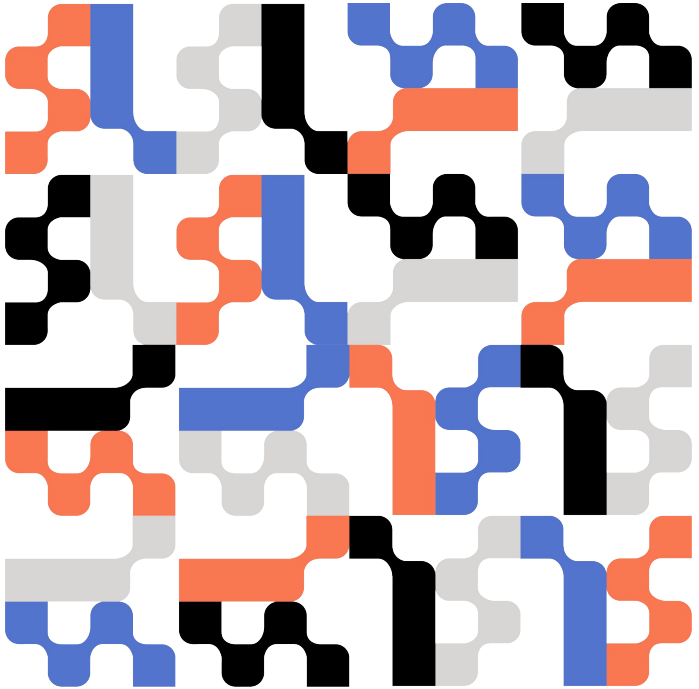
MODUS CREATE | TWEAG by Modus Create

FawltyDeps

Finding **undeclared** and **unused** dependencies in your notebooks and projects



Johan Herland – @jherland

FOSDEM 2025

WIKIPEDIA
The Free Encyclopedia

Search Wikipedia | Search

Replication crisis

18 languages

Contents [hide]

(Top)

- Background
- Prevalence
- Causes
- Consequences
- Remedies
- See also
- Notes
- References
- Further reading

WIKIPEDIA
The Free Encyclopedia

Search Wikipedia | Search

Data science

46 languages

Article | Talk | Read | Edit | View history | Tools

Contents [hide]

(Top)

- Foundations
  - Relationship to statistics
- Etymology
  - Early usage
  - Modern usage
- Data Science and Data Analysis
- History
- See also
- References

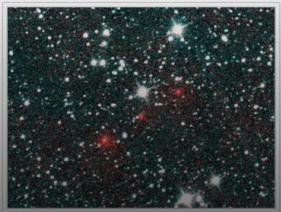
From Wikipedia, the free encyclopedia

Not to be confused with information science.

Data science is an interdisciplinary academic field[1] that uses statistics, scientific computing, scientific methods, processes, algorithms and systems to extract or extrapolate knowledge and insights from noisy, structured, and unstructured data.[2]

Data science also integrates domain knowledge from the underlying application domain (e.g., natural sciences, information technology, and medicine).[3] Data science is multifaceted and can be described as a science, a research paradigm, a research method, a discipline, a workflow, and a profession.[4]

Data science is a "concept to unify statistics, data analysis, informatics, and their related methods" to "understand and analyze actual phenomena" with data.[5] It uses techniques and theories drawn from many fields within the context of mathematics, statistics, computer science, information science, and domain knowledge.[6] However, data science is different from computer science and information science. Turing Award winner Jim Gray imagined data science as a "fourth paradigm" of science (empirical, theoretical, computational, and now data-driven) and asserted that "everything about science is changing because of the impact of information technology" and the data deluge.[7][8]



The existence of Comet NEOWISE (here depicted as a series of red dots) was discovered by analyzing astronomical survey data acquired by a space telescope, the Wide-field Infrared Survey Explorer.

A **data scientist** is a professional who creates programming code and combines it with statistical knowledge to create insights from data.[9]

Foundations [edit]

PAPER

https://arxiv.org/abs/2308.07333

Computational reproducibility of Jupyter notebooks from biomedical publications

Sheeba Samuel[1,2*†] and Daniel Mietchen[3,4,5†‡]

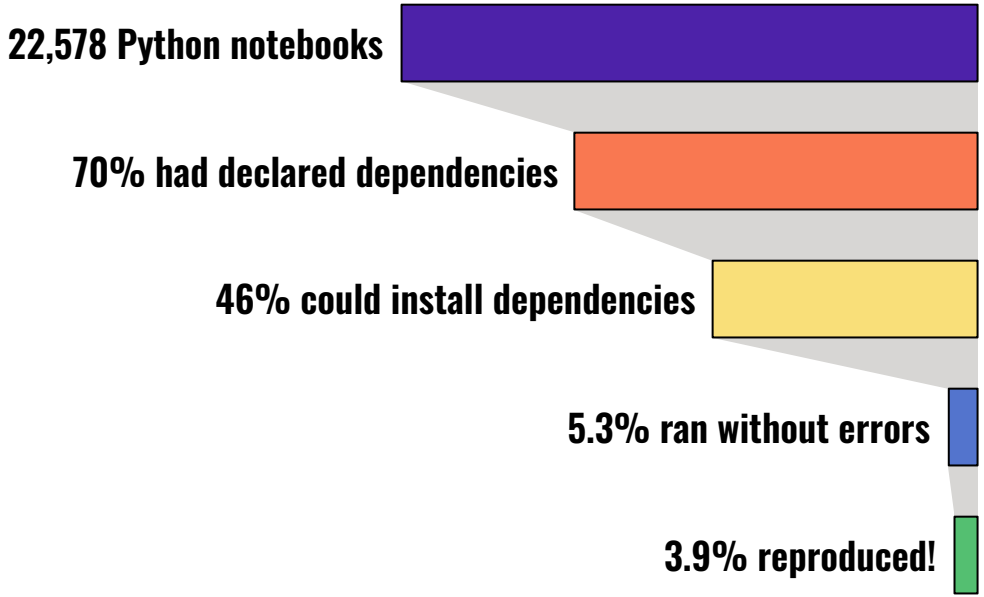
[1]Heinz-Nixdorf Chair for Distributed Information Systems, Friedrich Schiller University Jena, Germany and [2]Michael Stifel Center Jena, Germany and [3]Ronin Institute, Montclair, New Jersey, United States and [4]Institute for Globally Distributed Open Research and Education (IGDORE) and [5]FIZ Karlsruhe — Leibniz Institute for Information Infrastructure, Berlin, Germany

[*]sheeba.samuel@uni-jena.de
[†]These authors contributed equally to this work
[‡]daniel.mietchen@ronininstitute.org

Abstract

Background Jupyter notebooks facilitate the bundling of executable code with its documentation and output in one interactive environment, and they represent a popular mechanism to document and share computational workflows, including for research publications. The reproducibility of computational aspects of research is a key component of scientific reproducibility but has not yet been assessed at scale for Jupyter notebooks associated with biomedical publications.

Approach We address computational reproducibility at two levels: (1) Using fully automated workflows, we analyzed the computational reproducibility of Jupyter notebooks associated with publications indexed in the biomedical literature repository PubMed Central. We identified such notebooks by mining the article's full text, trying to locate them on GitHub and attempting to



Results Out of 27,271 Jupyter notebooks from 2,660 GitHub repositories associated with 3,467 publications, 22,578 notebooks were written in Python, including 15,817 that had their dependencies declared in standard requirement files and that we attempted to re-run automatically. For 10,388 of these, all declared dependencies could be installed successfully, and we re-ran them to assess reproducibility. Of these, 1,203 notebooks ran through without any errors, including 879 that produced results identical to those reported in the original notebook, and 324 for which our results differed from the originally reported ones. Running the other notebooks resulted in exceptions.

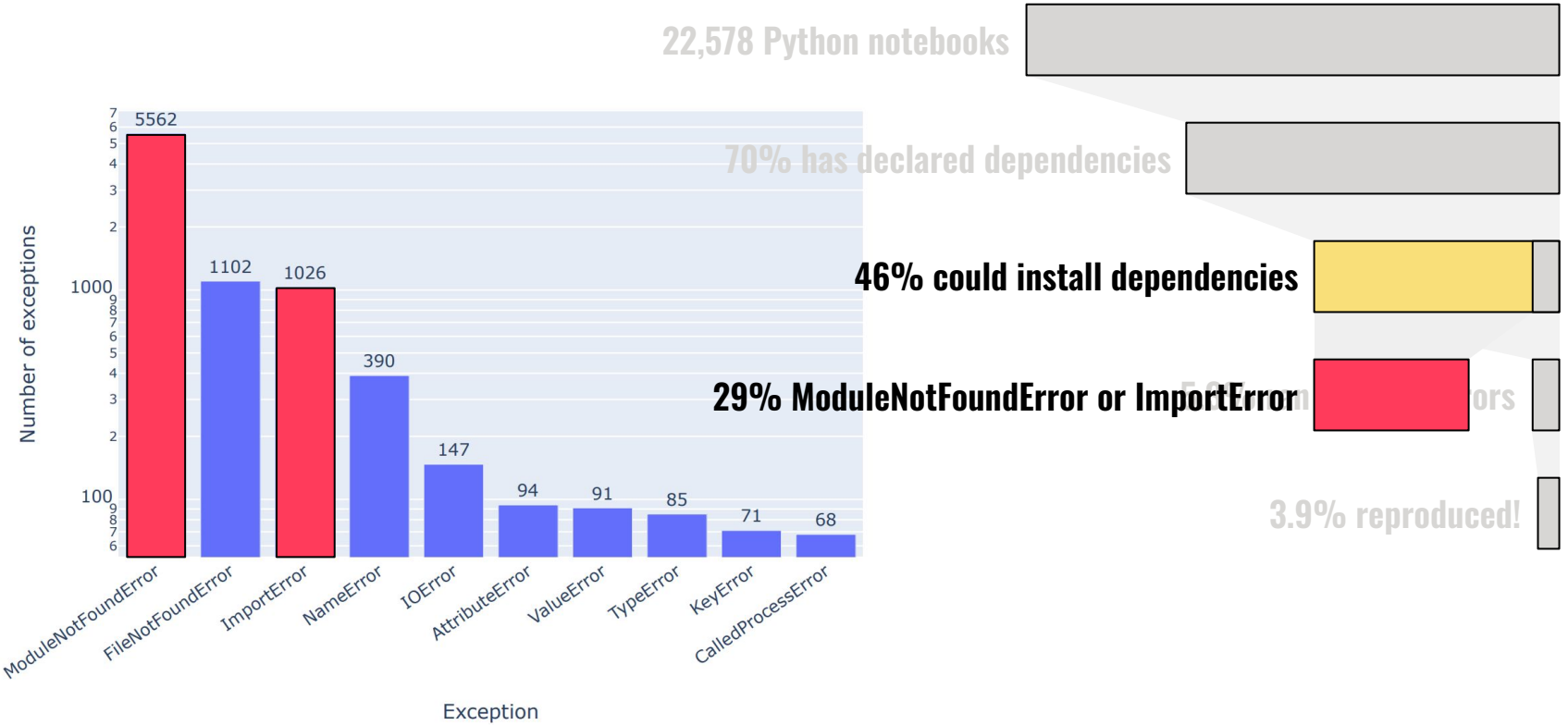
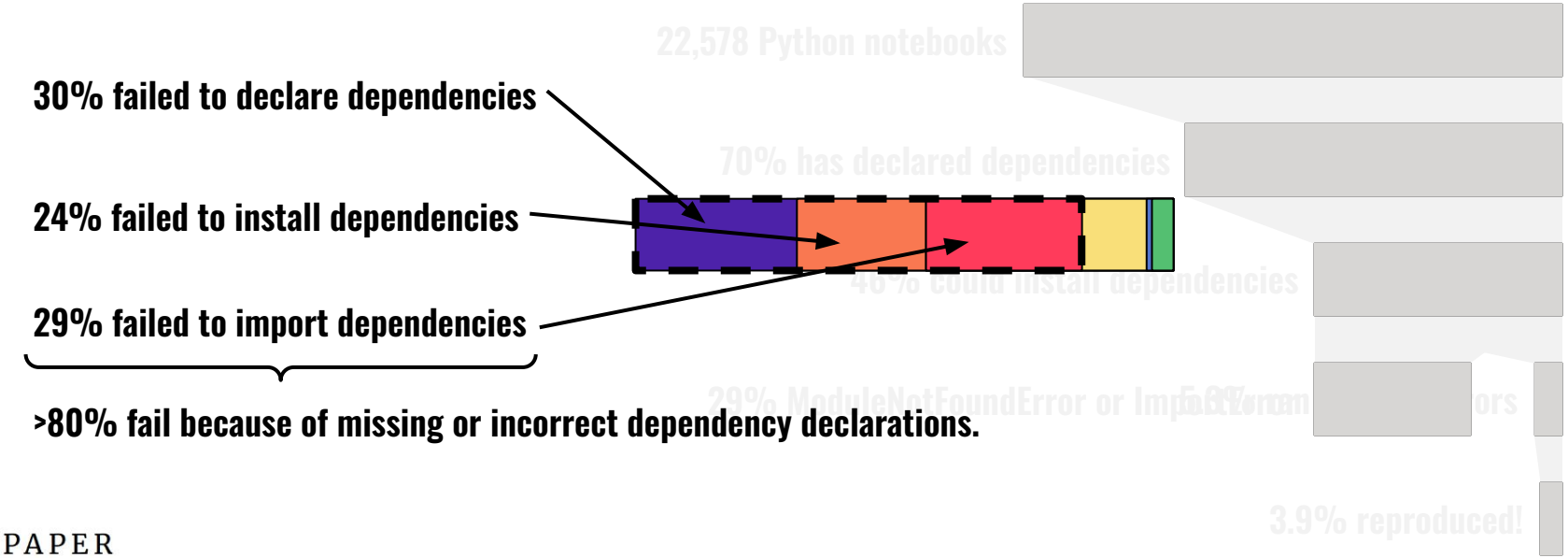


Figure 19. Exceptions occurring in Jupyter notebooks in our corpus. See Table 5 for information about the nature of these errors and potential fixes.



PAPER

Computational reproducibility of Jupyter notebooks from biomedical publications

Sheeba Samuel[1,2*†] and Daniel Mietchen[3,4,5†‡]

https://arxiv.org/abs/2310.12598

Less is More? An Empirical Study on Configuration Issues in Python PyPI Ecosystem

Yun Peng
The Chinese University of Hong Kong
Hong Kong, China
ypeng@cse.cuhk.edu.hk

Ruida Hu
Harbin Institute of Technology
Shenzhen, China
200111107@stu.hit.edu.cn

Ruoke Wang
Harbin Institute of Technology
Shenzhen, China
200110930@stu.hit.edu.cn

Cuiyun Gao*
Harbin Institute of Technology
Shenzhen, China
gaocuiyun@hit.edu.cn

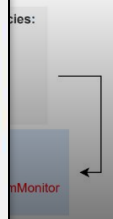
Shuqing Li
The Chinese University of Hong Kong
Hong Kong, China
sqli21@cse.cuhk.edu.hk

Michael R. Lyu
The Chinese University of Hong Kong
Hong Kong, China
lyu@cse.cuhk.edu.hk

ABSTRA

Python is the
source comm
diverse third
less, the utili
conflicts in
pendency co
to automatic
version-level checks and inference, based on the assumption that configurations of libraries in the PyPI ecosystem are correct. However, our study reveals that this assumption is not universally valid, and relying solely on version-level checks proves inadequate in ensuring compatible run-time environments.

Only 5,371 (65%) libraries, comprising 131,720 (39%) releases, successfully pass all three checks of PyCon. This indicates that approximately 30% of libraries and 51% of releases on the PyPI platform can be installed but may encounter source-level compatibility problems.



y library PFRL.

in fostering its prosperity. The accessibility and utility of Python are further amplified by the public libraries available on the Python Package Index (PyPI) platform. With over 470 thousand Python

Table 2: Configuration issues detected by PyCon. There may be multiple issues occurring in one release.

Category	Issue	Check	#Releases	Fatal?	Possible Reasons
Incomplete Configuration	Missing configuration files	Installation Check	251	✔	Missing required information
	Missing required libraries for setup	Installation Check	3,318	✔	
	Missing Python versions	Dependency Check	55,138	✘	
	Missing required libraries for direct imports	Import Validation	142,521	✘	
Incorrect Configuration	Dependency conflicts in setup	Installation Check	6,318	✔	Unsolvable constraints
	Incorrect Python versions	Installation Check	4,155	✔	Incorrect dependencies
	Other run-time Errors in setup	Installation Check	3,464	✔	Missing files
	Inconsistent configurations with metadata	Dependency Check	592	✘	Naming error
	Inconsistent version numbers with release dates	Dependency Check	12,018	✘	Confusing version orders
	Missing required modules for indirect imports	Import Validation	11,023	✘	Incorrect dependencies
	Inconsistent modules in direct imports with installed depenencies	Import Validation	6,678	✘	
	Other run-time Errors in imports	Import Validation	8,178	✘	
Incorrect Code	Missing source code	Dependency Check	2,588	✔	Creating placeholders
	Parsing error	Dependency Check	431	✔	Invalid syntax/encoding
	Multiple version control failure	Import Validation	15,507	✘	Incorrect dependencies

Dependencies and dependency declarations

code.py:

```
import numpy as np

...
```

pyproject.toml

```
[project]
name = "my_project"
dependencies = ["numpy"]
```

setup.cfg:

```
[options]
install_requires =
    numpy==1.26.2
```

setup.py:

```
from setuptools import setup

setup(
    name="my_project",
    install_requires=["numpy"]
)
```

```
$ python code.py
Traceback (most recent call last):
  File ".../code.py", line 1, in <module>
    import numpy as np
ModuleNotFoundError: No module named 'numpy'
```

Dependencies and dependency declarations

code.py:

```
import numpy as np

...
```

requirements.txt:

```
numpy
```

Undeclared and unused dependencies

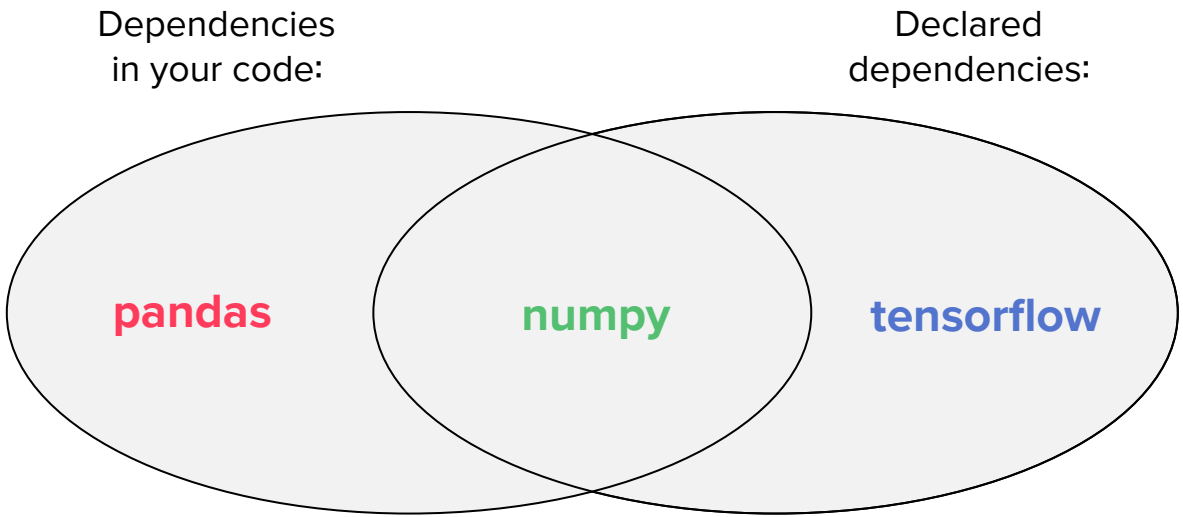
code.py:

```
import numpy as np
import pandas

...
```

requirements.txt:

```
numpy
tensorflow
```

Undeclared and unused dependencies



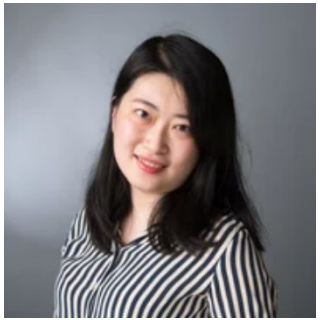
Can we automate this analysis?

TWEAG
by Modus Create

Nour El Mawass

Maria Knorps

Johan Herland



Zhihan Zhang

Richard Bullington-McGuire

FawltyDeps

code.py:

```
import numpy as np
import pandas

...
```

requirements.txt:

```
numpy
tensorflow
```

```
$ fawltydeps --detailed

These imports appear to be undeclared dependencies:
- 'pandas' imported at:
    code.py:2

These dependencies appear to be unused (i.e. not imported):
- 'tensorflow' declared in:
    requirements.txt
```

FawltyDeps

Python code and compatibility:

- Python scripts (`.py`)
- Jupyter notebooks (`.ipynb`)
- Python v3.8 – v3.13
- Linux, MacOS, and Windows

Dependency declarations:

- `requirements.txt`
- `pyproject.toml` (PEP 621, Poetry, Pixi)
- `setup.py` (simple)
- `setup.cfg`
- `environment.yml` (Conda)
- `pixi.toml` (Pixi)

Availability:

- via PyPI:
  - `pip install fawltydeps`
  - As a dev dependency: `uv add fawltydeps`
  - As a stand-alone tool: `uvx fawltydeps`
- as a pre-commit hook or Github Action

Configurable:

- `fawltydeps --help`
- Command-line options override configuration
- `$fawltydeps_<directive>` in environment
- `[tool.fawltydeps]` in `pyproject.toml`
- `fawltydeps --generate-toml-config`

FawltyDeps output

code.py:

```
import numpy as np
import pandas

...
```

requirements.txt:

```
numpy
tensorflow
```

```
$ fawltydeps --list-imports --detailed
code.py:1: numpy
code.py:2: pandas
```

```
$ fawltydeps --list-deps --detailed
requirements.txt: numpy
requirements.txt: tensorflow
```

```
$ fawltydeps --list-sources --detailed

Sources of Python code:
  code.py (using ./ as base for 1st-party imports)

Sources of declared dependencies:
  requirements.txt (parsed as a requirements.txt file)
```

FawltyDeps

code.py:

```
import numpy as n
import pandas

...
```

```
$ fawltydeps --li
code.py:1: numpy
code.py:2: pandas
```

```
$ fawltydeps --json
{
  "settings": { ... },
  "sources": [
    {
      "source_type": "CodeSource",
      "path": "code.py",
      "base_dir": "."
    },
    {
      "source_type": "DepsSource",
      "path": "requirements.txt",
      "parser_choice": "requirements.txt"
    }
  ],
  "imports": [
    {
      "name": "numpy",
      "source": {"path": "code.py", "linen
    },
    {
      "name": "pandas",
      "source": {"path": "code.py", "linen
    }
  ],
  "declared_deps": [
    {
      "name": "numpy",
      "source": {"path": "requirements.txt
    },
    {
      "name": "tensorflow",
      "source": {"path": "requirements.txt"}
    }
  ],
  "resolved_deps": {
    "tensorflow": {
      "package_name": "tensorflow",
      "import_names": ["tensorflow"],
      "resolved_with": "IdentityMapping",
      "debug_info": null
    },
    "numpy": {
      "package_name": "numpy",
      "import_names": ["numpy"],
      "resolved_with": "IdentityMapping",
      "debug_info": null
    }
  },
  "undeclared_deps": [
    {
      "name": "pandas",
      "references": [
        {"path": "code.py", "lineno": 2}
      ]
    }
  ],
  "unused_deps": [
    {
      "name": "tensorflow",
      "references": [
        {"path": "requirements.txt"}
      ]
    }
  ],
  "version": "0.18.0"
}
```

FawltyDeps as pre-commit hook

https://pre-commit.com


```
$ pip install pre-commit
[...]
$ vim .pre-commit-config.yaml
$ pre-commit install
pre-commit installed at .git/hooks/pre-commit
```

.pre-commit-config.yaml:

```
repos:
  - repo: https://github.com/tweag/FawltyDeps
    rev: v0.18.0
    hooks:
      - id: check-undeclared
      - id: check-unused
```

FawltyDeps as pre-commit hook

https://pre-commit.com


```
$ vim code.py
$ git commit -am “Add pandas”
[...]
FawltyDeps-undeclared.................Failed
- hook id: check-undeclared
- exit code: 3

These imports appear to be undeclared
dependencies:
- 'pandas' imported at:
    code.py:2

FawltyDeps-unused..(no files to check)Skipped
```

code.py:

```
 import numpy as np
+import pandas

 ...
```

FawltyDeps as pre-commit hook

https://pre-commit.com


```
$ vim code.py requirements.txt
$ git commit -am "Add pandas"
FawltyDeps-undeclared.................Passed
FawltyDeps-unused.....................Passed
[main 6ece52e] Add pandas
 2 files changed, 2 insertions(+)
$ vim requirements.txt
$ git commit -am "Declare tensorflow dependency"
FawltyDeps-undeclared..(no files to check)Skipped
FawltyDeps-unused.........................Failed
- hook id: check-unused
- exit code: 4

These dependencies appear to be unused (...):
- 'tensorflow' declared in:
    requirements.txt
```

code.py:

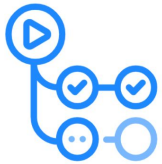
```
import numpy as np
import pandas

...
```

requirements.txt:

```
 numpy
 pandas
+tensorflow
```

FawltyDeps in CI (GitHub Actions)

github.com/tweag/FawltyDeps-action

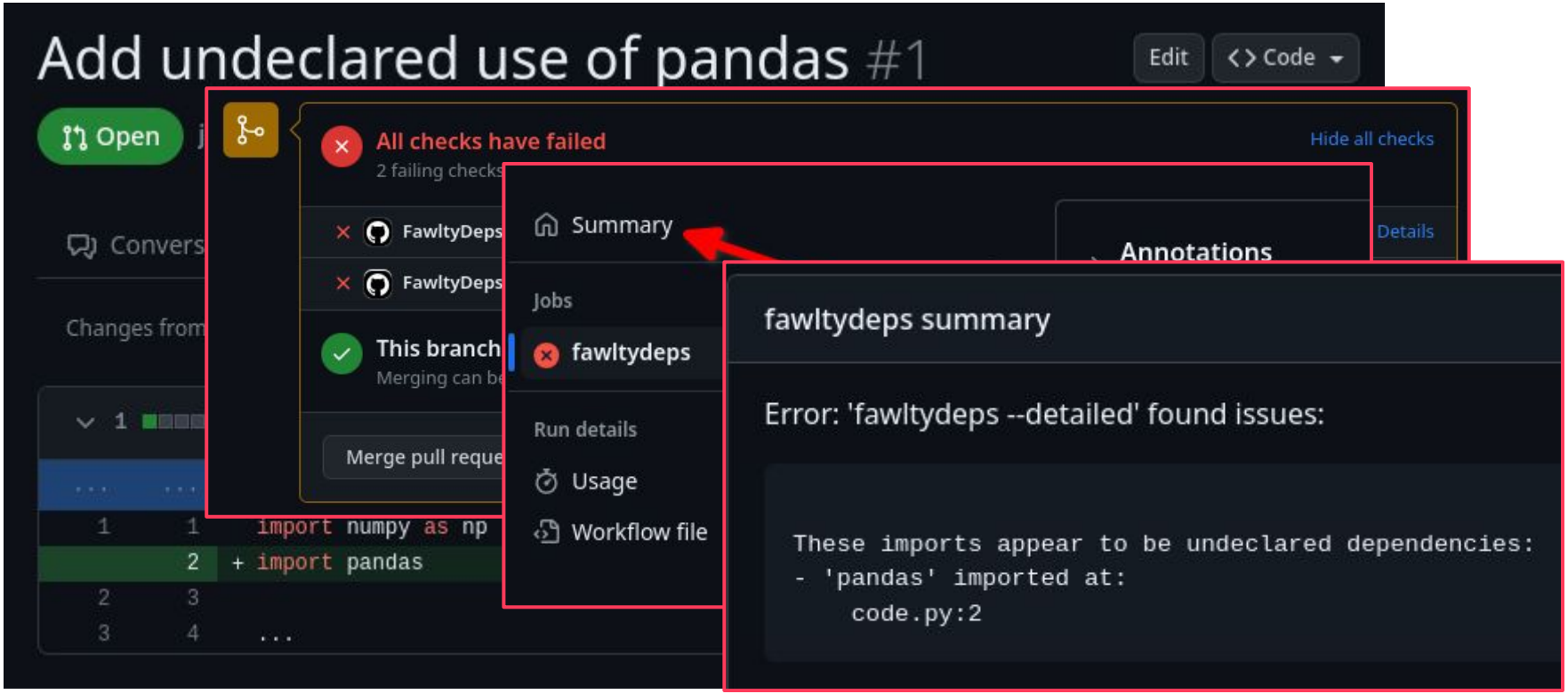


GitHub Actions

.github/workflows/fawltydeps.yaml:

```yaml
name: FawltyDeps

on: [push, pull_request]

jobs:
  fawltydeps:
    runs-on: ubuntu-latest
    steps:
      - name: checkout
        uses: actions/checkout@v4
      - name: lint - FawltyDeps
        uses: tweag/FawltyDeps-action@v0.2.0
        with:
          options: --detailed
```

FawltyDeps in CI (GitHub Action)



How to draw an Owl.

"A fun and creative guide for beginners"

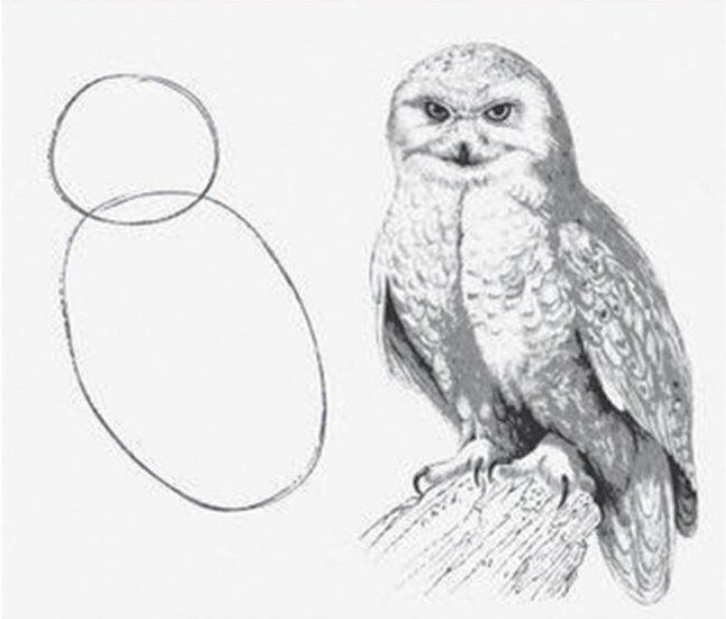


Fig 1. Draw two circles

Fig 2. Draw rest of the damn Owl

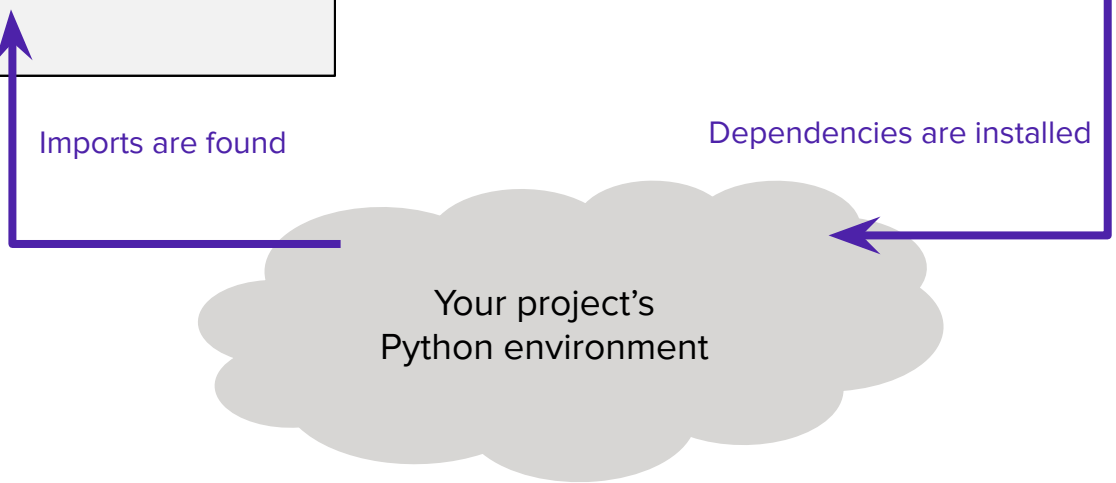
Matching imports names to package names

code.py:

```
import numpy as np
import sklearn
import setuptools
import pkg_resources

...
```

requirements.txt:

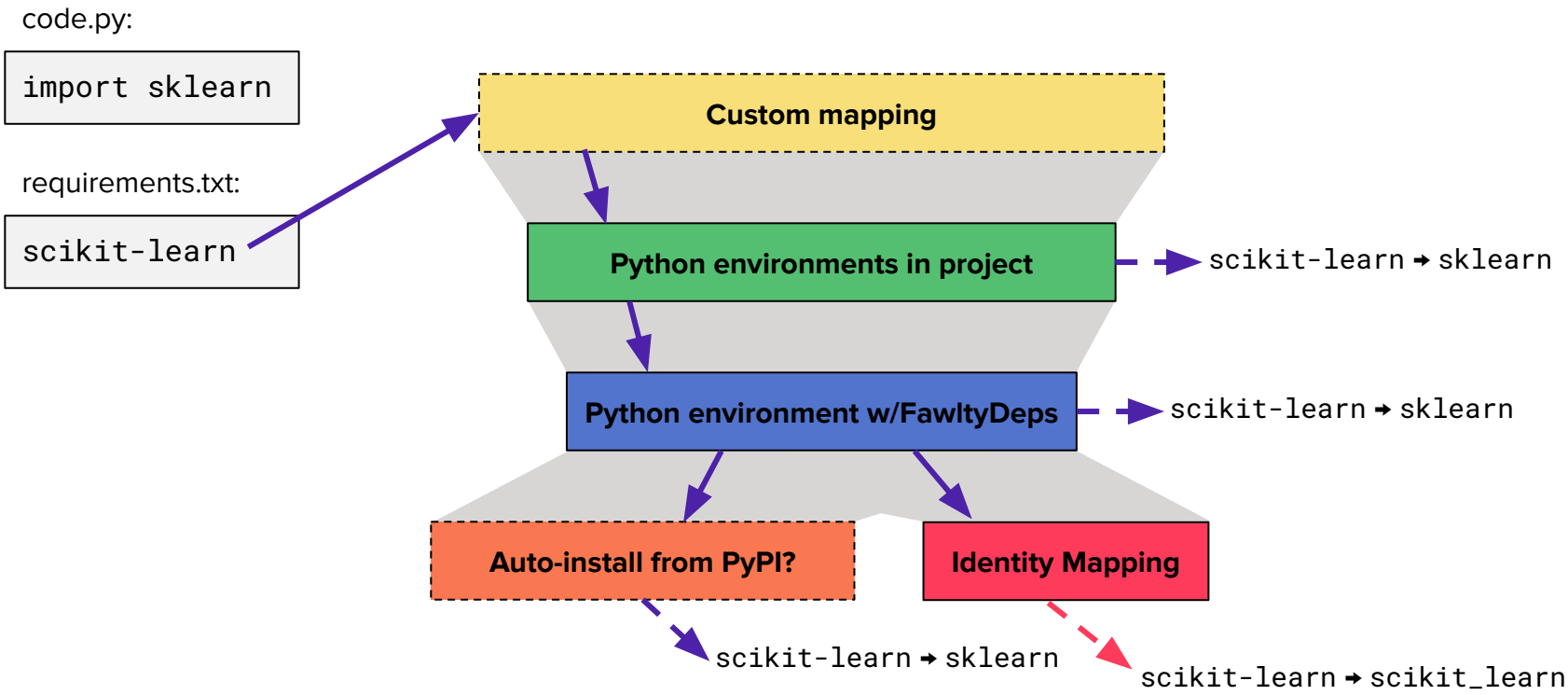
```
numpy
scikit-learn
setuptools
```



Resolving dependencies into import names

- Python environments inside the project
  - Search the entire project?
  - Search within the paths given to FawltyDeps
  - Python environments given with `--pyenv`
- Python environment where FawltyDeps is installed
- Custom mapping: `[tool.fawltydeps.custom_mapping]`
  - Takes precedence, if given
- Auto-install from PyPI? `--install-deps`
  - Expensive, disabled by default
- Final resort: Identity mapping

Resolving dependencies into import names



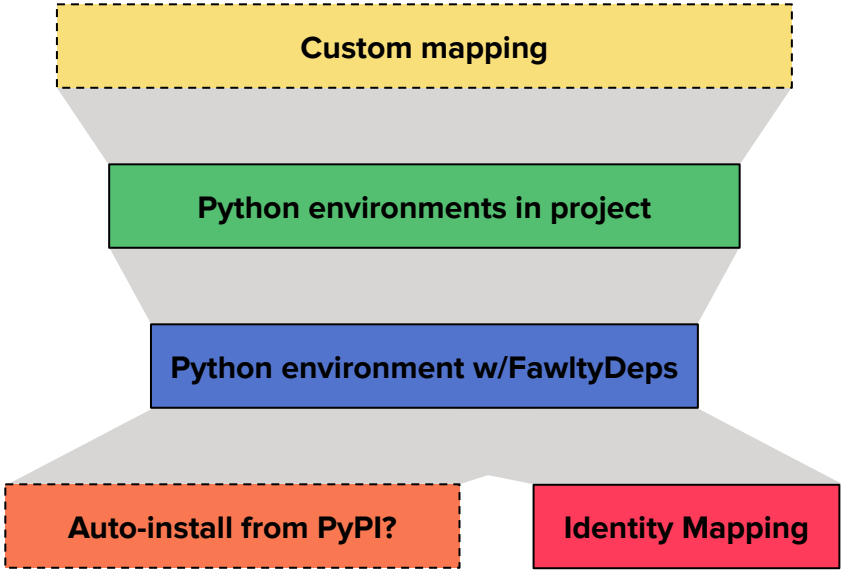
Resolving dependencies into import names

code.py:

```
import sklearn
```

requirements.txt:

```
scikit-learn
```



More complexities and advanced use cases

- Exclude parts of your project from analysis:
  - `--exclude tests/`
  - `--exclude-from .gitignore`

- Some dependencies are never meant to be imported:
  - Tools: tox, black, flake8, ruff, pylint, mypy, etc.
  - Type stubs
  - Indirect imports or plugin-style package collections
  - `--ignore-unused`

- Conditional imports, alternative imports, e.g.: `try: import foo; except: import bar`
  - `--ignore-undeclared`

Work in progress

- Dynamic imports, e.g.: `importlib.import_module(<some dynamic expression>)`

- Required dependencies vs. various degrees of optional dependencies
  - `install-requires` vs. `extras-require` in `setup.py`
  - `[project.dependencies]` vs `[project.optional-dependencies]` in `pyproject.toml`
  - PEP 735: `[dependency-groups]` in `pyproject.toml`
  - Poetry: `[tool.poetry.group.$NAME.dependencies]` in `pyproject.toml`
  - Development dependencies: `dev-requirements.txt` vs `requirements.txt`

Summary

- Reproducibility matters
- Missing or incorrect dependency declarations is the greatest culprit

- **FawltyDeps** finds **undeclared** and **unused** dependencies in your project
  - Works with Python scripts and Jupyter notebooks, across Python version and platforms
  - Understands the many ways of declaring dependencies
  - Works out of the box on most projects, and can be configured to suit *all* projects
  - Can be an automated part of your workflow, either pre-commit or in your CI

QUESTIONS?

Slides

- FawltyDeps on GitHub
- FawltyDeps on PyPI
- FawltyDeps on Discord

- Tweag blog: tweag.io/blog
- Tweag: tweag.io
- Modus Create: moduscreate.com

@jherland

THANK YOU!

MODUS CREATE | TWEAG by Modus Create