name	Notebook	1	1	0.0 %	1.00

Results

- 24 rules

- 3 levels : Python, Notebook, ML

- Analysis on 5000 notebooks

rule_name	level	num_violations	num_notebooks	pct_notebooks	violations_per_affected_nb
N3 - Imports at the top	Notebook	15527	1931	38.6 %	8.04
P2 - Variables reassignments	Python	14767	2874	57.5 %	5.14
P6 - Unused import	Python	8476	2987	59.7 %	2.84
M4.1 - read_csv missing usecols param	ML	6275	3330	66.6 %	1.88
N4 - Long code cells	Notebook	6235	1949	39.0 %	3.20
M4.2 - read_csv missing dtype param	ML	6226	3313	66.3 %	1.88
P1 - Unused variables	Python	5180	2492	49.8 %	2.08
N2 - Version control	Notebook	4951	4951	99.0 %	1.00
P3 - Variables naming	Python	4681	1772	35.4 %	2.64
M1 - Uncontrolled randomness	ML	3343	1002	20.0 %	3.34
P7 - Reimported	Python	2746	872	17.4 %	3.15
M3 - Implicit hyperparameters	ML	2552	1014	20.3 %	2.52
M2 - In-place API misuse	ML	2511	646	12.9 %	3.89
N5 - Empty code cells	Notebook	1968	1004	20.1 %	1.96
P9 - Global variables in function	Python	1103	580	11.6 %	1.90
N6 - Missing text cells	Notebook	904	904	18.1 %	1.00
M5.2 - Merge parameter validate should be explicit	ML	689	218	4.4 %	3.16
M5.1 - Merge parameters should be explicit	ML	299	129	2.6 %	2.32
N7 - Notebook too long	Notebook	251	251	5.0 %	1.00
P4 - Too many parameters	Python	176	125	2.5 %	1.41
P5 - Consider using from import	Python	159	130	2.6 %	1.22
N8 - Non-linear execution	Notebook	154	154	3.1 %	1.00
P8 - Too many local	Python	147	115	2.3 %	1.28
N1.2 - Non portable chars in nb name	Notebook	1	1	0.0 %	1.00

kaggle

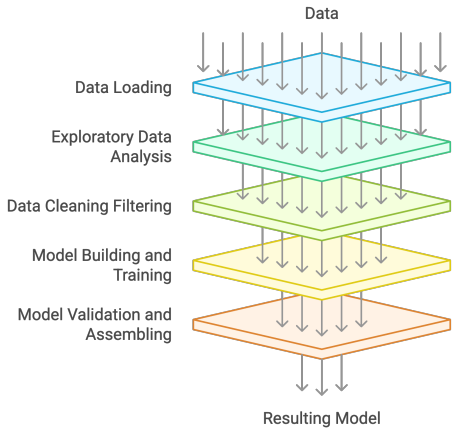
Future work (short term)

- Linter in notebooks/IDE using LSP

- MoTion usage for complex context queries

- Semantic rules

Future work (long term)



1 **Python Level**

Maintaining code quality and conventions.

2 **Structural Level**

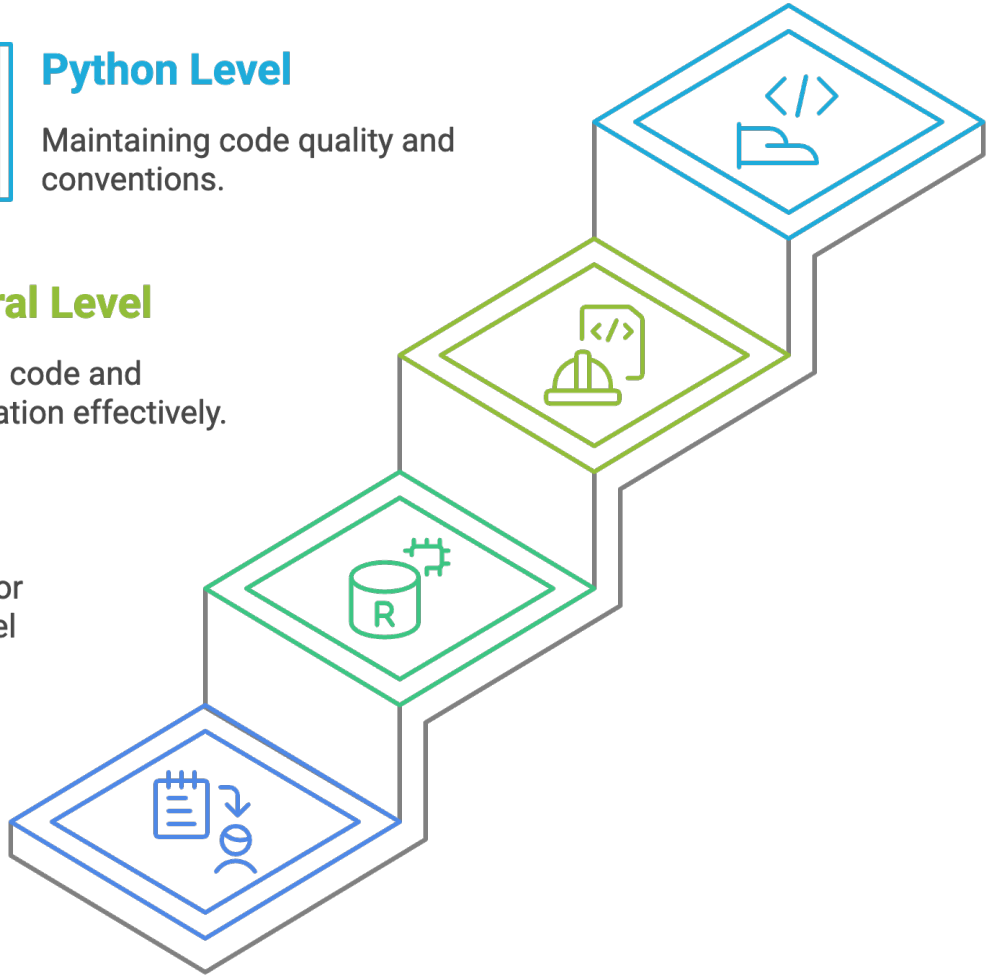
Organizing code and documentation effectively.

3 **Usage Practices**

Enforce best practices for reproducibility and model specification.

4 **Workflow Level**

Ensuring a logical sequence of steps (workflow) in the notebook.



Conclusion

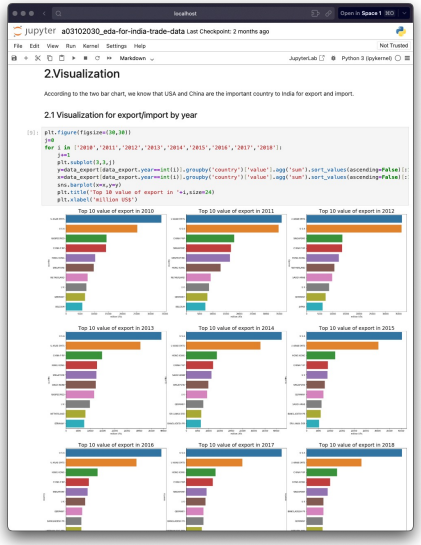


Table 1
General Python Linting Rules Implemented in Vespucci

Rule Name	Description	Motivation
P1 - Unused variables	Identifies variables that are assigned but never used.	Helps remove dead code and clarify the intent.
P2 - Variables reassignments	Detects repeated use of the same variable name after initial assignment.	Prevents confusion and potential bugs in notebooks.
P3 - Variables naming	Enforces a minimum variable name length (more than 3 characters, except for ML conventions like X, y).	Promotes readability and avoids ambiguous identifiers.
P4 - Too many parameters	Detects functions with more than 5 parameters.	Encourages simpler, modular function design.
P5 - Consider using from import	Recommends using `from module import x` instead of importing the full module.	Improves clarity and avoid namespace pollution
P6 - Unused import	Identifies unused imports with an exception for the star import (`from module import *`) where we can not statically determine which symbols are actually used in the code.	Improves clarity.
P7 - Reimported	Identifies redundant imports of the same module.	Reduces code clutter and redundancy.
P8 - Too many local	Detects functions that declare more than 11 local variables.	Indicates high complexity.
P9 - Global variables in function	Warns against the use of global variables within functions.	Prevents hidden dependencies and side effects.

rule_name	level	num_violations	num_notebooks	pct_notebooks	violations_per_affected_nb
N3 - Imports at the top	Notebook	15527	1931	38.6 %	8.04
P2 - Variables reassignments	Python	14767	2874	57.5 %	5.14
P6 - Unused import	Python	8476	2987	59.7 %	2.84
M4.1 - read_csv missing usecols param	ML	6275	3330	66.6 %	1.88
N4 - Long code cells	Notebook	6235	1949	39.0 %	3.20
M4.2 - read_csv missing dtype param	ML	6226	3313	66.3 %	1.88
P1 - Unused variables	Python	5180	2492	49.8 %	2.08
N2 - Version control	Notebook	4951	4951	99.0 %	1.00
P3 - Variables naming	Python	4681	1772	35.4 %	2.64
M1 - Uncontrolled randomness	ML	3343	1002	20.0 %	3.34
P7 - Reimported	Python	2746	872	17.4 %	3.15
M3 - Implicit hyperparameters	ML	2552	1014	20.3 %	2.52
M2 - In-place API misuse	ML	2511	646	12.9 %	3.89
N5 - Empty code cells	Notebook	1968	1004	20.1 %	1.96
P9 - Global variables in function	Python	1103	580	11.6 %	1.90
N6 - Missing text cells	Notebook	904	904	18.1 %	1.00
M5.2 - Merge parameter validate should be explicit	ML	689	218	4.4 %	3.16
M5.1 - Merge parameters should be explicit	ML	299	129	2.6 %	2.32
N7 - Notebook too long	Notebook	251	251	5.0 %	1.00
P4 - Too many parameters	Python	176	125	2.5 %	1.41
P5 - Consider using from import	Python	159	130	2.6 %	1.22
N8 - Non-linear execution	Notebook	154	154	3.1 %	1.00
P8 - Too many local	Python	147	115	2.3 %	1.28
N1.2 - Non portable chars in nb name	Notebook	1	1	0.0 %	1.00

Marius Mignard : marius.mignard@inria.fr