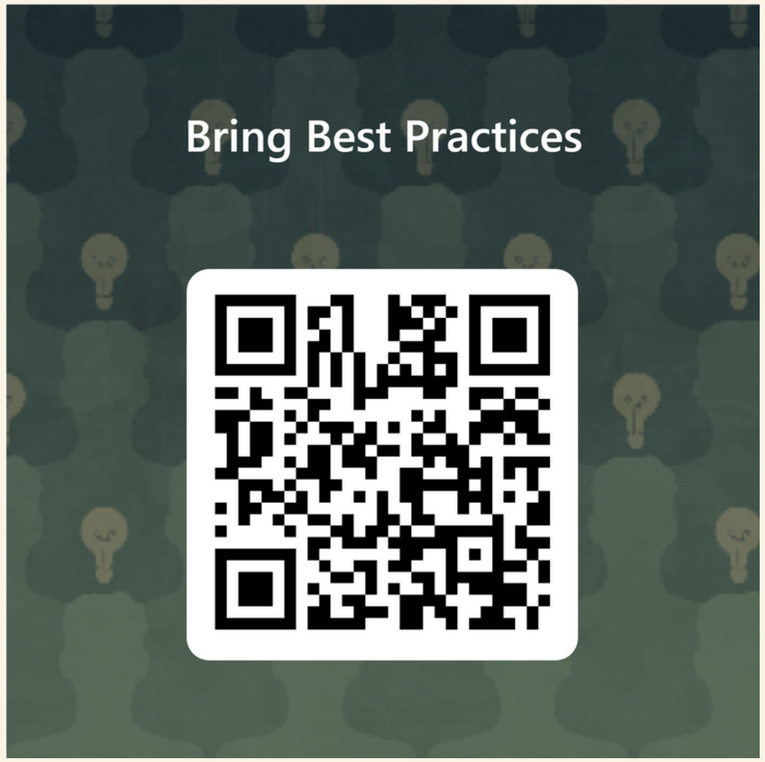
welcome to

BRING BEST PRACTICES TO YOUR MESSY DATA SCIENCE TEAM!

with Reproducibility, Compliance, and Consistency in Mind

Live Poll



FORMS.OFFICE.COM/R/V8VUEWPOBP

introduce

ABOUT ME

A Husband, a Dad for 6yo and 2yo girls, a Teacher, an Archer and Mountain Biker

PhD from King's College London in Medical Imaging Sciences in 2015

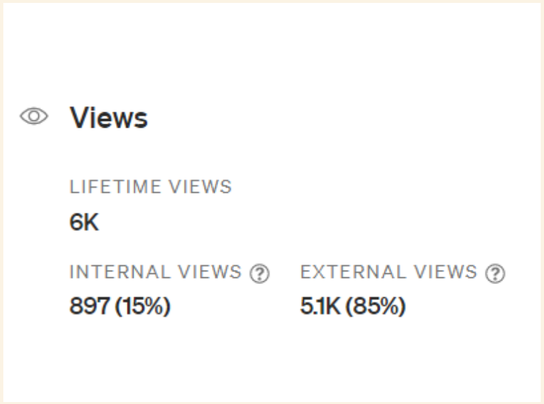
Lead Data Scientist and **MlOps** at **NIQ BrandBank**

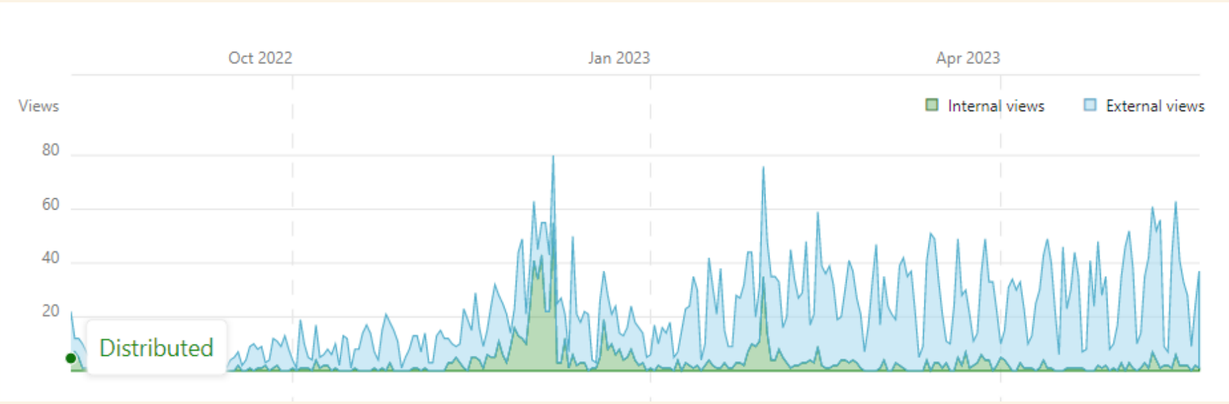
GABRIEL HARRIS

Python HOW: Using Poetry, Make, and pre-commit-hooks to Setup a Repo Template for your Team

Bring consistency, rigour, and best practices to your messy data science team

Medium / Aug 5, 2022





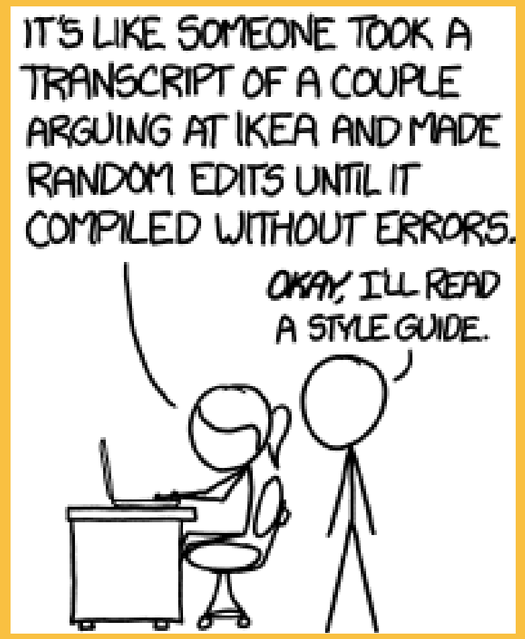
why?

¯_(ツ)_/¯

IT WORKS
on my machine

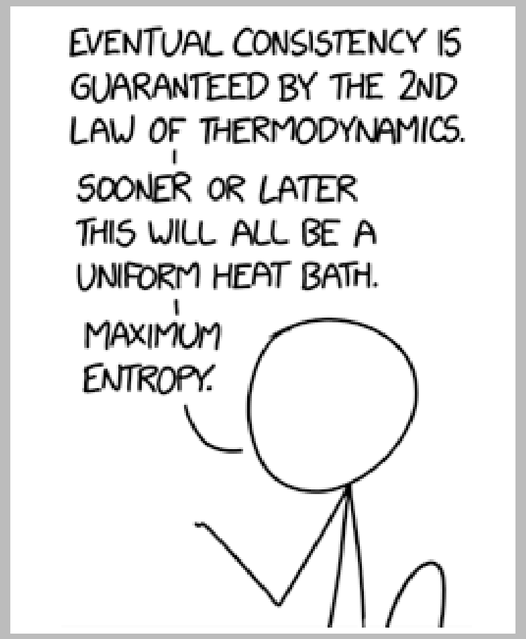
REPRODUCIBILITY

why?



COMPLIANCE

why?



CONSISTENCY

how?

PIP-TOOLS

JAZZ BAND

REPRODUCIBILITY

how?

PRE-COMMIT-HOOKS

COMPLIANCE

how?

GNU MAKE

CONSISTENCY

breathe

PRE PROJECT

HELPER PACKAGES

- BLACK
- ISORT
- PYDOCSTYLE
- RUFF
- SQLFLUFF
- PRE-COMMIT

BLACK

30.4K

Stars

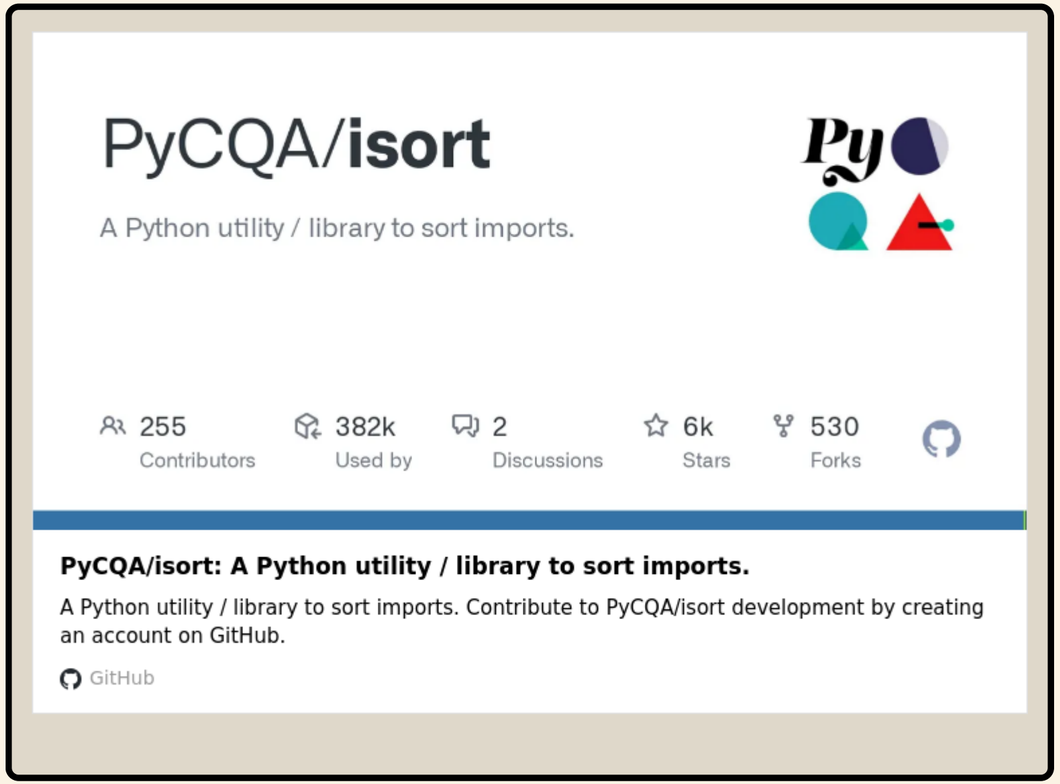
"Black is the uncompromising Python code formatter. By using it, you agree to cede control over minutiae of hand-formatting. In return, Black gives you speed, determinism, and freedom from **pycodestyle** nagging about formatting. You will save time and mental energy for more important matters."

ISORT

5.8K

Stars

"isort sorts imports alphabetically, and automatically separated into sections and by type."



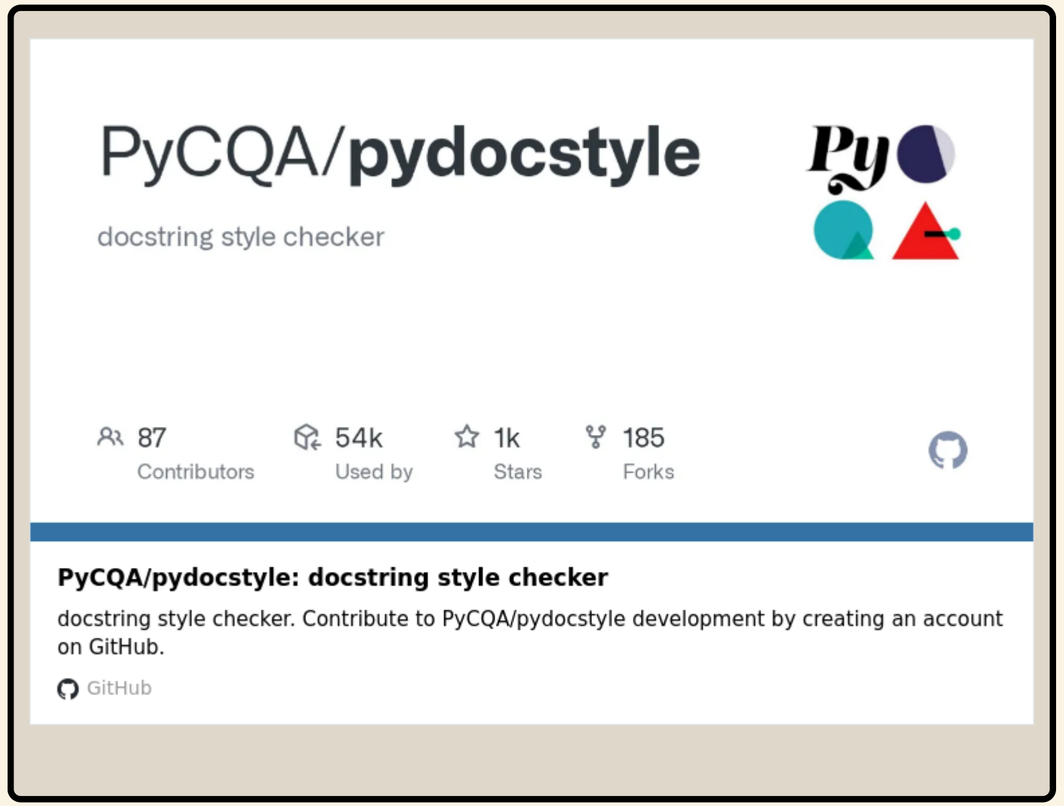
PYDOCSTYLE

1K

Stars

"pydocstyle is a static analysis tool for checking compliance with Python docstring conventions."

“There are three conventions that may be used by pydocstyle: **pep257**, **numpy** and **google**”.

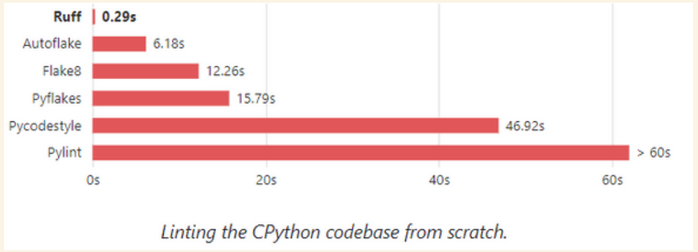


RUFF

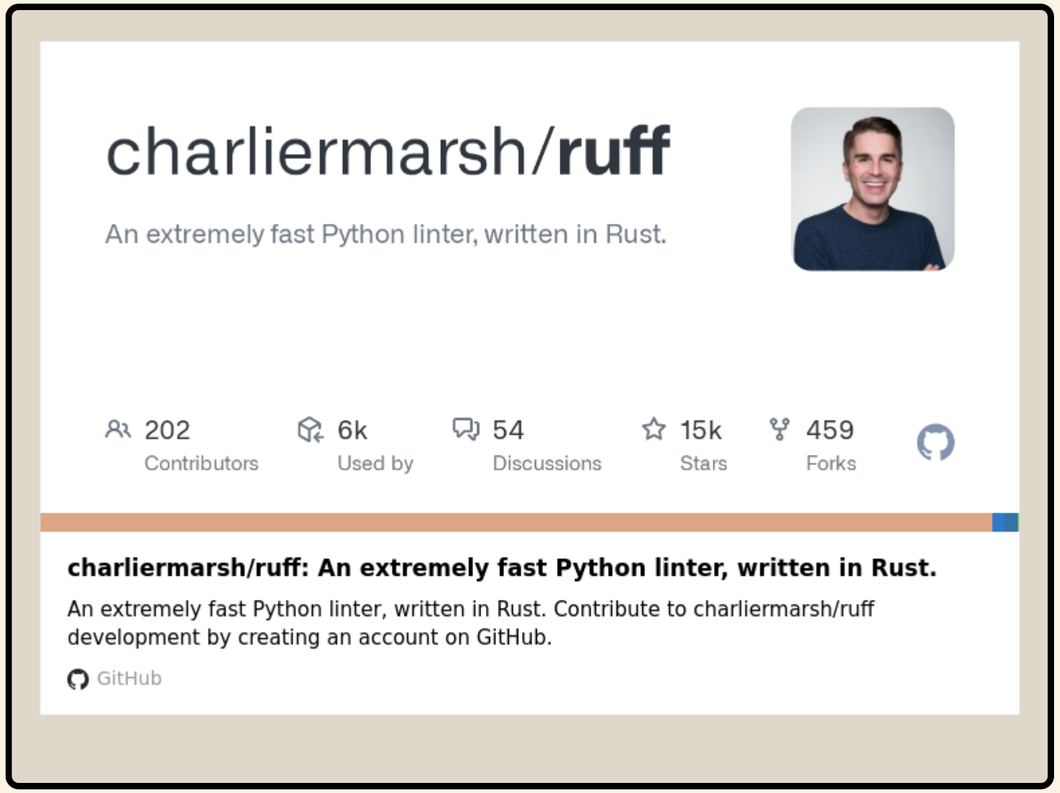
15K

Stars

"An extremely fast Python linter, written in **Rust**. Can be used to replace Flake8, isort, pydocstyle (plus dozens of plugins) all while executing tens or hundreds of times faster than any individual tool."



Linting the CPython codebase from scratch.

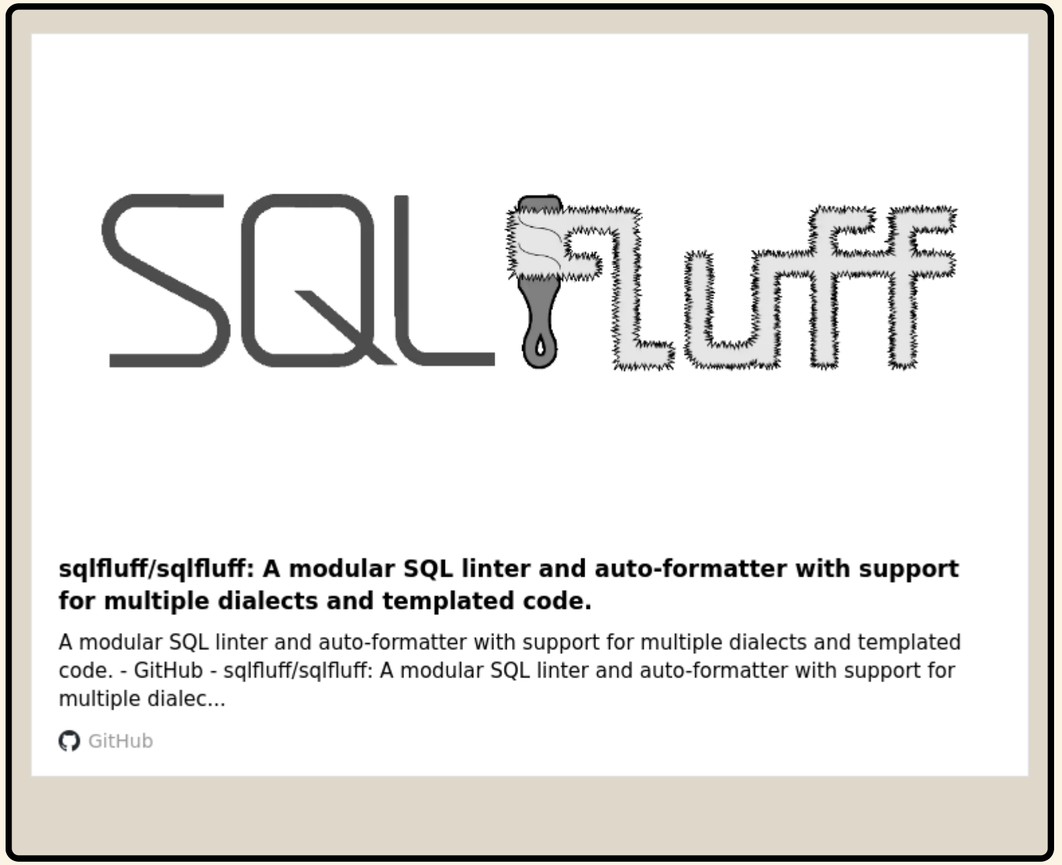


SQLFLUFF

6.1K

Stars

"SQLFluff is a dialect-flexible and configurable SQL linter. Designed with ELT applications in mind, SQLFluff also works with Jinja templating and dbt. SQLFluff will auto-fix most linting errors, allowing you to focus your time on what matters."

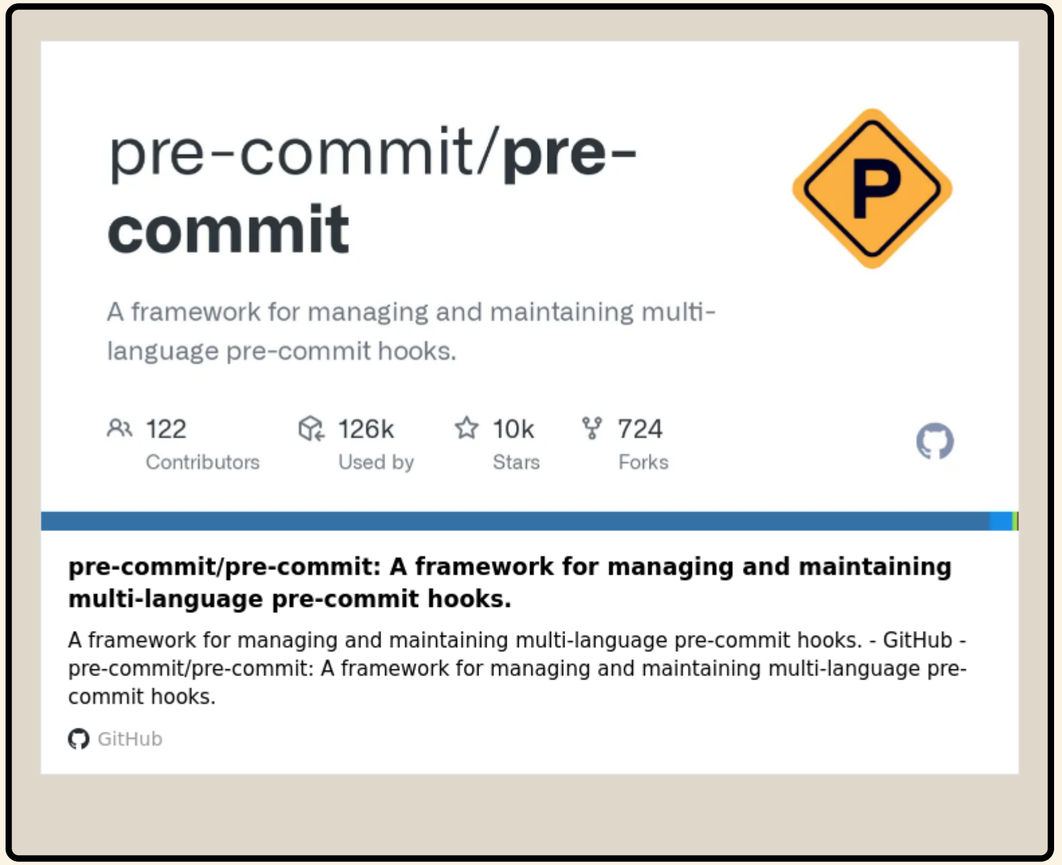


PRE-COMMIT

10.4K

Stars

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



breathe

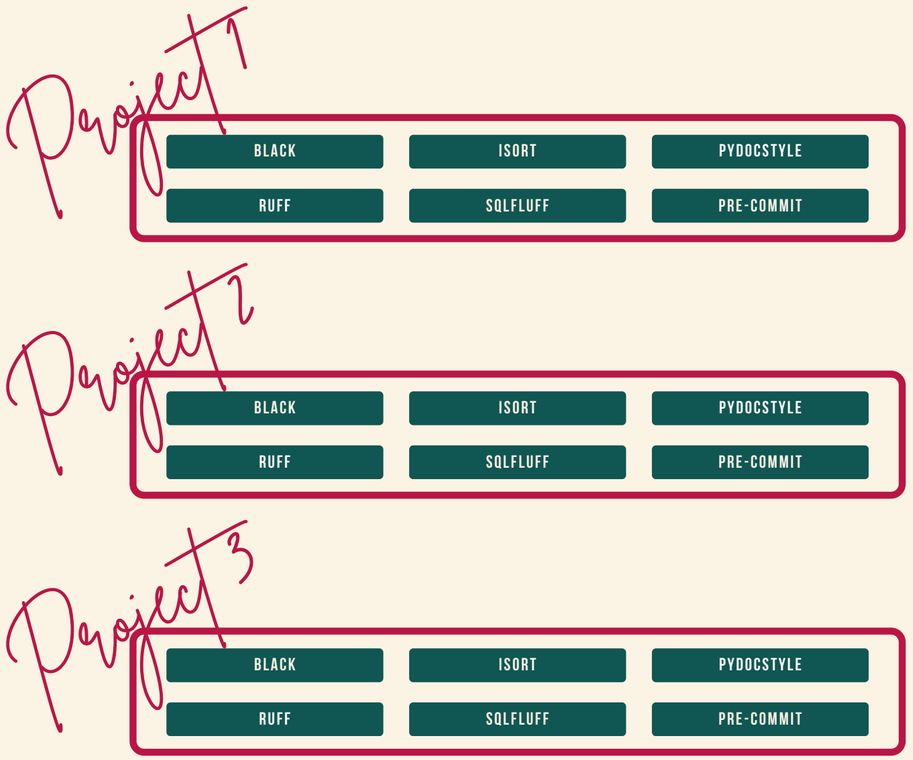
HELPER PACKAGES

INSTALLATION

HELPER PACKAGES PER PROJECT

"Install packages for each project."

We can do better

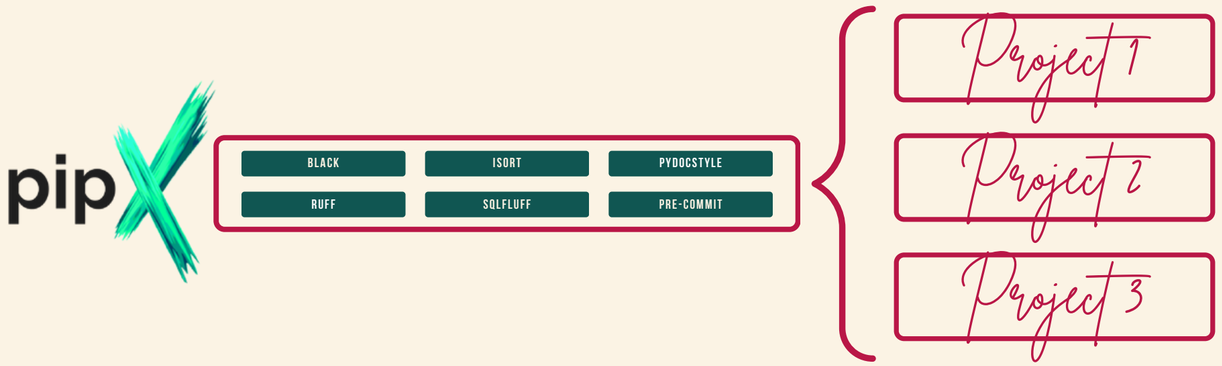


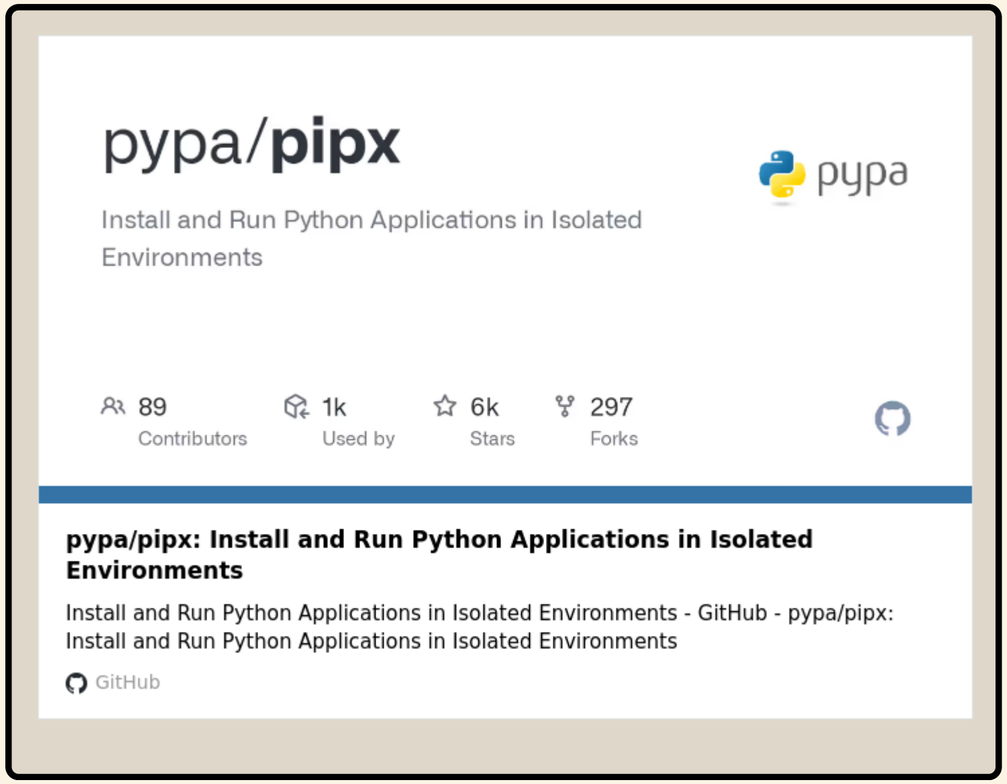
HELPER PACKAGES WITH PIPX

6.5K

Stars

"**PIPX** installs packages in isolated environments that can be run globally."





HELPER PACKAGES WITH PIPX

Create virtual environment for pipx

- It's best to create a virtual environment just for pipx in your $HOME\.venv
- It doesn't matter which python version you choose. We will use 3.11.

```
> py -3.11 -m venv $HOME\.venv\pipx.venv
```

HELPER PACKAGES WITH PIPX

Install pipx

- Activate pipx.venv and install pipx

```
> $HOME\.venv\pipx.venv\Scripts\Activate.ps1
(pipx.venv) > pip install pipx
```

- This installs **pipx.exe** in $HOME\.venv\pipx.venv\Scripts which is not in $PATH
- To fix this, add it to $PATH and check it works

```
(pipx.venv) > pipx --version
1.2.0
```

HELPER PACKAGES WITH PIPX

Install helper packages

```
(pipx.venv) > pipx install black
(pipx.venv) > pipx install isort
(pipx.venv) > pipx install pydocstyle
(pipx.venv) > pipx install ruff
(pipx.venv) > pipx install sqlfluff
(pipx.venv) > pipx install pre-commit
```

- pipx installs package executables in $HOME\.local\bin which is not in $PATH
- To fix this, add it to $PATH , or run pipx ensurepath

```
(pipx.venv) > pipx ensurepath
Success! Added C:\Users\<username>\.local\bin to the PATH environment variable.
You will need to open a new terminal or re-login for the PATH changes to take effect.
Otherwise pipx is ready to go! ✨    ✨
```

HELPER PACKAGES WITH PIPX

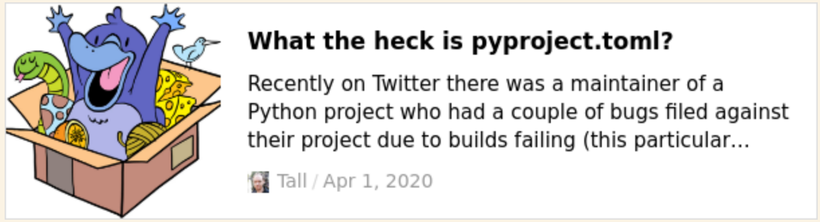
List installed packages

```
> pipx list
venvs are in C:\Users\<username>\.local\pipx\venvs
apps are exposed on your $PATH at C:\Users\<username>\.local\bin
  package black 23.3.0, installed using Python 3.11.3
   - black.exe
   - blackd.exe
  package flake8 6.0.0, installed using Python 3.11.3
   - flake8.exe
  package isort 5.12.0, installed using Python 3.11.3
   - isort-identify-imports.exe
   - isort.exe
  package pip-chill 1.0.3, installed using Python 3.11.3
   - pip-chill.exe
  package pre-commit 3.3.1, installed using Python 3.11.3
   - pre-commit.exe
  package pydocstyle 6.3.0, installed using Python 3.11.3
   - pydocstyle.exe
  package ruff 0.0.269, installed using Python 3.11.3
   - ruff.exe
  package sqlfluff 2.0.7, installed using Python 3.11.3
   - sqlfluff.exe
```

HELPER PACKAGES

CONFIGURATIONS

HELPER PACKAGES
WITH PYPROJECT.TOML



"the purpose of PEP 518 was to come up with a way for projects to specify what build tools they required … all projects should (eventually) have … development tools realized they … could put their own configuration"

- Brett Cannon, Python Steering Council

pyproject.toml

```toml
[tool.black]
line-length = 88

[tool.isort]
profile = "black"

[tool.pydocstyle]
convention = "google"
add-ignore = "D100,D101,D102,D103,D104,D105,D106,D107"

[tool.ruff]
line-length = 88

[tool.sqlfluff.core]
dialect = "tsql"
exclude_rules = "L016,L031"
max_line_length = 88
```

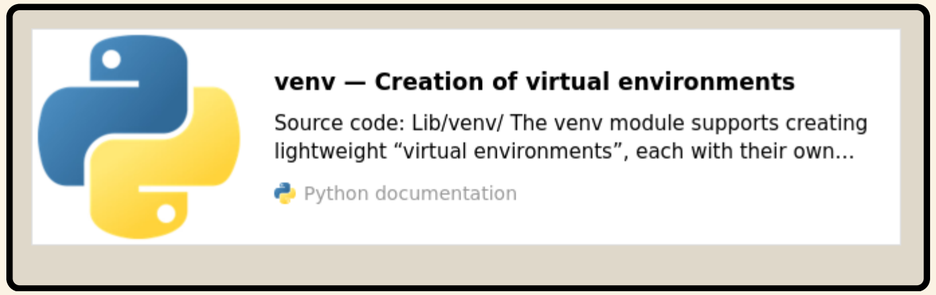
breathe

NEW PROJECT

VENV MODULE

The **venv** module, proposed in PEP 405, supports creating lightweight “virtual environments”, each with their own independent set of Python packages installed in their site directories.

You do isolate packages, right? right?




```
> py -3.10 -m venv $path\to\project\.venv
> $path\to\project\.venv\Scripts\Activate.ps1
(.venv) >
```

how?

PIP-TOOLS

JAZZ BAND

REPRODUCIBILITY

how?

PRE-COMMIT-HOOKS

COMPLIANCE

how?

GNU MAKE

CONSISTENCY

PIP-TOOLS

6.9K

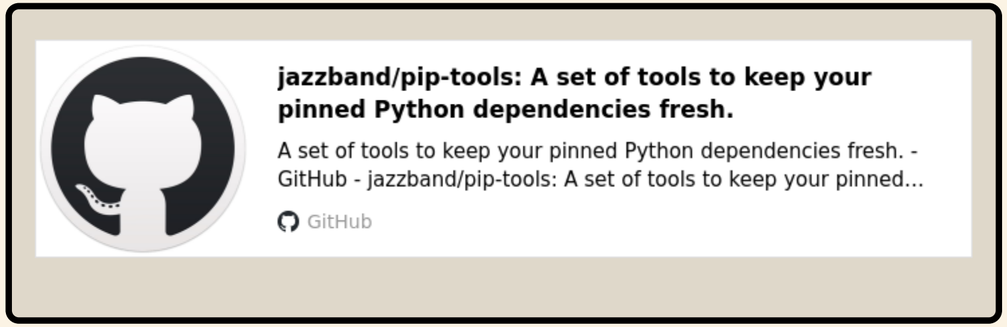
Stars

"A set of command line tools to help you keep your pip-based packages fresh, even when you've pinned them."

You do pin packages, right? right?

```
(.venv) > pip install pip-tools
```

Installed in each Project

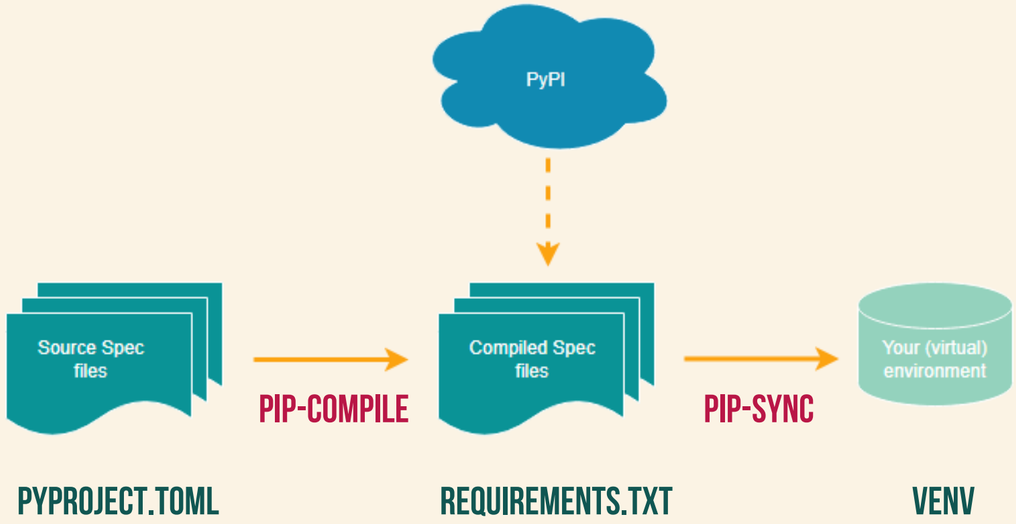


PIP-TOOLS =

PIP-COMPILE + PIP-SYNC

The PIP-COMPLIE command compiles a **requirements.txt** file from your dependencies, specified in either **pyproject.toml**, setup.cfg, setup.py, or requirements.in

The PIP-SYNC command syncs your **venv** to reflect exactly what's in **requirements.txt**. This will install/upgrade/uninstall everything necessary to match requirements.txt contents



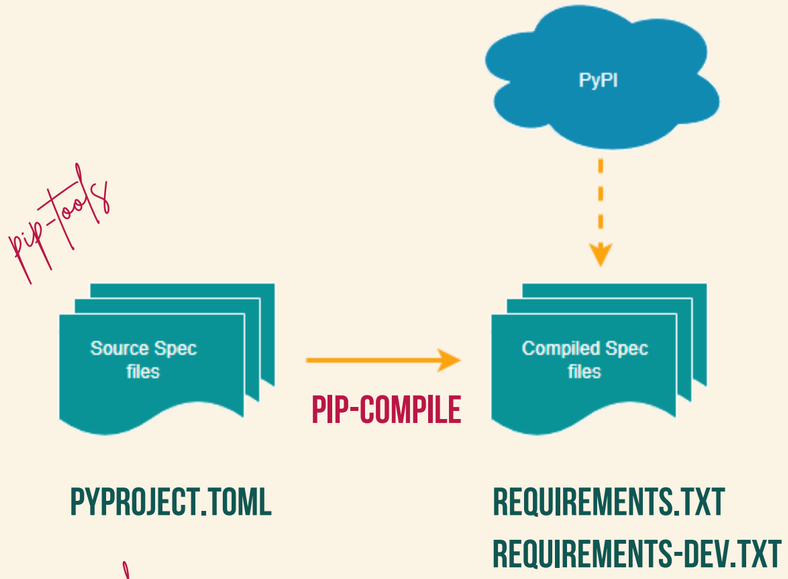
PIP-COMPILE

Generates **requirements.txt** using the latest versions that fulfil the dependencies specified in **pyproject.toml**

```
(.venv) > pip-compile pyproject.toml --output-file requirements.txt
```

Generates **requirements-dev.txt** using the latest versions that fulfil dependencies + optional-dependencies specified in **pyproject.toml**

```
(.venv) > pip-compile pyproject.toml --output-file requirements-dev.txt  --extra dev
```



pyproject.toml

```toml
dependencies = [
    "pandas",
    "xlrd",
    ]

[project.optional-dependencies]
dev = [
    "ipykernel",
    "ipython",
    ]
```

PIP-COMPILE

If an existing **requirements.txt** file is found that fulfils the dependencies, then no changes will be made.
To update all packages in **requirements.txt** to the latest version

```
(.venv) > pip-compile --upgrade
```

To update a specific package to the latest or a specific version

```
(.venv) > pip-compile --upgrade-package pandas
(.venv) > pip-compile --upgrade-package pandas==2.0.0
```

Combine both to provide constraints on the allowed upgrades

```
(.venv) > pip-compile --upgrade --upgrade-package pandas==2.0.0
```

PIP-COMPILE

PYPROJECT.TOML

pyproject.toml

```toml
dependencies = [
    "pandas",
    "xlrd",
    ]

[project.optional-dependencies]
dev = [
    "ipykernel",
    "ipython",
    ]
```

REQUIREMENTS.TXT

requirements.txt

```
#
# This file is autogenerated by pip-compile with Python 3.10
# by the following command:
#
#    pip-compile --allow-unsafe --output-file=requirements.txt --resolver=backtracking pyproject.toml
#
numpy==1.24.3
    # via pandas
pandas==2.0.2
    # via Template (pyproject.toml)
python-dateutil==2.8.2
    # via pandas
pytz==2023.3
    # via pandas
six==1.16.0
    # via python-dateutil
tzdata==2023.3
    # via pandas
xlrd==2.0.1
    # via Template (pyproject.toml)
```

dependencies

REQUIREMENTS-DEV.TXT

requirements-dev.txt

```
#
# This file is autogenerated by pip-compile with Python 3.10
# by the following command:
#
#    pip-compile --allow-unsafe --extra=dev --output-file=requirements-dev.txt --resolver=backtracking pyproject.toml
#
asttokens==2.2.1
    # via stack-data
backcall==0.2.0
    # via ipython
colorama==0.4.6
    # via ipython
comm==0.1.3
    # via ipykernel
debugpy==1.6.7
    # via ipykernel
decorator==5.1.1
    # via ipython
executing==1.2.0
    # via stack-data
ipykernel==6.23.1
    # via Template (pyproject.toml)
ipython==8.13.2
    # via
```

dependencies + optional dependencies

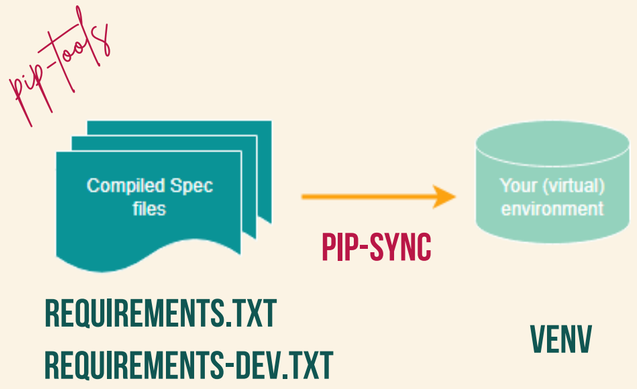
PIP-SYNC

Update **venv** to reflect exactly what's in **requirements.txt**. This will install, upgrade, and uninstall everything necessary

```
(.venv) > pip-sync requirements.txt
```

Or what's in **requirements-dev.txt**

```
(.venv) > pip-sync requirements-dev.txt
```



breathe

how?

PIP-TOOLS

JAZZ BAND

REPRODUCIBILITY

how?

PRE-COMMIT-HOOKS

COMPLIANCE

how?

GNU MAKE

CONSISTENCY

PRE-COMMIT-HOOKS

"Git hook scripts are useful for identifying simple issues before submission to code review. By pointing these issues out before code review, this allows a code reviewer to focus on the architecture of a change while not wasting time with trivial style nitpicks."

Install pre-commit package


```
(pipx.venv) > pipx install pre-commit
```

PRE-COMMIT-HOOKS

Add pre-commit configuration

Specify hook in .pre-commit-config.yaml

Hook

```yaml
- repo: <link to a git package with a .pre-commit-hooks.yaml>
  rev: <package version>
  hooks:
  -   id: hook id found in .pre-commit-hooks.yaml
  -   arg: list of parameters to pass to the hook  (default [])
```

.PRE-COMMIT-CONFIG.YAML

.pre-commit-config.yaml

```yaml
  - repo: https://github.com/PyCQA/isort
    rev: '5.12.0'
    hooks:
    -   id: isort
        args: ['--profile=black']
```

PRE-COMMIT-HOOKS

BLACK

.PRE-COMMIT-CONFIG.YAML

```yaml
- repo: https://github.com/psf/black
  rev: '23.3.0'
  hooks:
  -   id: black
  -   id: black-jupyter
```

BLACK/.PRE-COMMIT-HOOKS.YAML

```yaml
- id: black
  name: black
  description: "Black: The uncompromising Python code formatter"
  entry: black
  language: python
  minimum_pre_commit_version: 2.9.2
  require_serial: true
  types_or: [python, pyi]
- id: black-jupyter
  name: black-jupyter
  description:
    "Black: The uncompromising Python code formatter (with Jupyter Notebook support)"
  entry: black
  language: python
  minimum_pre_commit_version: 2.9.2
  require_serial: true
  types_or: [python, pyi, jupyter]
  additional_dependencies: [".[jupyter]"]
```

PRE-COMMIT-HOOKS

PYDOCSTYLE

.PRE-COMMIT-CONFIG.YAML

```yaml
- repo: https://github.com/pycqa/pydocstyle
  rev: '6.3.0'
  hooks:
  -  id: pydocstyle
  -  args: ['--add-ignore=D100,D101,D102,D103,D104,D105,D106,D107',
'--convention=google']
```

PYDOCSTYLE/.PRE-COMMIT-HOOKS.YAML

```yaml
-   id: pydocstyle
    name: pydocstyle
    description: pydocstyle is a static analysis tool for checking compliance with Python docstring conventions.
    entry: pydocstyle
    language: python
    types: [python]
```

PRE-COMMIT-HOOKS

RUFF

.PRE-COMMIT-CONFIG.YAML

```yaml
- repo: https://github.com/charliermarsh/ruff-pre-commit
  rev: 'v0.0.270'
  hooks:
  -   id: ruff
```

RUFF-PRE-COMMIT/.PRE-COMMIT-HOOKS.YAML

```yaml
-   id: ruff
    name: ruff
    description: "Run 'ruff' for extremely fast Python linting"
    entry: ruff check --force-exclude
    language: python
    'types_or': [python, pyi]
    args: []
    require_serial: true
    additional_dependencies: []
    minimum_pre_commit_version: '2.9.2'
```

PRE-COMMIT-HOOKS

SQLFLUFF

.PRE-COMMIT-CONFIG.YAML

```yaml
- repo: https://github.com/sqlfluff/sqlfluff
  rev: '2.1.1'
  hooks:
  -  id: sqlfluff-fix
  -  args: ['--dialect=tsql', '--exclude_rules=L016,L031']
  -  id: sqlfluff-lint
  -  args: ['--dialect=tsql', '--exclude_rules=L016,L031']
```

SQLFLUFF/.PRE-COMMIT-HOOKS.YAML

```yaml
- id: sqlfluff-lint
  name: sqlfluff-lint
  # Set `--processes 0` to use maximum parallelism
  entry: sqlfluff lint --processes 0
  language: python
  description: "Lints sql files with `SQLFluff`"
  types: [sql]
  require_serial: true
  additional_dependencies: []

- id: sqlfluff-fix
  name: sqlfluff-fix
  # Set a couple of default flags:
  #  - `--force` to disable confirmation
  #  - `--show-lint-violations` shows issues to not require running `sqlfluff lint`
  #  - `--processes 0` to use maximum parallelism
  # By default, this hook applies all rules.
  entry: sqlfluff fix --force --show-lint-violations --processes 0
  language: python
  description: "Fixes sql lint errors with `SQLFluff`"
  types: [sql]
  require_serial: true
  additional_dependencies: []
```

PRE-COMMIT-HOOKS

PRE-COMMIT-HOOKS

.pre-commit-config.yaml

```yaml
- repo: https://github.com/pre-commit/pre-commit-hooks
  rev: 'v4.4.0'
  hooks:
  -  id: check-ast
  -  id: check-json
  -  id: check-toml
  -  id: check-yaml
  -  id: end-of-file-fixer
  -  id: trailing-whitespace
```

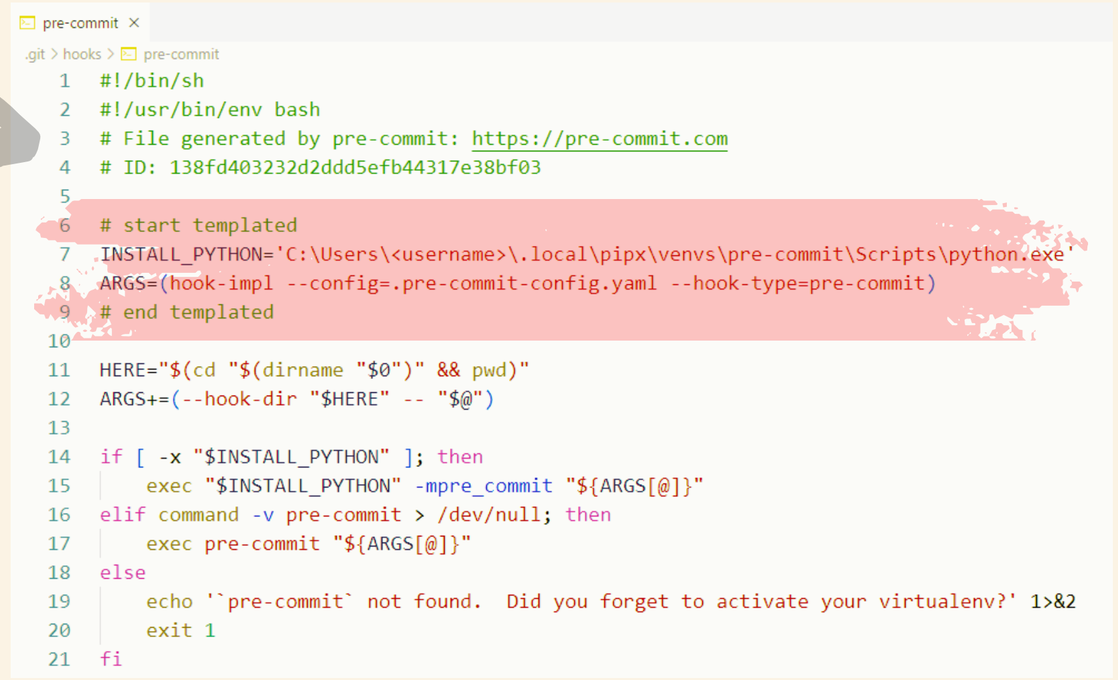
pre-commit-hooks/.pre-commit-hooks.yaml

```yaml
7   -   id: check-ast
8       name: check python ast
9       description: simply checks whether the files parse as valid python.
10      entry: check-ast
11      language: python
12      types: [python]
43  -   id: check-json
44      name: check json
45      description: checks json files for parseable syntax.
46      entry: check-json
47      language: python
48      types: [json]
74  -   id: check-toml
75      name: check toml
76      description: checks toml files for parseable syntax.
77      entry: check-toml
78      language: python
79      types: [toml]
92  -   id: check-yaml
93      name: check yaml
94      description: checks yaml files for parseable syntax.
95      entry: check-yaml
96      language: python
97      types: [yaml]
128 -   id: end-of-file-fixer
129     name: fix end of files
130     description: ensures that a file is either empty, or ends with one newline.
131     entry: end-of-file-fixer
132     language: python
133     types: [text]
134     stages: [commit, push, manual]
195 -   id: trailing-whitespace
196     name: trim trailing whitespace
197     description: trims trailing whitespace.
198     entry: trailing-whitespace-fixer
199     language: python
200     types: [text]
201     stages: [commit, push, manual]
```

PRE-COMMIT-HOOKS

Install the git hook scripts

```
(.venv) > pre-commit install
pre-commit installed at .git/hooks/pre-commit
```



To update your hooks to the latest version automatically

```
(.venv) > pre-commit autoupdate
[https://github.com/psf/black] already up to date!
[https://github.com/PyCQA/isort] already up to date!
[https://github.com/pycqa/pydocstyle] already up to date!
[https://github.com/charliermarsh/ruff-pre-commit] already up to date!
[https://github.com/sqlfluff/sqlfluff] updating 2.1.0 -> 2.1.1
[https://github.com/pre-commit/pre-commit-hooks] already up to date!
```

This will bring the hooks to the latest tag on the default branch

PRE-COMMIT-HOOKS

Now **pre-commit** will run automatically on **git commit**

Hook ids

```
(.venv) > git commit -m "Update hooks and deps"
black.....(no files to check) Skipped
black-jupyter.....(no files to check) Skipped
isort.....(no files to check) Skipped
pydocstyle.....(no files to check) Skipped
ruff.....(no files to check) Skipped
sqlfluff-fix.....(no files to check) Skipped
sqlfluff-lint.....(no files to check) Skipped
check python ast.....(no files to check) Skipped
fix end of files.....Failed
- hook id: end-of-file-fixer
- exit code: 1
- files were modified by this hook

Fixing .pre-commit-config.yaml

trim trailing whitespace.....Passed
check yaml.....Passed
check toml.....(no files to check) Skipped
check json.....(no files to check) Skipped
```

breathe

how?

PIP-TOOLS

JAZZ BAND

REPRODUCIBILITY

how?

PRE-COMMIT-HOOKS

COMPLIANCE

how?

GNU MAKE

CONSISTENCY

GNU MAKE

"GNU Make is a tool which enables the end user to build and install your package without knowing the details of how that is done. Make gets its knowledge from a file called the **makefile**.

Make is not limited to building a package. You can also use Make for **anything else you want to do often enough** to make it worth while writing down how to do it."


```
> choco install make
```

MAKEFILE

Makefile

```
.EXPORT_ALL_VARIABLES:
.PHONY: venv install update upgrade pre-commit check clean

GLOBAL_PYTHON = $(shell py -3.9 -c 'import sys; print(sys.executable)')
LOCAL_PYTHON = .venv\\Scripts\\python.exe
LOCAL_PIP_COMPILE = .venv\\Scripts\\pip-compile.exe
LOCAL_PIP_SYNC = .venv\\Scripts\\pip-sync.exe

setup: venv install pre-commit

## Create an empty environment
venv: $(GLOBAL_PYTHON)
    @echo "Creating .venv..."
    - deactivate
    ${GLOBAL_PYTHON} -m venv .venv

## Install dependencies
install: ${LOCAL_PYTHON}
    @echo "Installing dependencies..."
    ${LOCAL_PYTHON} -m pip install --upgrade pip
    ${LOCAL_PYTHON} -m pip install pip-tools
    ${LOCAL_PIP_COMPILE} pyproject.toml --output-file requirements.txt --resolver=backtr
    ${LOCAL_PIP_COMPILE} pyproject.toml --output-file requirements-dev.txt --resolver=ba
    ${LOCAL_PIP_SYNC} requirements-dev.txt --pip-args "--no-cache-dir"

## Update dependencies
update: ${LOCAL_PYTHON} ${LOCAL_PIP_SYNC}
    @echo "Updating dependencies..."
    ${LOCAL_PYTHON} -m pip install --upgrade pip
```

MAKE RULES

"A **rule** in **makefile** tells Make how to execute a series of **commands** … It also specifies a list of **dependencies** … This list should include all files which are used as inputs to the commands in the rule

```
target: dependencies
commands
```

Rule

Run

```
make target
```

MAKEFILE

Makefile M

Makefile

```
1  .EXPORT_ALL_VARIABLES:
2  .PHONY: venv install update upgrade pre-commit check clean
3
4  GLOBAL_PYTHON = $(shell py -3.9 -c 'import sys; print(sys.executable)')
5  LOCAL_PYTHON = .venv\\Scripts\\python.exe
6  LOCAL_PIP_COMPILE = .venv\\Scripts\\pip-compile.exe
7  LOCAL_PIP_SYNC = .venv\\Scripts\\pip-sync.exe
8
9  ## Create an empty environment
10 venv: $(GLOBAL_PYTHON)
11     @echo "Creating .venv..."
12     - deactivate
13     ${GLOBAL_PYTHON} -m venv .venv
14
15 ## Install dependencies
16 install: ${LOCAL_PYTHON}
17     @echo "Installing dependencies..."
18     ${LOCAL_PYTHON} -m pip install --upgrade pip
19     ${LOCAL_PYTHON} -m pip install pip-tools
20     ${LOCAL_PIP_COMPILE} pyproject.toml --output-file requirements.txt --resolver=backtr
21     ${LOCAL_PIP_COMPILE} pyproject.toml --output-file requirements-dev.txt --resolver=ba
22     ${LOCAL_PIP_SYNC} requirements-dev.txt --pip-args "--no-cache-dir"
23
```

MAKE RULES

Create a virtual environment

```
make venv
"Creating .venv..."
deactivate
```

Makefile

```
.EXPORT_ALL_VARIABLES:
.PHONY: venv install sync upgrade pre-commit check clean

GLOBAL_PYTHON = $(shell py -3.9 -c 'import sys; print(sys.executable)')
LOCAL_PYTHON = .venv\\Scripts\\python.exe
LOCAL_PIP_COMPILE = .venv\\Scripts\\pip-compile.exe
LOCAL_PIP_SYNC = .venv\\Scripts\\pip-sync.exe

## Create an empty environment
venv: $(GLOBAL_PYTHON)
    @echo "Creating .venv..."
    - deactivate
    ${GLOBAL_PYTHON} -m venv .venv
```

MAKE RULES

Install dependencies

```
make install
"Installing dependencies..."
"Syncing dependencies..."
```

Sync dependencies

```
make sync
"Syncing dependencies..."
```

Upgrade dependencies

```
make upgrade
"Upgrading dependencies..."
```

Makefile M

Makefile

```
## Install & sync dependencies
install: ${LOCAL_PYTHON}
    @echo "Installing dependencies..."
    ${LOCAL_PYTHON} -m pip install --upgrade pip
    ${LOCAL_PYTHON} -m pip install pip-tools
    ${LOCAL_PIP_COMPILE} pyproject.toml --output-file requirements.txt --resolver=backtracking --allow-unsafe
    ${LOCAL_PIP_COMPILE} pyproject.toml --output-file requirements-dev.txt --resolver=backtracking --allow-unsafe --extra dev
    @echo "Syncing dependencies..."
    ${LOCAL_PIP_SYNC} requirements-dev.txt --pip-args "--no-cache-dir"

## Sync dependencies
sync: ${LOCAL_PYTHON} ${LOCAL_PIP_SYNC}
    @echo "Syncing dependencies..."
    ${LOCAL_PYTHON} -m pip install --upgrade pip
    ${LOCAL_PIP_SYNC} requirements-dev.txt --pip-args "--no-cache-dir"

## Upgrade dependencies
upgrade: ${LOCAL_PYTHON} ${LOCAL_PIP_COMPILE}
    @echo "Upgrading dependencies..."
    ${LOCAL_PYTHON} -m pip install --upgrade pip
    ${LOCAL_PIP_COMPILE} --upgrade --output-file requirements.txt --resolver=backtracking --allow-unsafe
```

MAKE RULES

Install pre-commit hooks

```
make pre-commit
"Setting up pre-commit..."
```

Setup

```
make setup
"Creating .venv..."
deactivate
"Installing dependencies..."
"Setting up pre-commit..."
```

Makefile M

Makefile

```
36  ## Install pre-commit hooks
37  pre-commit:
38      @echo "Setting up pre-commit..."
39      pre-commit install
40      pre-commit autoupdate
41
42  setup: venv install pre-commit
```

MAKE RULES

Run helper packages

```
make check
"Running checks..."
ruff check --fix .
isort .
Skipped 2 files
black .
All done! ✨ 🍰 ✨
7 files left unchanged.
pydocstyle .
sqlfluff fix .
==== finding fixable violations ====
==== no fixable linting violations found ====
All Finished 🏁!
sqlfluff lint .
All Finished 🏁!
```

Makefile M

Makefile

```
44  ## Running checks
45  check: ${LOCAL_PYTHON}
46      @echo "Running checks..."
47      ruff check --fix .
48      isort .
49      black .
50      pydocstyle .
51      sqlfluff fix .
52      sqlfluff lint .
53
54  ## clean all temporary files
55  clean:
56      if exist .git\\hooks ( rmdir .git\\hooks /q /s )
57      - deactivate
58      if exist .venv\\ ( rmdir .venv /q /s )
```

Clean

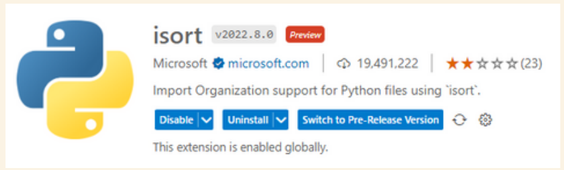
```
make clean
```

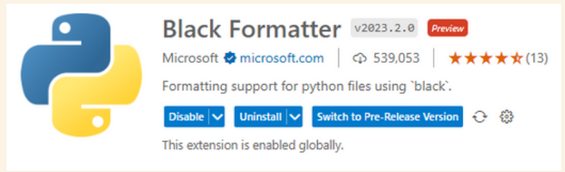
breathe

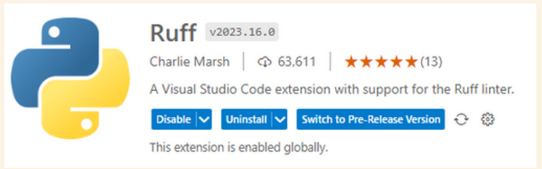
BONUS

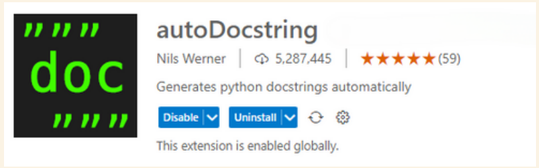
VSCODE INTEGRATION

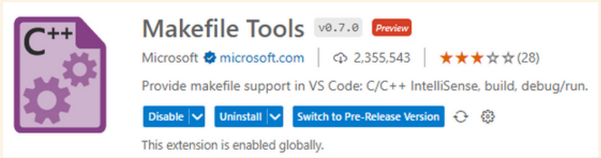
VSCODE INTEGRATION EXTENSIONS

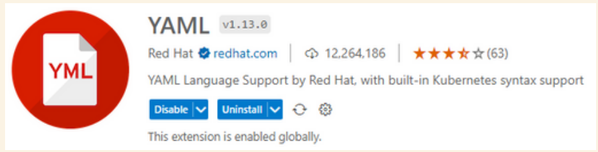


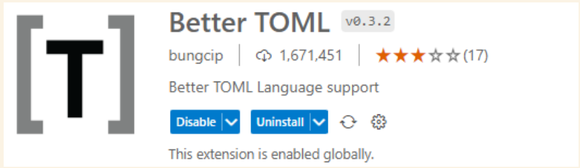








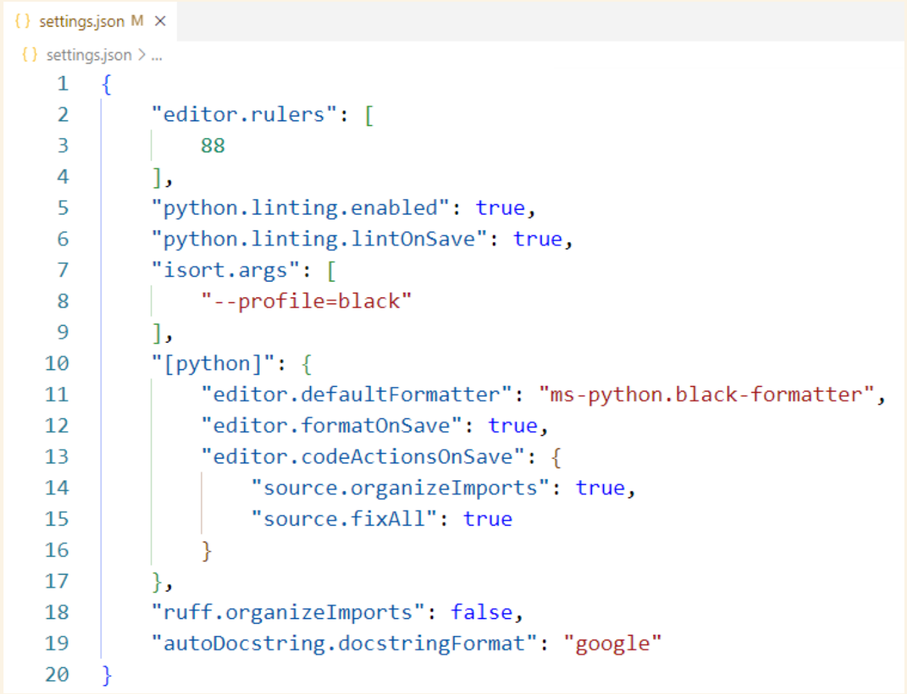




VSCODE INTEGRATION SETTINGS

CTRL + SHIFT + P

Preferences: Open User Settings (JSON)



Thank you!