# Contributing to Snorkel

We love contributors, so first and foremost, thank you!

We're actively working on our contributing guidelines, so this document is subject to change.

First things first: we adhere to the

[Contributor Covenant Code of Conduct](http://contributor-covenant.org/version/1/4/),

and expect all of our contributors to adhere to it as well.

## Development environment

### Installing

Snorkel uses [tox](https://tox.readthedocs.io) to manage development environments.

To get started, [install tox](https://tox.readthedocs.io/en/latest/install.html),

clone Snorkel, then use `tox` to create a development environment:

```bash

git clone https://github.com/snorkel-team/snorkel

pip3 install -U tox

cd snorkel

tox --devenv .env

```

Running `tox --devenv .env` will install create a virtual environment with Snorkel

and all of its dependencies installed in the directory `.env`.

This can be used in a number of ways, e.g. with `source .env/bin/activate`

or for [linting in VSCode](https://code.visualstudio.com/docs/python/environments#\_where-the-extension-looks-for-environments).

For example, you can simply activate this environment and start using Snorkel:

```bash

source .env/bin/activate

python3 -c "import snorkel.labeling; print(dir(snorkel.labeling))"

```

### Testing and committing

There are a number of useful tox commands defined:

```bash

tox -e py36 # Run unit tests pytest in Python 3.6

tox -e py37 # Run unit tests pytest in Python 3.7

tox -e coverage # Compute unit test coverage

tox -e spark # Run Spark-based tests (marked with @pytest.mark.spark)

tox -e complex # Run more complex, integration tests (marked with @pytest.mark.complex)

tox -e doctest # Run doctest on modules

tox -e check # Check style/linting with black, isort, and flake8

tox -e type # Run static type checking with mypy

tox -e fix # Fix style issues with black and isort

tox -e doc # Build documentation with Sphinx

tox # Run unit tests, doctests, style checks, linting, and type checking

```

Make sure to run `tox` before committing.

CI won't pass without `tox` succeeding.

As noted, we use a few additional tools that help to ensure that any commits or pull requests you submit conform with our established standards.

We use the following packages:

\* [isort](https://github.com/timothycrosley/isort): import standardization

\* [black](https://black.readthedocs.io/en/stable/): automatic code formatting

\* [flake8](http://flake8.pycqa.org/en/latest/): PEP8 linting

\* [mypy](http://mypy-lang.org/): static type checking

\* [pydocstyle](http://www.pydocstyle.org/): docstring compliance

\* [doctest-plus](https://github.com/astropy/pytest-doctestplus): check docstring code examples

The Snorkel maintainers are big fans of [VSCode](https://code.visualstudio.com/)'s Python tooling.

Here's a `settings.json` that takes advantage of the packages above (except isort) with in-line linting:

```json

{

"python.jediEnabled": true,

"python.formatting.provider": "black",

"python.linting.flake8Enabled": true,

"python.linting.mypyEnabled": true,

"python.linting.pydocstyleEnabled": true,

"python.linting.pylintEnabled": false,

}

```

### Docstrings

Snorkel ? documentation.

We expect all PRs to add or update API documentation for any affected pieces of code.

We use [NumPy style docstrings](https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example\_numpy.html), and enforce style compliance with pydocstyle as indicated above.

Docstrings can be cumbersome to write, so we encourage people to use tooling to speed up the process.

For VSCode, we like [autoDocstring](https://marketplace.visualstudio.com/items?itemName=njpwerner.autodocstring).

Just install the extension and add the following configuration to the `settings.json` example above.

Note that we use PEP 484 type hints, so parameter types should be removed from the docstring (although note that return types should still be included).

```json

{

"autoDocstring.docstringFormat": "numpy",

"autoDocstring.guessTypes": false

}

```

There are some standards we follow that our tooling doesn't automatically check/initialize:

\* Examples, examples, examples.

We love examples in docstrings; it's often the best form of documentation.

The `Example` or `Examples` section should come after `Parameters` but before `Attributes`.

Running `tox -e doctest` will test your docstring examples.

\* Make sure to add `Attributes` sections to docstrings to document public attributes of

classes.

The `Attributes` section should be the last part of the docstring.

\* No need to document private methods or attributes.

### Complex/integration/long-running tests

Any test that runs longer than half a second should be marked with the

`@pytest.mark.complex` decorator.

Typically, these will be integration tests or tests that verify complex

properties like model convergence.

We exclude long-running tests from the default `tox` and Travis builds

on non-master and non-release branches to keep things moving fast.

If you're touching areas of the code that could break a long-running test,

you should include the results of `tox -e complex` in the PR's test plan.

To see the durations of the 10 longest-running tests, run

`tox -e py3 -- -m 'not complex and not spark' --durations 10`.

### PySpark tests

PySpark tests are invoked separately from the rest since they require

installing Java and the large PySpark package.

They are executed on Travis, but not by default for a local `tox` command.

If you're making changes to Spark-based operators, make sure you have

Java 8 installed locally and then run `tox -e spark`.

If you add a test that imports PySpark mark it with the

`@pytest.mark.spark` decorator.

Add the `@pytest.mark.complex` decorator as well if it runs a Spark

action (e.g. `.collect()`).

## PRs

### Submitting PRs

When submitting a PR, make sure to use the preformatted template.

Except in special cases, all PRs should be against `master`.

Avoid using "staging branches" as much as possible.

If you want to add complicated features, please

[stack your PRs](https://graysonkoonce.com/stacked-pull-requests-keeping-github-diffs-small/)

to ensure an effective review process.

It's unlikely that we'll approve any

[single PR over 500 lines](https://www.ibm.com/developerworks/rational/library/11-proven-practices-for-peer-review/index.html).

### Requesting reviews

Direct commits to master are blocked, and PRs require an approving review

to merge into master.

By convention, the Snorkel maintainers will review PRs when:

\* An initial review has been requested

\* A maintainer is tagged in the PR comments and asked to complete a review

We ask that you make sure initial CI checks are passing before requesting a review.

### Merging

The PR author owns the test plan and has final say on correctness.

Therefore, it is up to the PR author to give the final okay on merging

(or merge their PR if they have write access).