From Ideas to APIs Delivering Fast with Modern Python

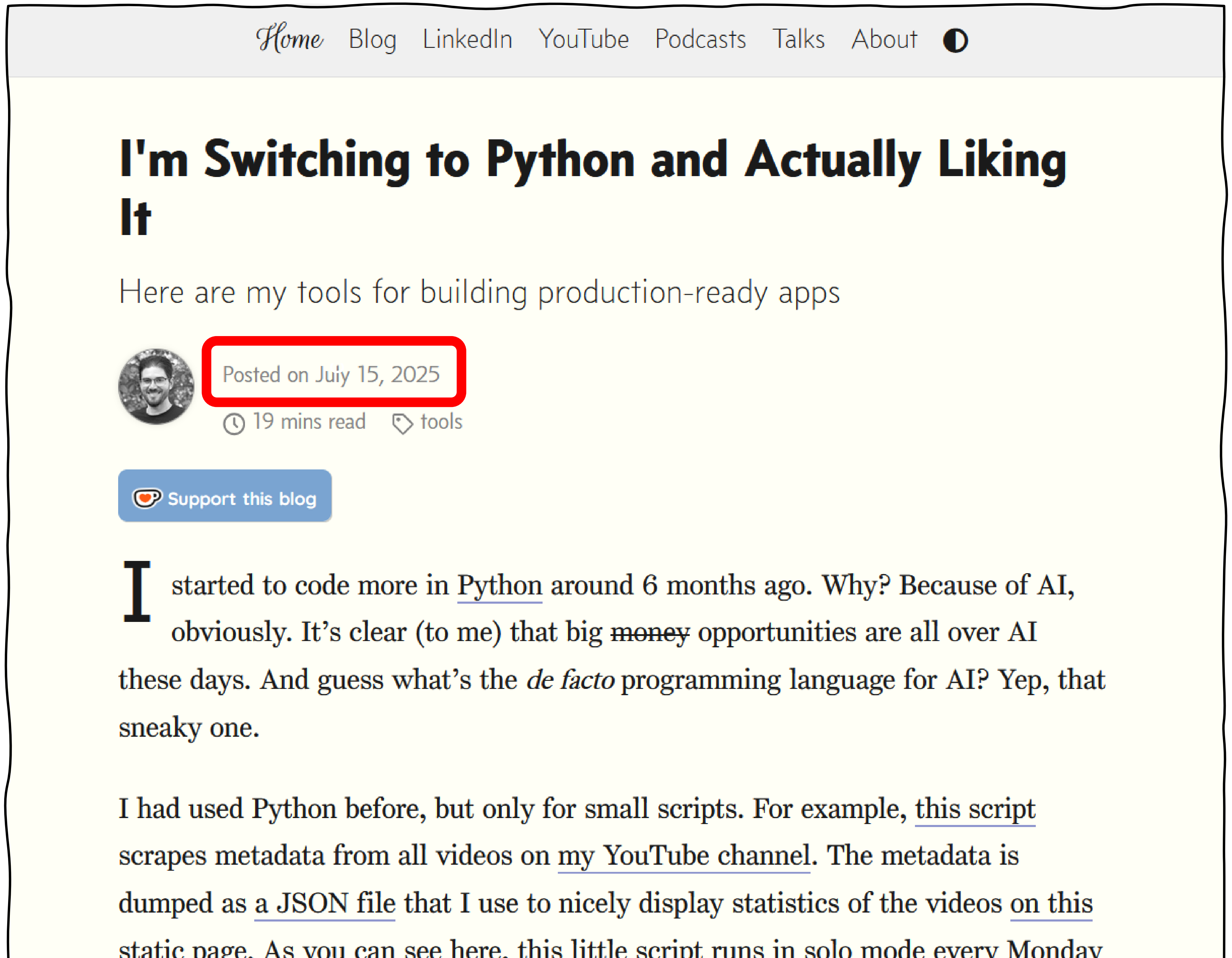
How to shorten your path from notebook idea to running API, focusing on the plumbing that actually ships apps and models to production environments.

César Soto Valero

Data Scientist / ML Engineer

PyData

https://www.cesarsotovalero.net/blog/i-am-switching-to-python-and-actually-liking-it.html



I'm Switching to Python and Actually Liking It

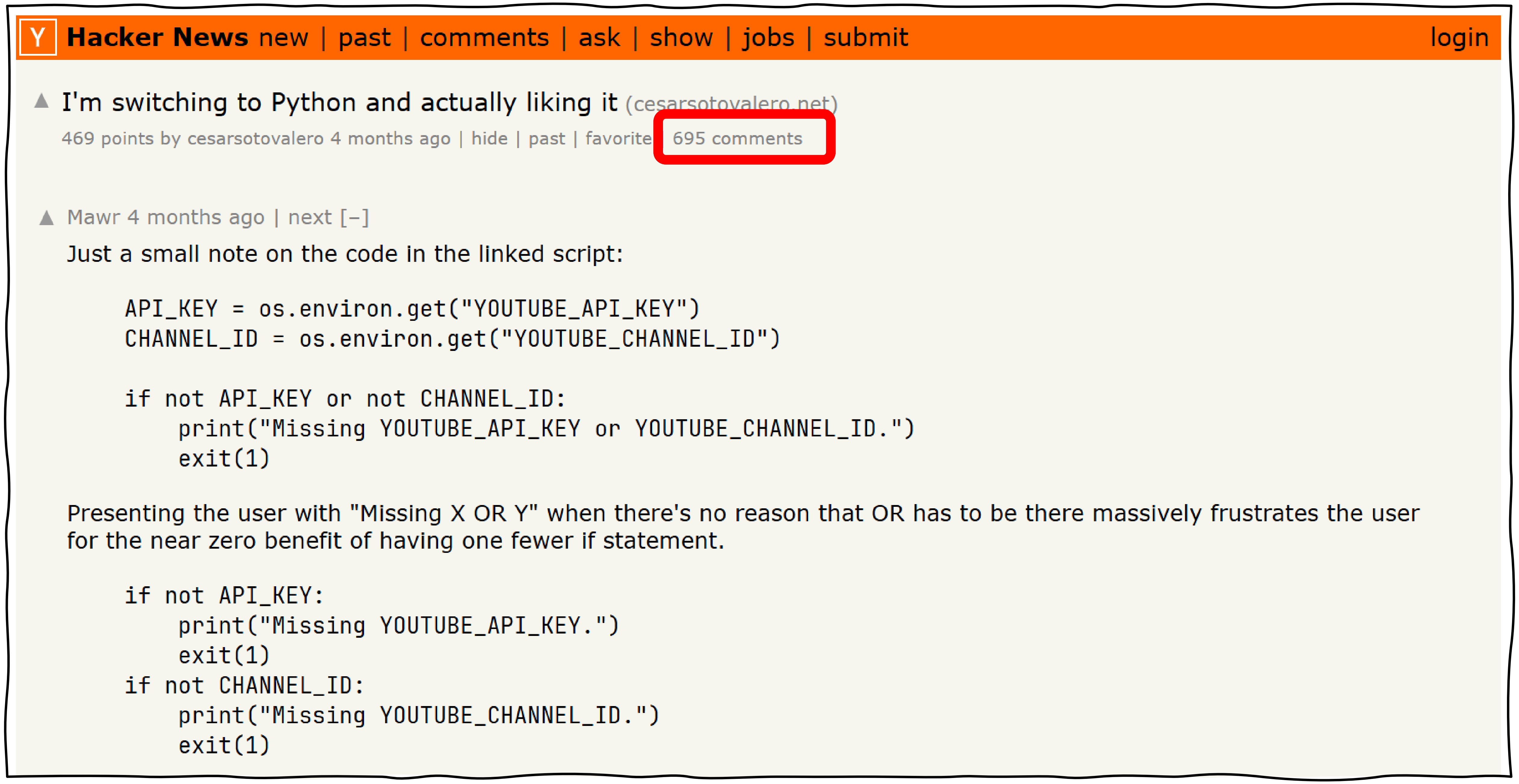
Here are my tools for building production-ready apps

Posted on July 15, 2025

19 mins read tools

I started to code more in Python around 6 months ago. Why? Because of AI, obviously. It's clear (to me) that big ~~money~~ opportunities are all over AI these days. And guess what's the *de facto* programming language for AI? Yep, that sneaky one.

I had used Python before, but only for small scripts. For example, this script scrapes metadata from all videos on my YouTube channel. The metadata is dumped as a JSON file that I use to nicely display statistics of the videos on this static page. As you can see here, this little script runs in solo mode every Monday https://news.ycombinator.com/item?id=44579717



Y **Hacker News** new | past | comments | ask | show | jobs | submit login

▲ I'm switching to Python and actually liking it (cesarsotovalero.net)

469 points by cesarsotovalero 4 months ago | hide | past | favorite | 695 comments

▲ Mawr 4 months ago | next [–]

Just a small note on the code in the linked script:

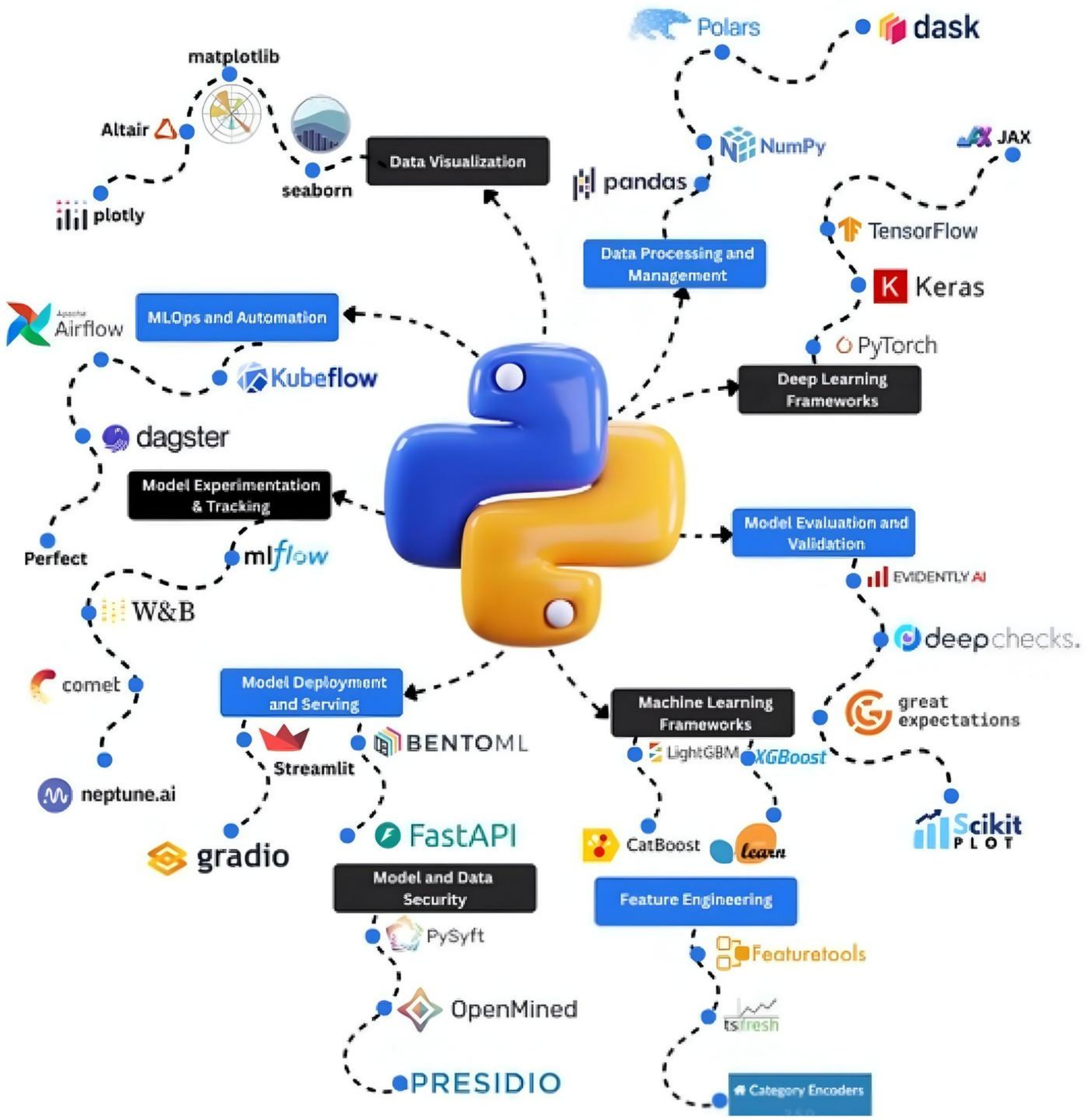
```
API_KEY = os.environ.get("YOUTUBE_API_KEY")
CHANNEL_ID = os.environ.get("YOUTUBE_CHANNEL_ID")

if not API_KEY or not CHANNEL_ID:
    print("Missing YOUTUBE_API_KEY or YOUTUBE_CHANNEL_ID.")
    exit(1)
```

Presenting the user with "Missing X OR Y" when there's no reason that OR has to be there massively frustrates the user for the near zero benefit of having one fewer if statement.

```
if not API_KEY:
    print("Missing YOUTUBE_API_KEY.")
    exit(1)
if not CHANNEL_ID:
    print("Missing YOUTUBE_CHANNEL_ID.")
    exit(1)
```

AI
ME
TRADITIONAL ML
imgflip.com

Python tools that you should know



TensorFlow

PyTorch

Transformers

W&B

Weight and Biases

JAX

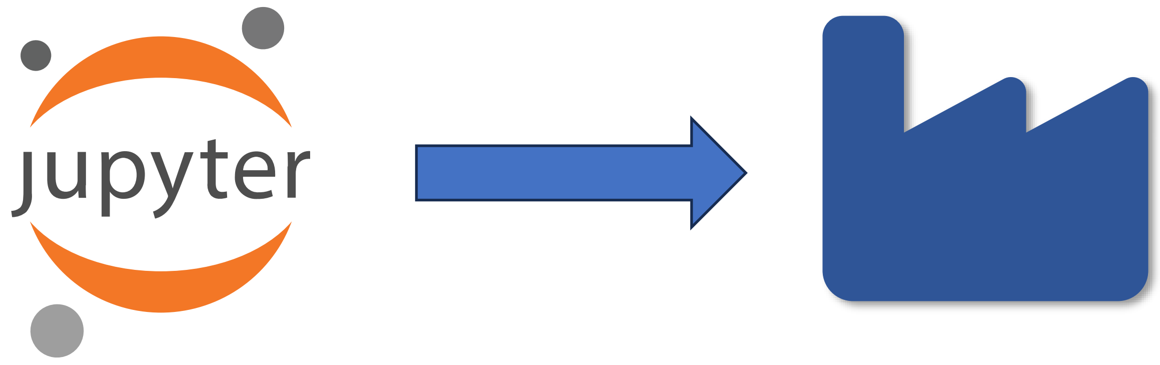
LangChain

ACME

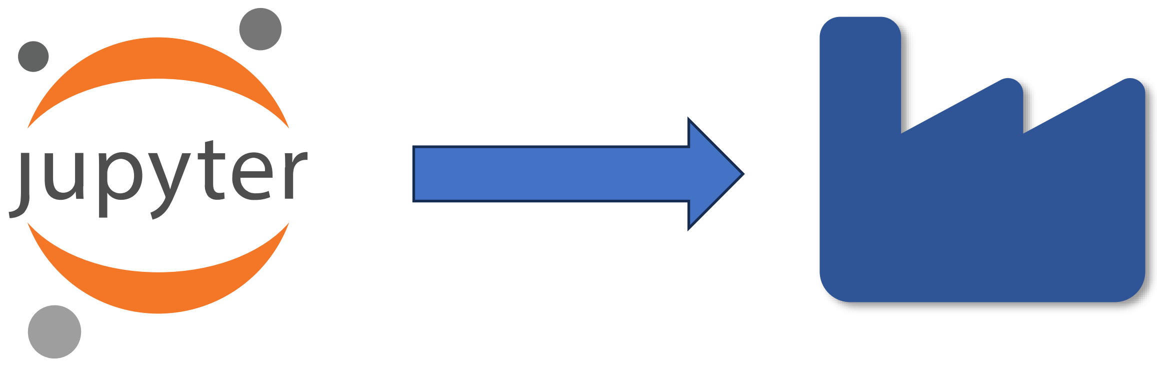
Llama Index

Diffusers

Acme



Most Notebooks Never Survive The Real World!



Most notebooks never survive the real world

The Screenshot Trap

You answer the business question, screenshot the chart for Slack, and the repository quietly rots. Analysis complete, value delivered once, code forgotten.

Endless Tweaks

Stakeholders ask for "just one more adjustment" forever. Without structure, every tweak means hunting through notebooks and manually re-running cells in the right order.

Missing Infrastructure

No tests to catch regressions, no reproducible environments, no API endpoints, no path to adoption. The insights exist only in your local machine.

Our real bottleneck isn't model accuracy or algorithm choice, it's the plumbing that transforms analysis into production systems.

But AI will write most of the code, right? 🤔

Well, not really. 🙂

You need structure 🧱

Monorepo: One Home For Everything

- Single place to search, build, deploy
- Atomic changes across backend and frontend
- Simplified dependency management
- Notebooks beside production code

```
project/
│
├── .github/ # GitHub Actions workflows for CI/CD pipelines
│   ├── workflows/ # Directory containing YAML files for automated workflows
│   └── dependabot.yml # Configuration for Dependabot to manage dependencies
│
├── .vscode/ # VSCode configuration for the project
│   ├── launch.json # Debugging configurations for VSCode
│   └── settings.json # Project-specific settings for VSCode
│
├── docs/ # Website and docs (a static SPA with MkDocs)
│
├── project-api/ # Backend API for handling business logic and heavy processing
│   ├── data/ # Directory for storing datasets or other static files
│   ├── notebooks/ # Jupyter notebooks for quick (and dirty) experimentation and prototyping
│   ├── tools/ # Utility scripts and tools for development or deployment
│   ├── src/ # Source code for the backend application
│   │   ├── app/ # Main application code
│   │   └── tests/ # Unit tests for the backend
│   │
│   ├── .dockerignore # Specifies files to exclude from Docker builds
│   ├── .python-version # Python version specification for pyenv
│   ├── Dockerfile # Docker configuration for containerizing the backend
│   ├── Makefile # Automation tasks for building, testing, and deploying
│   ├── pyproject.toml # Python project configuration file
│   ├── README.md # Documentation for the backend API
│   └── uv.lock # Lock file for dependencies managed by UV
│
├── project-ui/ # Frontend UI for the project (Next.js, React, etc.)
│
├── .gitignore # Global Git ignore file for the repository
├── .pre-commit-config.yaml # Configuration for pre-commit hooks
├── CONTRIBUTING.md # Guidelines for contributing to the project
├── docker-compose.yml # Docker Compose configuration for multi-container setups
├── LICENSE # License information for the project (I always choose MIT)
├── Makefile # Automation tasks for building, testing, and deploying
└── README.md # Main documentation for the project (main features, installation, and usage)
```

React

```
project/
│
├── .github/ # GitHub Actions workflows for CI/CD pipelines
│   ├── workflows/ # Directory containing YAML files for automated workflows
│   └── dependabot.yml # Configuration for Dependabot to manage dependencies
│
├── .vscode/ # VSCode configuration for the project
│   ├── launch.json # Debugging configurations for VSCode
│   └── settings.json # Project-specific settings for VSCode
│
├── docs/ # Website and docs (a static SPA with MkDocs)
│
├── project-api/ # Backend API for handling business logic and heavy processing
│   ├── data/ # Directory for storing datasets or other static files
│   ├── notebooks/ # Jupyter notebooks for quick (and dirty) experimentation and prototyping
│   ├── tools/ # Utility scripts and tools for development or deployment
│   ├── src/ # Source code for the backend application
│   │   ├── app/ # Main application code
│   │   └── tests/ # Unit tests for the backend
│   │
│   ├── .dockerignore # Specifies files to exclude from Docker builds
│   ├── .python-version # Python version specification for pyenv
│   ├── Dockerfile # Docker configuration for containerizing the backend
│   ├── Makefile # Automation tasks for building, testing, and deploying
│   ├── pyproject.toml # Python project configuration file
│   ├── README.md # Documentation for the backend API
│   └── uv.lock # Lock file for dependencies managed by UV
│
├── project-ui/ # Frontend UI for the project (Next.js, React, etc.)
│
├── .gitignore # Global Git ignore file for the repository
├── .pre-commit-config.yaml # Configuration for pre-commit hooks
├── CONTRIBUTING.md # Guidelines for contributing to the project
├── docker-compose.yml # Docker Compose configuration for multi-container setups
├── LICENSE # License information for the project (I always choose MIT)
├── Makefile # Automation tasks for building, testing, and deploying
└── README.md # Main documentation for the project (main features, installation, and usage)
```

$\}$ UI components

FastAPI

```
project/
│
├── .github/ # GitHub Actions workflows for CI/CD pipelines
│   ├── workflows/ # Directory containing YAML files for automated workflows
│   └── dependabot.yml # Configuration for Dependabot to manage dependencies
│
├── .vscode/ # VSCode configuration for the project
│   ├── launch.json # Debugging configurations for VSCode
│   └── settings.json # Project-specific settings for VSCode
│
├── docs/ # Website and docs (a static SPA with MkDocs)
│
├── project-api/ # Backend API for handling business logic and heavy processing
│   ├── data/ # Directory for storing datasets or other static files
│   ├── notebooks/ # Jupyter notebooks for quick (and dirty) experimentation and prototyping
│   ├── tools/ # Utility scripts and tools for development or deployment
│   ├── src/ # Source code for the backend application
│   │   ├── app/ # Main application code
│   │   └── tests/ # Unit tests for the backend
│   │
│   ├── .dockerignore # Specifies files to exclude from Docker builds
│   ├── .python-version # Python version specification for pyenv
│   ├── Dockerfile # Docker configuration for containerizing the backend
│   ├── Makefile # Automation tasks for building, testing, and deploying
│   ├── pyproject.toml # Python project configuration file
│   ├── README.md # Documentation for the backend API
│   └── uv.lock # Lock file for dependencies managed by UV
│
├── project-ui/ # Frontend UI for the project (Next.js, React, etc.)
│
├── .gitignore # Global Git ignore file for the repository
├── .pre-commit-config.yaml # Configuration for pre-commit hooks
├── CONTRIBUTING.md # Guidelines for contributing to the project
├── docker-compose.yml # Docker Compose configuration for multi-container setups
├── LICENSE # License information for the project (I always choose MIT)
├── Makefile # Automation tasks for building, testing, and deploying
└── README.md # Main documentation for the project (main features, installation, and usage)
```

$\}$ Backend services

```
project/
│
├── .github/ # GitHub Actions workflows for CI/CD pipelines
│   ├── workflows/ # Directory containing YAML files for automated workflows
│   └── dependabot.yml # Configuration for Dependabot to manage dependencies
│
├── .vscode/ # VSCode configuration for the project
│   ├── launch.json # Debugging configurations for VSCode
│   └── settings.json # Project-specific settings for VSCode
│
├── docs/ # Website and docs (a static SPA with MkDocs)
│
├── project-api/ # Backend API for handling business logic and heavy processing
│   ├── data/ # Directory for storing datasets or other static files
│   ├── notebooks/ # Jupyter notebooks for quick (and dirty) experimentation and prototyping
│   ├── tools/ # Utility scripts and tools for development or deployment
│   ├── src/ # Source code for the backend application
│   │   ├── app/ # Main application code
│   │   └── tests/ # Unit tests for the backend
│   │
│   ├── .dockerignore # Specifies files to exclude from Docker builds
│   ├── .python-version # Python version specification for pyenv
│   ├── Dockerfile # Docker configuration for containerizing the backend
│   ├── Makefile # Automation tasks for building, testing, and deploying
│   ├── pyproject.toml # Python project configuration file
│   ├── README.md # Documentation for the backend API
│   └── uv.lock # Lock file for dependencies managed by UV
│
├── project-ui/ # Frontend UI for the project (Next.js, React, etc.)
│
├── .gitignore # Global Git ignore file for the repository
├── .pre-commit-config.yaml # Configuration for pre-commit hooks
├── CONTRIBUTING.md # Guidelines for contributing to the project
├── docker-compose.yml # Docker Compose configuration for multi-container setups
├── LICENSE # License information for the project (I always choose MIT)
├── Makefile # Automation tasks for building, testing, and deploying
└── README.md # Main documentation for the project (main features, installation, and usage)
```

$\}$ sources, tests, notebooks
scripts, files, etc.

```
project/
│
├── .github/ # GitHub Actions workflows for CI/CD pipelines
│   ├── workflows/ # Directory containing YAML files for automated workflows
│   └── dependabot.yml # Configuration for Dependabot to manage dependencies
│
├── .vscode/ # VSCode configuration for the project
│   ├── launch.json # Debugging configurations for VSCode
│   └── settings.json # Project-specific settings for VSCode
│
├── docs/ # Website and docs (a static SPA with MkDocs)
│
├── project-api/ # Backend API for handling business logic and heavy processing
│   ├── data/ # Directory for storing datasets or other static files
│   ├── notebooks/ # Jupyter notebooks for quick (and dirty) experimentation and prototyping
│   ├── tools/ # Utility scripts and tools for development or deployment
│   ├── src/ # Source code for the backend application
│   │   ├── app/ # Main application code
│   │   └── tests/ # Unit tests for the backend
│   │
│   ├── .dockerignore # Specifies files to exclude from Docker builds
│   ├── .python-version # Python version specification for pyenv
│   ├── Dockerfile # Docker configuration for containerizing the backend
│   ├── Makefile # Automation tasks for building, testing, and deploying
│   ├── pyproject.toml # Python project configuration file
│   ├── README.md # Documentation for the backend API
│   └── uv.lock # Lock file for dependencies managed by UV
│
├── project-ui/ # Frontend UI for the project (Next.js, React, etc.)
│
├── .gitignore # Global Git ignore file for the repository
├── .pre-commit-config.yaml # Configuration for pre-commit hooks
├── CONTRIBUTING.md # Guidelines for contributing to the project
├── docker-compose.yml # Docker Compose configuration for multi-container setups
├── LICENSE # License information for the project (I always choose MIT)
├── Makefile # Automation tasks for building, testing, and deploying
└── README.md # Main documentation for the project (main features, installation, and usage)
```

Dependency management, containerization, and documentation

```
project/
│
├── .github/ # GitHub Actions workflows for CI/CD pipelines
│   ├── workflows/ # Directory containing YAML files for automated workflows
│   └── dependabot.yml # Configuration for Dependabot to manage dependencies
│
├── .vscode/ # VSCode configuration for the project
│   ├── launch.json # Debugging configurations for VSCode
│   └── settings.json # Project-specific settings for VSCode
│
├── docs/ # Website and docs (a static SPA with MkDocs)
│
├── project-api/ # Backend API for handling business logic and heavy processing
│   ├── data/ # Directory for storing datasets or other static files
│   ├── notebooks/ # Jupyter notebooks for quick (and dirty) experimentation and prototyping
│   ├── tools/ # Utility scripts and tools for development or deployment
│   ├── src/ # Source code for the backend application
│   │   ├── app/ # Main application code
│   │   └── tests/ # Unit tests for the backend
│   │
│   ├── .dockerignore # Specifies files to exclude from Docker builds
│   ├── .python-version # Python version specification for pyenv
│   ├── Dockerfile # Docker configuration for containerizing the backend
│   ├── Makefile # Automation tasks for building, testing, and deploying
│   ├── pyproject.toml # Python project configuration file
│   ├── README.md # Documentation for the backend API
│   └── uv.lock # Lock file for dependencies managed by UV
│
├── project-ui/ # Frontend UI for the project (Next.js, React, etc.)
│
├── .gitignore # Global Git ignore file for the repository
├── .pre-commit-config.yaml # Configuration for pre-commit hooks
├── CONTRIBUTING.md # Guidelines for contributing to the project
├── docker-compose.yml # Docker Compose configuration for multi-container setups
├── LICENSE # License information for the project (I always choose MIT)
├── Makefile # Automation tasks for building, testing, and deploying
└── README.md # Main documentation for the project (main features, installation, and usage)
```

$\}$ CI/CD pipelines

GitHub Actions

```
project/
│
├── .github/ # GitHub Actions workflows for CI/CD pipelines
│   ├── workflows/ # Directory containing YAML files for automated workflows
│   └── dependabot.yml # Configuration for Dependabot to manage dependencies
│
├── .vscode/ # VSCode configuration for the project
│   ├── launch.json # Debugging configurations for VSCode
│   └── settings.json # Project-specific settings for VSCode
│
├── docs/ # Website and docs (a static SPA with MkDocs)
│
├── project-api/ # Backend API for handling business logic and heavy processing
│   ├── data/ # Directory for storing datasets or other static files
│   ├── notebooks/ # Jupyter notebooks for quick (and dirty) experimentation and prototyping
│   ├── tools/ # Utility scripts and tools for development or deployment
│   ├── src/ # Source code for the backend application
│   │   ├── app/ # Main application code
│   │   └── tests/ # Unit tests for the backend
│   │
│   ├── .dockerignore # Specifies files to exclude from Docker builds
│   ├── .python-version # Python version specification for pyenv
│   ├── Dockerfile # Docker configuration for containerizing the backend
│   ├── Makefile # Automation tasks for building, testing, and deploying
│   ├── pyproject.toml # Python project configuration file
│   ├── README.md # Documentation for the backend API
│   └── uv.lock # Lock file for dependencies managed by UV
│
├── project-ui/ # Frontend UI for the project (Next.js, React, etc.)
│
├── .gitignore # Global Git ignore file for the repository
├── .pre-commit-config.yaml # Configuration for pre-commit hooks
├── CONTRIBUTING.md # Guidelines for contributing to the project
├── docker-compose.yml # Docker Compose configuration for multi-container setups
├── LICENSE # License information for the project (I always choose MIT)
├── Makefile # Automation tasks for building, testing, and deploying
└── README.md # Main documentation for the project (main features, installation, and usage)
```

} VSCode configs & preferences

```
project/
│
├── .github/ # GitHub Actions workflows for CI/CD pipelines
│   ├── workflows/ # Directory containing YAML files for automated workflows
│   └── dependabot.yml # Configuration for Dependabot to manage dependencies
│
├── .vscode/ # VSCode configuration for the project
│   ├── launch.json # Debugging configurations for VSCode
│   └── settings.json # Project-specific settings for VSCode
│
├── docs/ # Website and docs (a static SPA with MkDocs)
│
├── project-api/ # Backend API for handling business logic and heavy processing
│   ├── data/ # Directory for storing datasets or other static files
│   ├── notebooks/ # Jupyter notebooks for quick (and dirty) experimentation and prototyping
│   ├── tools/ # Utility scripts and tools for development or deployment
│   ├── src/ # Source code for the backend application
│   │   ├── app/ # Main application code
│   │   └── tests/ # Unit tests for the backend
│   │
│   ├── .dockerignore # Specifies files to exclude from Docker builds
│   ├── .python-version # Python version specification for pyenv
│   ├── Dockerfile # Docker configuration for containerizing the backend
│   ├── Makefile # Automation tasks for building, testing, and deploying
│   ├── pyproject.toml # Python project configuration file
│   ├── README.md # Documentation for the backend API
│   └── uv.lock # Lock file for dependencies managed by UV
│
├── project-ui/ # Frontend UI for the project (Next.js, React, etc.)
│
├── .gitignore # Global Git ignore file for the repository
├── .pre-commit-config.yaml # Configuration for pre-commit hooks
├── CONTRIBUTING.md # Guidelines for contributing to the project
├── docker-compose.yml # Docker Compose configuration for multi-container setups
├── LICENSE # License information for the project (I always choose MIT)
├── Makefile # Automation tasks for building, testing, and deploying
└── README.md # Main documentation for the project (main features, installation, and usage)
```

Other files

```
project/
│
├── .github/ # GitHub Actions workflows for CI/CD pipelines
│   ├── workflows/ # Directory containing YAML files for automated workflows
│   └── dependabot.yml # Configuration for Dependabot to manage dependencies
│
├── .vscode/ # VSCode configuration for the project
│   ├── launch.json # Debugging configurations for VSCode
│   └── settings.json # Project-specific settings for VSCode
│
├── docs/ # Website and docs (a static SPA with MkDocs)
│
├── project-api/ # Backend API for handling business logic and heavy processing
│   ├── data/ # Directory for storing datasets or other static files
│   ├── notebooks/ # Jupyter notebooks for quick (and dirty) experimentation and prototyping
│   ├── tools/ # Utility scripts and tools for development or deployment
│   ├── src/ # Source code for the backend application
│   │   ├── app/ # Main application code
│   │   └── tests/ # Unit tests for the backend
│   │
│   ├── .dockerignore # Specifies files to exclude from Docker builds
│   ├── .python-version # Python version specification for pyenv
│   ├── Dockerfile # Docker configuration for containerizing the backend
│   ├── Makefile # Automation tasks for building, testing, and deploying
│   ├── pyproject.toml # Python project configuration file
│   ├── README.md # Documentation for the backend API
│   └── uv.lock # Lock file for dependencies managed by UV
│
├── project-ui/ # Frontend UI for the project (Next.js, React, etc.)
│
├── .gitignore # Global Git ignore file for the repository
├── .pre-commit-config.yaml # Configuration for pre-commit hooks
├── CONTRIBUTING.md # Guidelines for contributing to the project
├── docker-compose.yml # Docker Compose configuration for multi-container setups
├── LICENSE # License information for the project (I always choose MIT)
├── Makefile # Automation tasks for building, testing, and deploying
└── README.md # Main documentation for the project (main features, installation, and usage)
```

The basic toolbox 🧰

Deterministic environments with uv and `pyproject.toml`

1

`uv init`

One command generates gitignore, virtual environment, `pyproject.toml`, and `lockfile`. Everything you need to start clean.

2

`uv add`

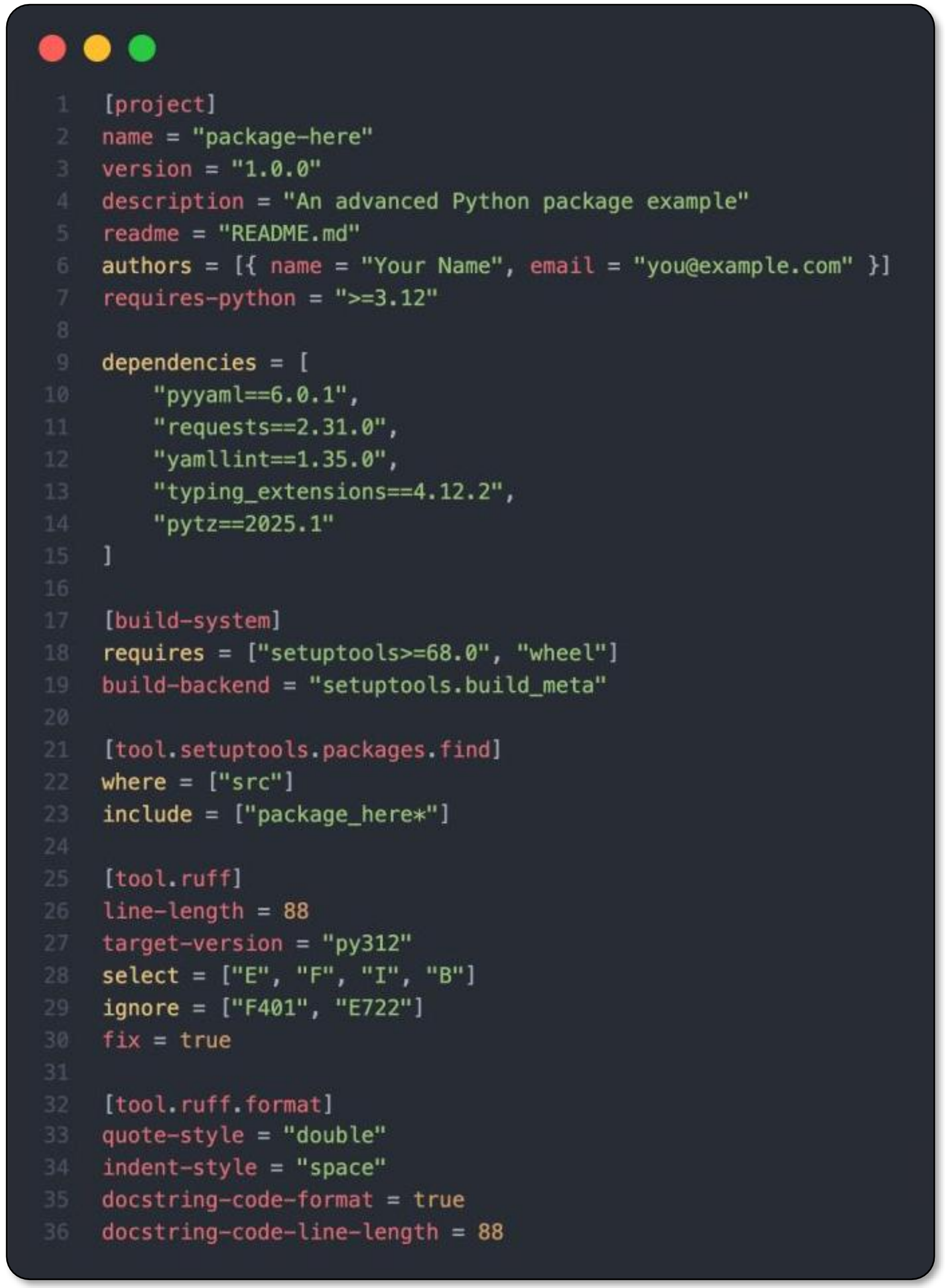
Centralizes dependency management. No more `requirements.txt` drift, manual version pinning, or conflicting files across projects.

3

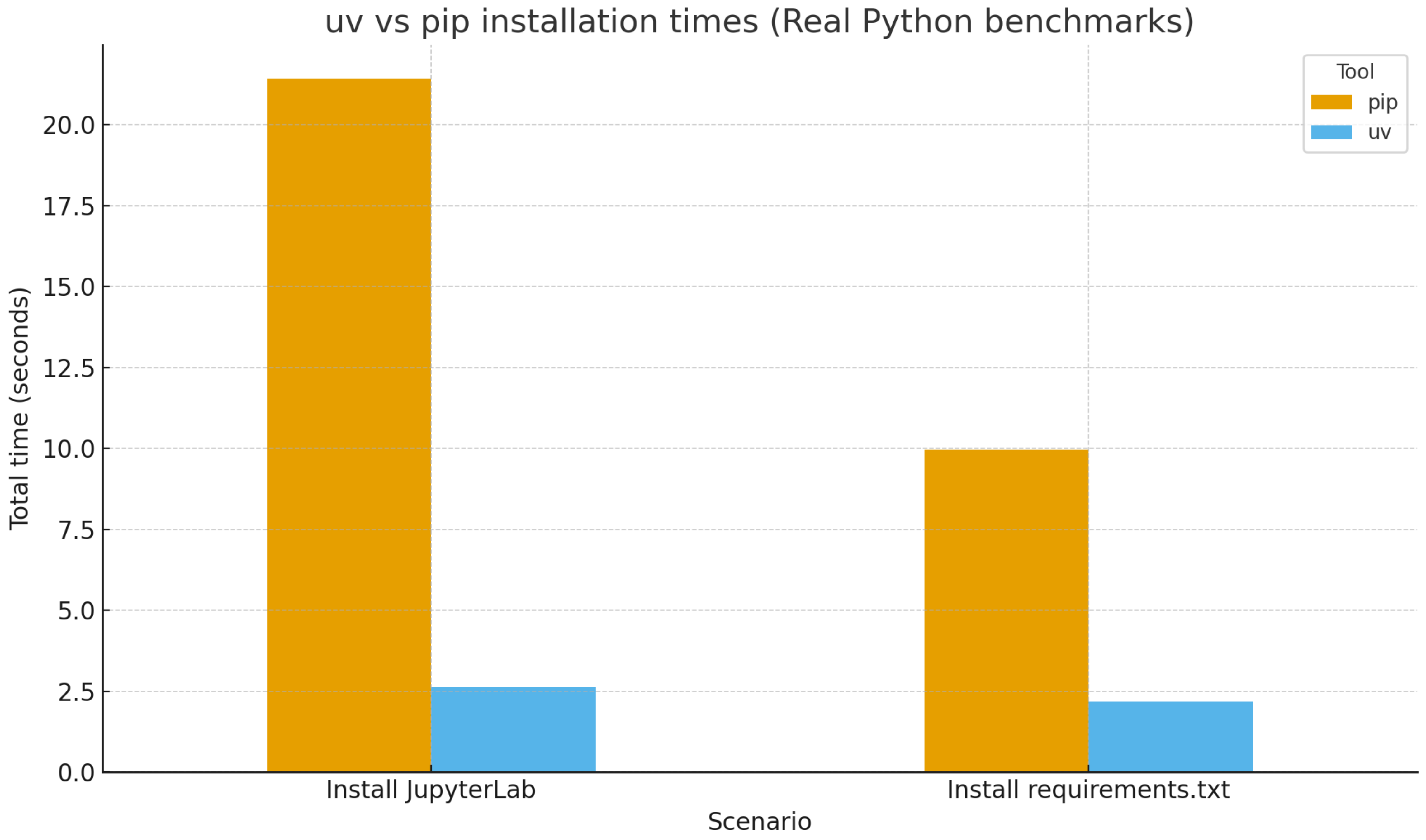
`uv sync`

Makes new project reproducible in minutes. Clone repo, run sync, get identical environment. Works for teammates and CI.

The key file is **pyproject.toml,** your single source of project truth. Dependencies, build configuration, tool settings, and project metadata all live here. One file to understand the entire project setup.



```toml
[project]
name = "package-here"
version = "1.0.0"
description = "An advanced Python package example"
readme = "README.md"
authors = [{ name = "Your Name", email = "you@example.com" }]
requires-python = ">=3.12"

dependencies = [
    "pyyaml==6.0.1",
    "requests==2.31.0",
    "yamllint==1.35.0",
    "typing_extensions==4.12.2",
    "pytz==2025.1"
]

[build-system]
requires = ["setuptools>=68.0", "wheel"]
build-backend = "setuptools.build_meta"

[tool.setuptools.packages.find]
where = ["src"]
include = ["package_here*"]

[tool.ruff]
line-length = 88
target-version = "py312"
select = ["E", "F", "I", "B"]
ignore = ["F401", "E722"]
fix = true

[tool.ruff.format]
quote-style = "double"
indent-style = "space"
docstring-code-format = true
docstring-code-line-length = 88
```


uv vs pip installation times (Real Python benchmarks)
Total time (seconds)
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Install JupyterLab
Install requirements.txt
Scenario
Tool
pip
uv

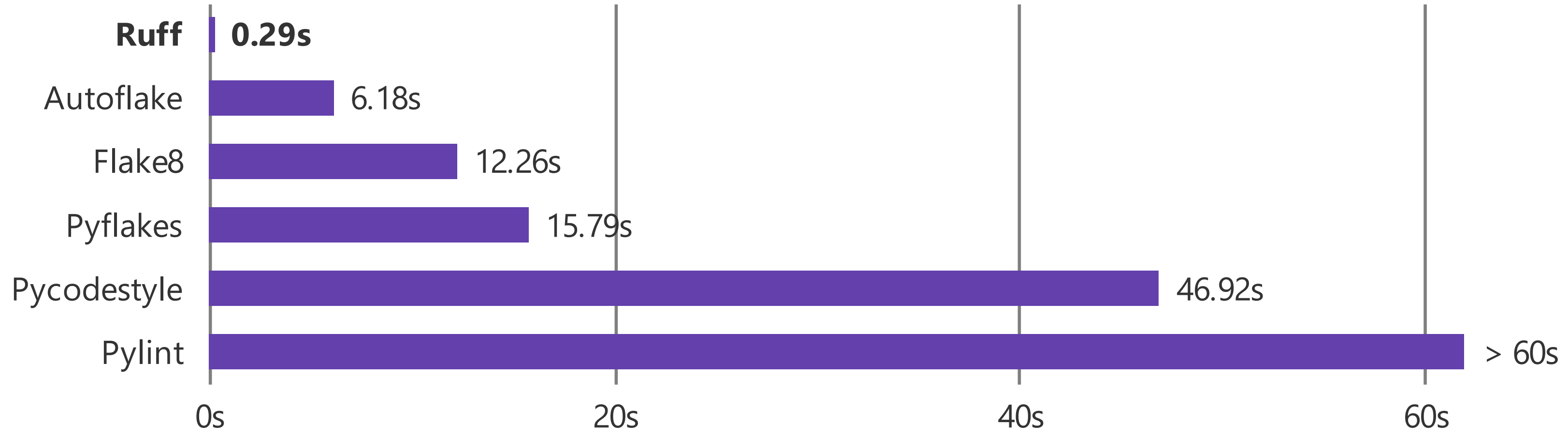
Code linting and formatting with Ruff

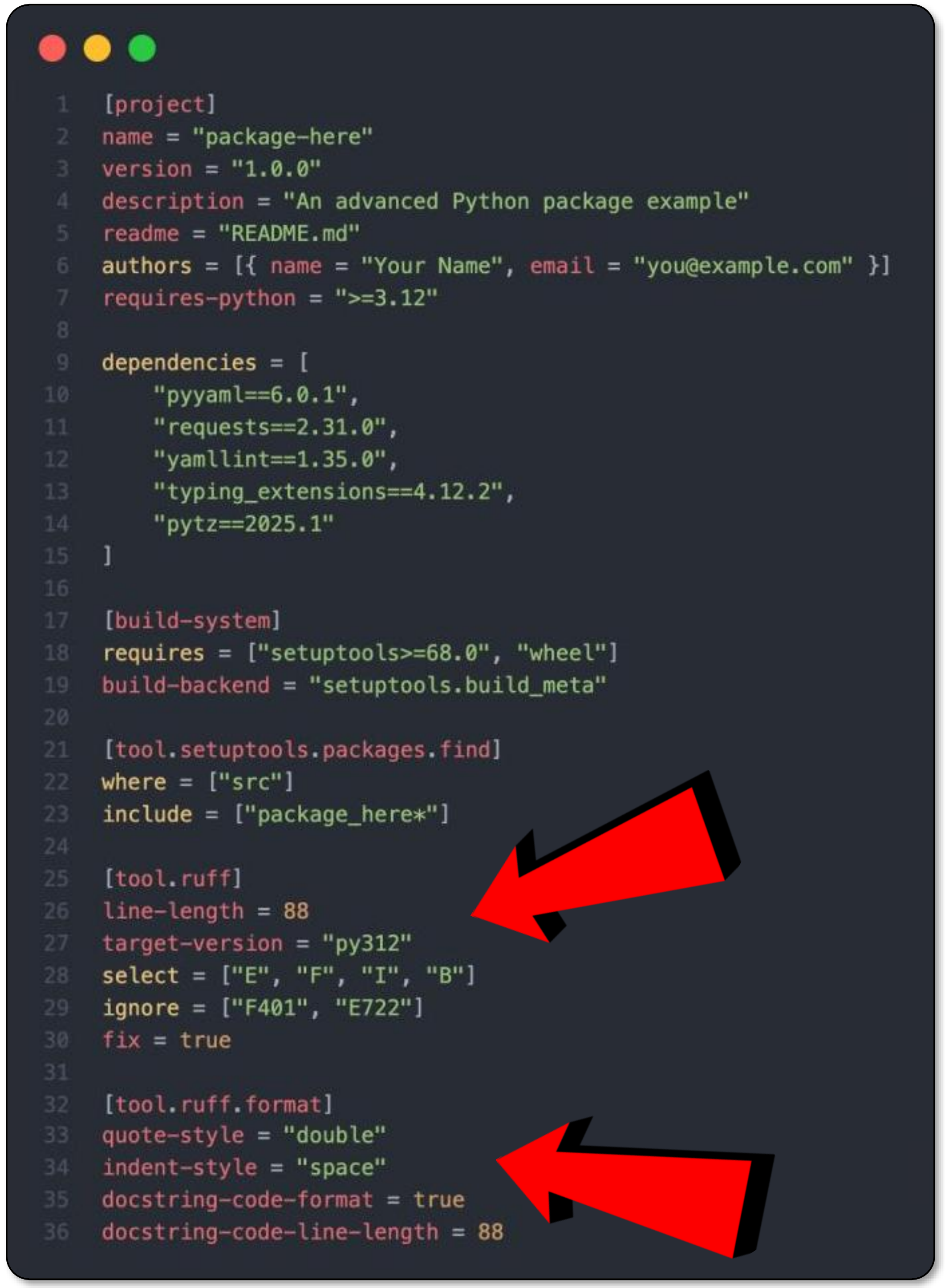
1 `ruff check ‘path/to/project’`

Lint all files in `/path/to/code` (and any subdirectories)

2 `ruff format ‘path/to/project’`

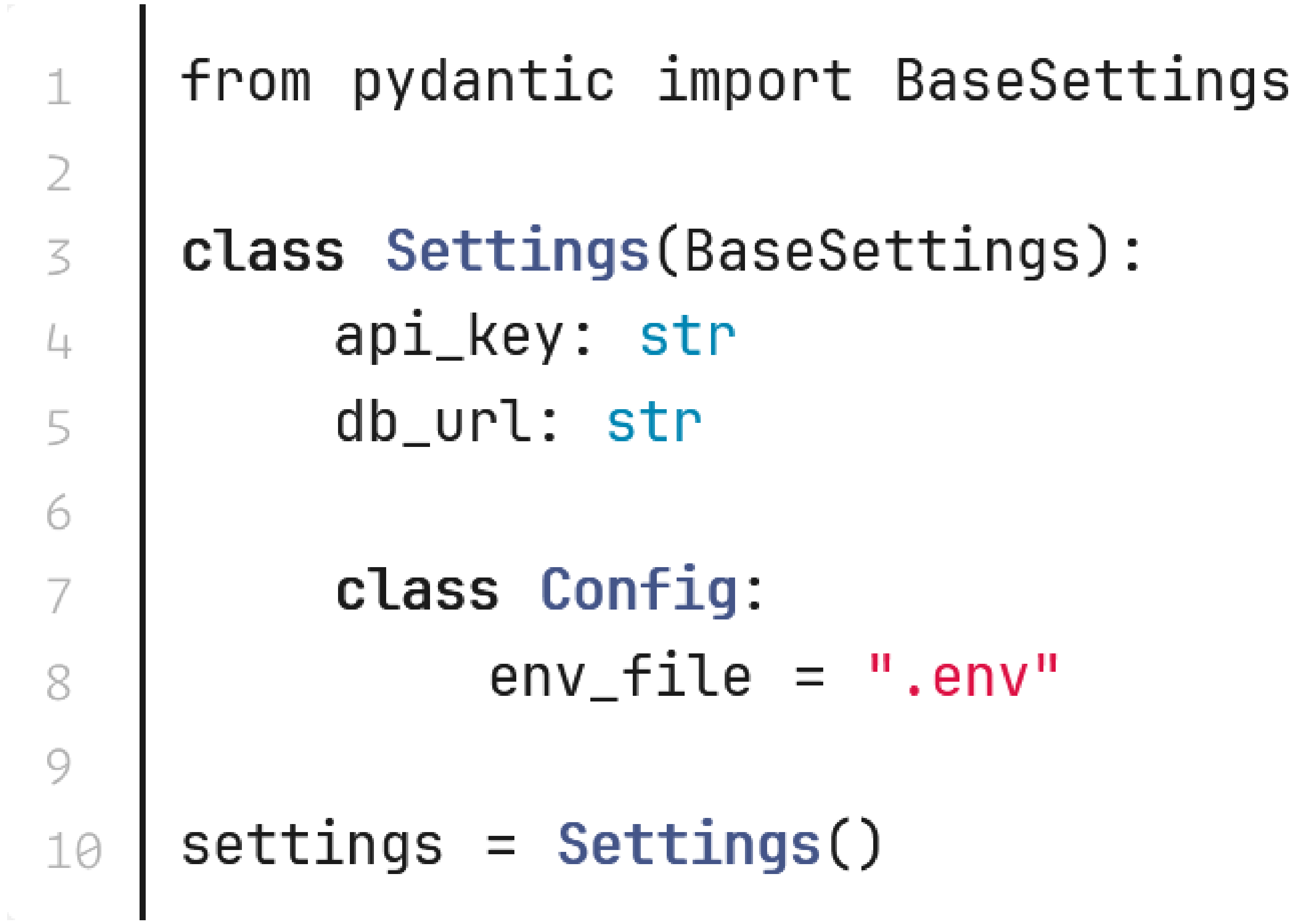
Format all files in `/path/to/code` (and any subdirectories)





```toml
[project]
name = "package-here"
version = "1.0.0"
description = "An advanced Python package example"
readme = "README.md"
authors = [{ name = "Your Name", email = "you@example.com" }]
requires-python = ">=3.12"

dependencies = [
    "pyyaml==6.0.1",
    "requests==2.31.0",
    "yamllint==1.35.0",
    "typing_extensions==4.12.2",
    "pytz==2025.1"
]

[build-system]
requires = ["setuptools>=68.0", "wheel"]
build-backend = "setuptools.build_meta"

[tool.setuptools.packages.find]
where = ["src"]
include = ["package_here*"]

[tool.ruff]
line-length = 88
target-version = "py312"
select = ["E", "F", "I", "B"]
ignore = ["F401", "E722"]
fix = true

[tool.ruff.format]
quote-style = "double"
indent-style = "space"
docstring-code-format = true
docstring-code-line-length = 88
```

Data validation and settings management with Pydantic



```python
from pydantic import BaseSettings

class Settings(BaseSettings):
    api_key: str
    db_url: str

    class Config:
        env_file = ".env"

settings = Settings()
```

Pydantic

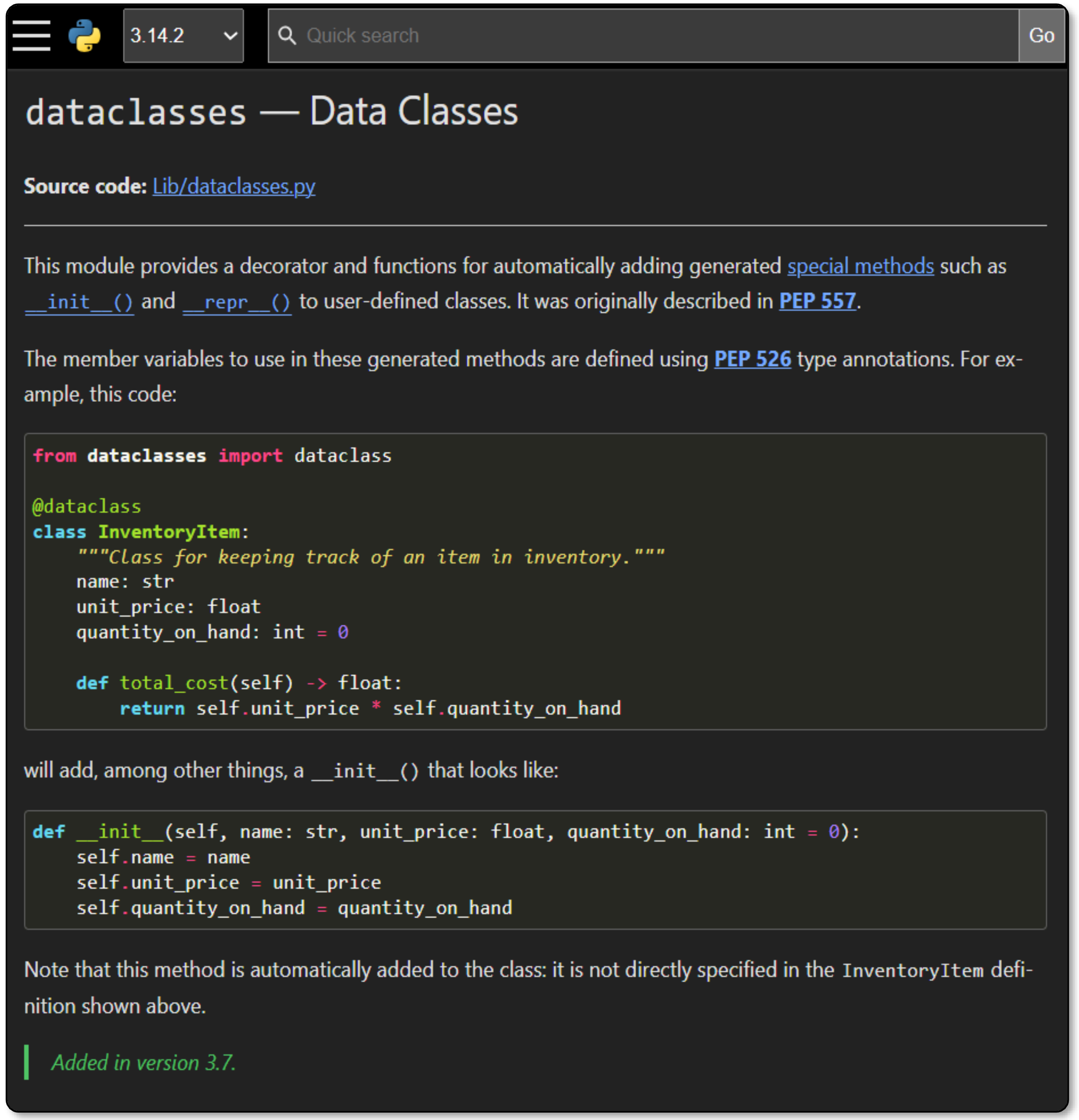
Data validation using Python type hints

https://docs.pydantic.dev/

Dataclasses for storing data

```python
from dataclasses import dataclass

@dataclass
class Point:
    x: int
    y: int

p = Point(1, 2)
print(p)  # Output: Point(x=1, y=2)
```

Dataclasses offer a simple syntax for creating classes that automatically generate special methods like `__init__()`, `__repr__()`, and `__eq__()`.



dataclasses — Data Classes

Source code: Lib/dataclasses.py

This module provides a decorator and functions for automatically adding generated special methods such as `__init__()` and `__repr__()` to user-defined classes. It was originally described in PEP 557.

The member variables to use in these generated methods are defined using PEP 526 type annotations. For example, this code:

```python
from dataclasses import dataclass

@dataclass
class InventoryItem:
    """Class for keeping track of an item in inventory."""
    name: str
    unit_price: float
    quantity_on_hand: int = 0

    def total_cost(self) -> float:
        return self.unit_price * self.quantity_on_hand
```

will add, among other things, a `__init__()` that looks like:

```python
def __init__(self, name: str, unit_price: float, quantity_on_hand: int = 0):
    self.name = name
    self.unit_price = unit_price
    self.quantity_on_hand = quantity_on_hand
```

Note that this method is automatically added to the class: it is not directly specified in the `InventoryItem` definition shown above.

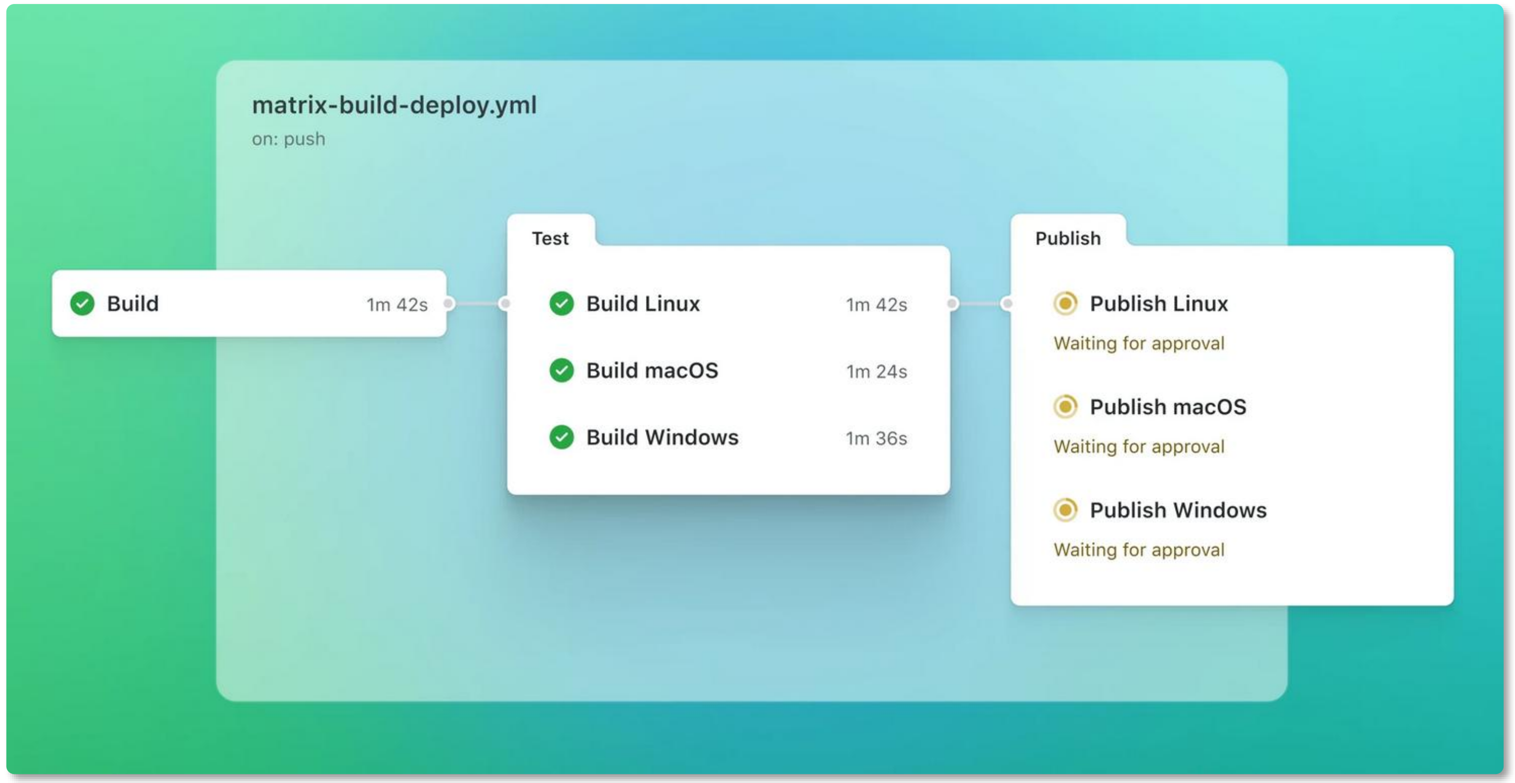
Added in version 3.7.

https://docs.python.org/3/library/dataclasses.html

Version control 🔁

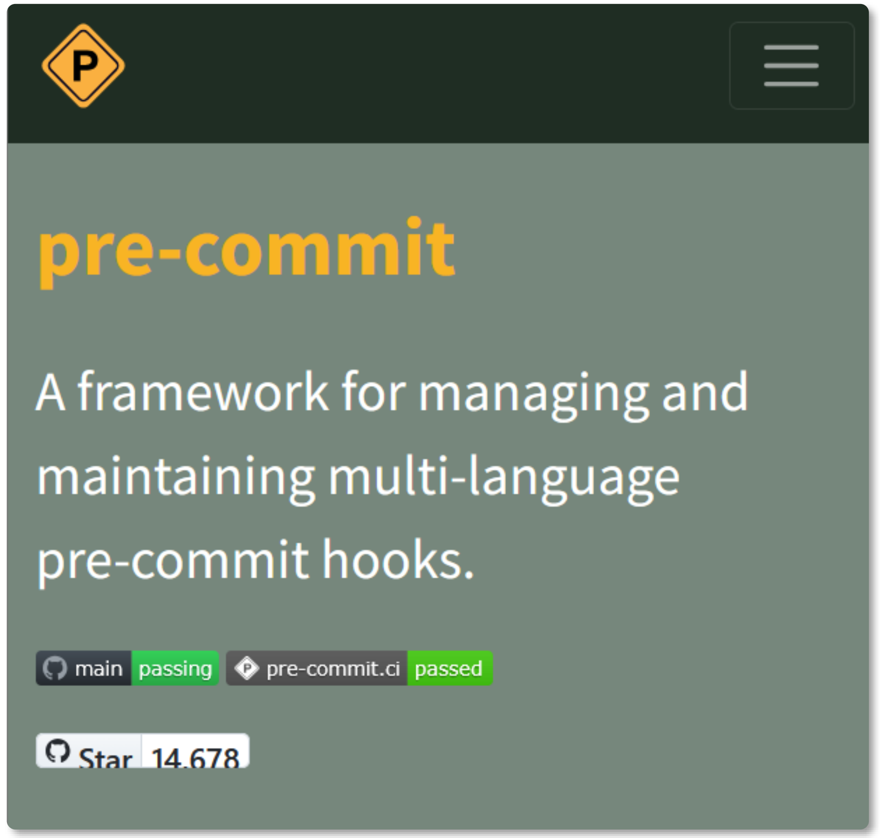
GitHub Actions

```yaml
name: CI-API

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  build-and-test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3
      - name: Build Docker image
        run: docker build -t project-api:ci ./project-api
      - name: Run tests
        run: docker run --rm project-api:ci pytest
```



Pre-commit hooks

```yaml
repos:

- repo: <https://github.com/astral-sh/ruff-pre-commit>
    rev: v0.12.3  # Ruff version.
    hooks:
  - id: ruff-check # Run the linter.
        args: [ --fix ]
  - id: ruff-format  # Run the formatter.
- repo: <https://github.com/gitleaks/gitleaks>
    rev: v8.27.2
    hooks:
  - id: gitleaks
```



pre-commit

A framework for managing and maintaining multi-language pre-commit hooks.

Open Source Secret Scanning

Gitleaks is an open source (MIT licensed) secret scanner for git repositories, files, directories, and stdin. With over 20 million docker downloads, 19k GitHub stars, 14 million GitHub downloads, thousands of weekly clones, and over 850k homebrew installs, gitleaks is the most trusted open source secret scanner among security professionals, enterprises, and developers. Gitleaks is maintained by Zach Rice.

Infrastructure Management 🏗️

Use make for cheap automation

```
DIR := . # project/project-api/Makefile

test:
 uv run pytest

format-fix:
 uv run ruff format $(DIR)
 uv run ruff check --select I --fix

lint-fix:
 uv run ruff check --fix
```

```
infrastructure-build:
 docker compose build

infrastructure-up:
 docker compose up --build -d

infrastructure-stop:
 docker compose stop
```

Next Step: Build!

1 Commit To One Project

Pick one upcoming project and commit to this toolchain. Real learning happens through applied practice, not passive reading.

2 Start Small

Begin with the skeleton repo and resist yak-shaving for one full week. Ship something imperfect rather than polish something imaginary.

3 Measure Speed

Track time from first business question to first callable endpoint. This metric reveals where your process helps or hinders.

4 Share The Template

Publish your template internally so others skip the suffering. Your documentation of lessons learned becomes the team's shared knowledge.

Q&A