<!--

Licensed to the Apache Software Foundation (ASF) under one

or more contributor license agreements. See the NOTICE file

distributed with this work for additional information

regarding copyright ownership. The ASF licenses this file

to you under the Apache License, Version 2.0 (the

"License"); you may not use this file except in compliance

with the License. You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing,

software distributed under the License is distributed on an

"AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY

KIND, either express or implied. See the License for the

specific language governing permissions and limitations

under the License.

-->

# Contributing

Contributions are welcome and are greatly appreciated! Every

little bit helps, and credit will always be given.

## Table of Contents

- [Contributing](#contributing)

- [Table of Contents](#table-of-contents)

- [Orientation](#orientation)

- [Types of Contributions](#types-of-contributions)

- [Report Bug](#report-bug)

- [Submit Ideas or Feature Requests](#submit-ideas-or-feature-requests)

- [Fix Bugs](#fix-bugs)

- [Implement Features](#implement-features)

- [Improve Documentation](#improve-documentation)

- [Add Translations](#add-translations)

- [Ask Questions](#ask-questions)

- [Pull Request Guidelines](#pull-request-guidelines)

- [Protocol](#protocol)

- [Authoring](#authoring)

- [Reviewing](#reviewing)

- [Merging](#merging)

- [Post-merge Responsibility](#post-merge-responsibility)

- [Managing Issues and PRs](#managing-issues-and-prs)

- [Revert Guidelines](#revert-guidelines)

- [Setup Local Environment for Development](#setup-local-environment-for-development)

- [Documentation](#documentation)

- [Images](#images)

- [API documentation](#api-documentation)

- [Flask server](#flask-server)

- [OS Dependencies](#os-dependencies)

- [Logging to the browser console](#logging-to-the-browser-console)

- [Frontend Assets](#frontend-assets)

- [nvm and node](#nvm-and-node)

- [Prerequisite](#prerequisite)

- [Installing Dependencies](#installing-dependencies)

- [Building](#building)

- [Docker (docker-compose)](#docker-docker-compose)

- [Updating NPM packages](#updating-npm-packages)

- [Feature flags](#feature-flags)

- [Git Hooks](#git-hooks)

- [Linting](#linting)

- [Conventions](#conventions)

- [Python](#python)

- [Typing](#typing)

- [Python](#python-1)

- [TypeScript](#typescript)

- [Testing](#testing)

- [Python Testing](#python-testing)

- [Frontend Testing](#frontend-testing)

- [Integration Testing](#integration-testing)

- [Translating](#translating)

- [Enabling language selection](#enabling-language-selection)

- [Extracting new strings for translation](#extracting-new-strings-for-translation)

- [Creating a new language dictionary](#creating-a-new-language-dictionary)

- [Tips](#tips)

- [Adding a new datasource](#adding-a-new-datasource)

- [Improving visualizations](#improving-visualizations)

- [Adding a DB migration](#adding-a-db-migration)

- [Merging DB migrations](#merging-db-migrations)

- [SQL Lab Async](#sql-lab-async)

- [Chart Parameters](#chart-parameters)

- [Datasource & Chart Type](#datasource--chart-type)

- [Time](#time)

- [GROUP BY](#group-by)

- [NOT GROUPED BY](#not-grouped-by)

- [Y Axis 1](#y-axis-1)

- [Y Axis 2](#y-axis-2)

- [Query](#query)

- [Filters Configuration](#filters-configuration)

- [Chart Options](#chart-options)

- [Y Axis](#y-axis)

- [Other](#other)

- [Unclassified](#unclassified)

## Orientation

Here's a list of repositories that contain Superset-related packages:

- [apache/incubator-superset](https://github.com/apache/incubator-superset)

is the main repository containing the `apache-superset` Python package

distributed on

[pypi](https://pypi.org/project/apache-superset/). This repository

also includes Superset's main TypeScript/JavaScript bundles and react apps under

the [superset-frontend](https://github.com/apache/incubator-superset/tree/master/superset-frontend)

folder.

- [apache-superset/superset-ui](https://github.com/apache-superset/superset-ui)

contains core Superset's

[npm packages](https://github.com/apache-superset/superset-ui/tree/master/packages).

These packages are shared across the React apps in the main repository,

and in visualization plugins.

- [apache-superset/superset-ui-plugins](https://github.com/apache-superset/superset-ui-plugins)

contains the code for the default visualizations that ship with Superset

and are maintained by the core community.

- [apache-superset/superset-ui-plugins-deckgl](https://github.com/apache-superset/superset-ui-plugins-deckgl)

contains the code for the geospatial visualizations that ship with Superset

and are maintained by the core community.

- [github.com/apache-superset](https://github.com/apache-superset) is the

Github organization under which we manage Superset-related

small tools, forks and Superset-related experimental ideas.

## Types of Contributions

### Report Bug

The best way to report a bug is to file an issue on GitHub. Please include:

- Your operating system name and version.

- Superset version.

- Detailed steps to reproduce the bug.

- Any details about your local setup that might be helpful in troubleshooting.

When posting Python stack traces, please quote them using

[Markdown blocks](https://help.github.com/articles/creating-and-highlighting-code-blocks/).

### Submit Ideas or Feature Requests

The best way is to file an issue on GitHub:

- Explain in detail how it would work.

- Keep the scope as narrow as possible, to make it easier to implement.

- Remember that this is a volunteer-driven project, and that contributions are welcome :)

For large features or major changes to codebase, please create \*\*Superset Improvement Proposal (SIP)\*\*. See template from [SIP-0](https://github.com/apache/incubator-superset/issues/5602)

### Fix Bugs

Look through the GitHub issues. Issues tagged with `#bug` are

open to whoever wants to implement them.

### Implement Features

Look through the GitHub issues. Issues tagged with

`#feature` is open to whoever wants to implement it.

### Improve Documentation

Superset could always use better documentation,

whether as part of the official Superset docs,

in docstrings, `docs/\*.rst` or even on the web as blog posts or

articles. See [Documentation](#documentation) for more details.

### Add Translations

If you are proficient in a non-English language, you can help translate

text strings from Superset's UI. You can jump in to the existing

language dictionaries at

`superset/translations/<language\_code>/LC\_MESSAGES/messages.po`, or

even create a dictionary for a new language altogether.

See [Translating](#translating) for more details.

### Ask Questions

There is a dedicated [`apache-superset` tag](https://stackoverflow.com/questions/tagged/apache-superset) on [StackOverflow](https://stackoverflow.com/). Please use it when asking questions.

## Pull Request Guidelines

A philosophy we would like to strongly encourage is

> Before creating a PR, create an issue.

The purpose is to separate problem from possible solutions.

\*\*Bug fixes:\*\* If you’re only fixing a small bug, it’s fine to submit a pull request right away but we highly recommend to file an issue detailing what you’re fixing. This is helpful in case we don’t accept that specific fix but want to keep track of the issue. Please keep in mind that the project maintainers reserve the rights to accept or reject incoming PRs, so it is better to separate the issue and the code to fix it from each other. In some cases, project maintainers may request you to create a separate issue from PR before proceeding.

\*\*Refactor:\*\* For small refactors, it can be a standalone PR itself detailing what you are refactoring and why. If there are concerns, project maintainers may request you to create a `#SIP` for the PR before proceeding.

\*\*Feature/Large changes:\*\* If you intend to change the public API, or make any non-trivial changes to the implementation, we requires you to file a new issue as `#SIP` (Superset Improvement Proposal). This lets us reach an agreement on your proposal before you put significant effort into it. You are welcome to submit a PR along with the SIP (sometimes necessary for demonstration), but we will not review/merge the code until the SIP is approved.

In general, small PRs are always easier to review than large PRs. The best practice is to break your work into smaller independent PRs and refer to the same issue. This will greatly reduce turnaround time.

Finally, never submit a PR that will put master branch in broken state. If the PR is part of multiple PRs to complete a large feature and cannot work on its own, you can create a feature branch and merge all related PRs into the feature branch before creating a PR from feature branch to master.

### Protocol

#### Authoring

- Fill in all sections of the PR template.

- Add prefix `[WIP]` to title if not ready for review (WIP = work-in-progress). We recommend creating a PR with `[WIP]` first and remove it once you have passed CI test and read through your code changes at least once.

- \*\*Screenshots/GIFs:\*\* Changes to user interface require before/after screenshots, or GIF for interactions

- Recommended capture tools ([Kap](https://getkap.co/), [LICEcap](https://www.cockos.com/licecap/), [Skitch](https://download.cnet.com/Skitch/3000-13455\_4-189876.html))

- If no screenshot is provided, the committers will mark the PR with `need:screenshot` label and will not review until screenshot is provided.

- \*\*Dependencies:\*\* Be careful about adding new dependency and avoid unnecessary dependencies.

- For Python, include it in `setup.py` denoting any specific restrictions and in `requirements.txt` pinned to a specific version which ensures that the application build is deterministic.

- For TypeScript/JavaScript, include new libraries in `package.json`

- \*\*Tests:\*\* The pull request should include tests, either as doctests, unit tests, or both. Make sure to resolve all errors and test failures. See [Testing](#testing) for how to run tests.

- \*\*Documentation:\*\* If the pull request adds functionality, the docs should be updated as part of the same PR. Doc string are often sufficient, make sure to follow the sphinx compatible standards.

- \*\*CI:\*\* Reviewers will not review the code until all CI tests are passed. Sometimes there can be flaky tests. You can close and open PR to re-run CI test. Please report if the issue persists. After the CI fix has been deployed to `master`, please rebase your PR.

- \*\*Code coverage:\*\* Please ensure that code coverage does not decrease.

- Remove `[WIP]` when ready for review. Please note that it may be merged soon after approved so please make sure the PR is ready to merge and do not expect more time for post-approval edits.

- If the PR was not ready for review and inactive for > 30 days, we will close it due to inactivity. The author is welcome to re-open and update.

#### Reviewing

- Use constructive tone when writing reviews.

- If there are changes required, state clearly what needs to be done before the PR can be approved.

- If you are asked to update your pull request with some changes there's no need to create a new one. Push your changes to the same branch.

- The committers reserve the right to reject any PR and in some cases may request the author to file an issue.

#### Merging

- At least one approval is required for merging a PR.

- PR is usually left open for at least 24 hours before merging.

- After the PR is merged, [close the corresponding issue(s)](https://help.github.com/articles/closing-issues-using-keywords/).

#### Post-merge Responsibility

- Project maintainers may contact the PR author if new issues are introduced by the PR.

- Project maintainers may revert your changes if a critical issue is found, such as breaking master branch CI.

## Managing Issues and PRs

To handle issues and PRs that are coming in, committers read issues/PRs and flag them with labels to categorize and help contributors spot where to take actions, as contributors usually have different expertises.

Triaging goals

- \*\*For issues:\*\* Categorize, screen issues, flag required actions from authors.

- \*\*For PRs:\*\* Categorize, flag required actions from authors. If PR is ready for review, flag required actions from reviewers.

First, add \*\*Category labels (a.k.a. hash labels)\*\*. Every issue/PR must have one hash label (except spam entry). Labels that begin with `#` defines issue/PR type:

| Label | for Issue | for PR |

| --------------- | --------------------------------------------------------------------------------------------------------------------------------------- | ------------------------------------------------------------------------------------------------------------------------------------------------- |

| `#bug` | Bug report | Bug fix |

| `#code-quality` | Describe problem with code, architecture or productivity | Refactor, tests, tooling |

| `#feature` | New feature request | New feature implementation |

| `#refine` | Propose improvement that does not provide new features and is also not a bug fix nor refactor, such as adjust padding, refine UI style. | Implementation of improvement that does not provide new features and is also not a bug fix nor refactor, such as adjust padding, refine UI style. |

| `#doc` | Documentation | Documentation |

| `#question` | Troubleshooting: Installation, Running locally, Ask how to do something. Can be changed to `#bug` later. | N/A |

| `#SIP` | Superset Improvement Proposal | N/A |

| `#ASF` | Tasks related to Apache Software Foundation policy | Tasks related to Apache Software Foundation policy |

Then add other types of labels as appropriate.

- \*\*Descriptive labels (a.k.a. dot labels):\*\* These labels that begin with `.` describe the details of the issue/PR, such as `.ui`, `.js`, `.install`, `.backend`, etc. Each issue/PR can have zero or more dot labels.

- \*\*Need labels:\*\* These labels have pattern `need:xxx`, which describe the work required to progress, such as `need:rebase`, `need:update`, `need:screenshot`.

- \*\*Risk labels:\*\* These labels have pattern `risk:xxx`, which describe the potential risk on adopting the work, such as `risk:db-migration`. The intention was to better understand the impact and create awareness for PRs that need more rigorous testing.

- \*\*Status labels:\*\* These labels describe the status (`abandoned`, `wontfix`, `cant-reproduce`, etc.) Issue/PRs that are rejected or closed without completion should have one or more status labels.

- \*\*Version labels:\*\* These have the pattern `vx.x` such as `v0.28`. Version labels on issues describe the version the bug was reported on. Version labels on PR describe the first release that will include the PR.

Committers may also update title to reflect the issue/PR content if the author-provided title is not descriptive enough.

If the PR passes CI tests and does not have any `need:` labels, it is ready for review, add label `review` and/or `design-review`.

If an issue/PR has been inactive for >=30 days, it will be closed. If it does not have any status label, add `inactive`.

## Revert Guidelines

Reverting changes that are causing issues in the master branch is a normal and expected part of the development process. In an open source community, the ramifications of a change cannot always be fully understood. With that in mind, here are some considerations to keep in mind when considering a revert:

- \*\*Availability of the PR author:\*\* If the original PR author or the engineer who merged the code is highly available and can provide a fix in a reasonable timeframe, this would counter-indicate reverting.

- \*\*Severity of the issue:\*\* How severe is the problem on master? Is it keeping the project from moving forward? Is there user impact? What percentage of users will experience a problem?

- \*\*Size of the change being reverted:\*\* Reverting a single small PR is a much lower-risk proposition than reverting a massive, multi-PR change.

- \*\*Age of the change being reverted:\*\* Reverting a recently-merged PR will be more acceptable than reverting an older PR. A bug discovered in an older PR is unlikely to be causing widespread serious issues.

- \*\*Risk inherent in reverting:\*\* Will the reversion break critical functionality? Is the medicine more dangerous than the disease?

- \*\*Difficulty of crafting a fix:\*\* In the case of issues with a clear solution, it may be preferable to implement and merge a fix rather than a revert.

Should you decide that reverting is desirable, it is the responsibility of the Contributor performing the revert to:

- \*\*Contact the interested parties:\*\* The PR's author and the engineer who merged the work should both be contacted and informed of the revert.

- \*\*Provide concise reproduction steps:\*\* Ensure that the issue can be clearly understood and duplicated by the original author of the PR.

- \*\*Put the revert through code review:\*\* The revert must be approved by another committer.

## Setup Local Environment for Development

First, [fork the repository on GitHub](https://help.github.com/articles/about-forks/), then clone it. You can clone the main repository directly, but you won't be able to send pull requests.

```bash

git clone git@github.com:your-username/incubator-superset.git

cd incubator-superset

```

### Documentation

The latest documentation and tutorial are available at https://superset.incubator.apache.org/.

Contributing to the official documentation is relatively easy, once you've setup

your environment and done an edit end-to-end. The docs can be found in the

`docs/` subdirectory of the repository, and are written in the

[reStructuredText format](https://en.wikipedia.org/wiki/ReStructuredText) (.rst).

If you've written Markdown before, you'll find the reStructuredText format familiar.

Superset uses [Sphinx](http://www.sphinx-doc.org/en/1.5.1/) to convert the rst files

in `docs/` to the final HTML output users see.

Finally, to make changes to the rst files and build the docs using Sphinx,

you'll need to install a handful of dependencies from the repo you cloned:

```bash

pip install -r requirements/documentation.txt

```

To get the feel for how to edit and build the docs, let's edit a file, build

the docs and see our changes in action. First, you'll want to

[create a new branch](https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging)

to work on your changes:

```bash

git checkout -b changes-to-docs

```

Now, go ahead and edit one of the files under `docs/`, say `docs/tutorial.rst` - change

it however you want. Check out the

[ReStructuredText Primer](http://docutils.sourceforge.net/docs/user/rst/quickstart.html)

for a reference on the formatting of the rst files.

Once you've made your changes, run this command to convert the docs into HTML:

```bash

make html

```

You'll see a lot of output as Sphinx handles the conversion. After it's done, the

HTML Sphinx generated should be in `docs/\_build/html`. Navigate there

and start a simple web server so we can check out the docs in a browser:

```bash

cd docs/\_build/html

python -m http.server # Python2 users should use SimpleHTTPServer

```

This will start a small Python web server listening on port 8000. Point your

browser to http://localhost:8000, find the file

you edited earlier, and check out your changes!

If you've made a change you'd like to contribute to the actual docs, just commit

your code, push your new branch to Github:

```bash

git add docs/tutorial.rst

git commit -m 'Awesome new change to tutorial'

git push origin changes-to-docs

```

Then, [open a pull request](https://help.github.com/articles/about-pull-requests/).

#### Images

If you're adding new images to the documentation, you'll notice that the images

referenced in the rst, e.g.

.. image:: \_static/images/tutorial/tutorial\_01\_sources\_database.png

aren't actually stored in that directory. Instead, you should add and commit

images (and any other static assets) to the `superset-frontend/images` directory.

When the docs are deployed to https://superset.incubator.apache.org/, images

are copied from there to the `\_static/images` directory, just like they're referenced

in the docs.

For example, the image referenced above actually lives in `superset-frontend/images/tutorial`. Since the image is moved during the documentation build process, the docs reference the image in `\_static/images/tutorial` instead.

#### API documentation

Generate the API documentation with:

```bash

pip install -r requirements/documentation.txt

python setup.py build\_sphinx

```

### Flask server

#### OS Dependencies

Make sure your machine meets the [OS dependencies](https://superset.incubator.apache.org/installation.html#os-dependencies) before following these steps.

Developers should use a virtualenv.

```bash

pip install virtualenv

```

Then proceed with:

```bash

# Create a virtual environemnt and activate it (recommended)

virtualenv -p python3 venv # setup a python3.6 virtualenv

source venv/bin/activate

# Install external dependencies

pip install -r requirements/local.txt

# Install Superset in editable (development) mode

pip install -e .

# Create an admin user in your metadata database

superset fab create-admin

# Initialize the database

superset db upgrade

# Create default roles and permissions

superset init

# Load some data to play with

superset load\_examples

# Start the Flask dev web server from inside your virtualenv.

# Note that your page may not have css at this point.

# See instructions below how to build the front-end assets.

FLASK\_ENV=development superset run -p 8088 --with-threads --reload --debugger

```

\*\*Note: the FLASK\_APP env var should not need to be set, as it's currently controlled

via `.flaskenv`, however if needed, it should be set to `superset.app:create\_app()`\*\*

If you have made changes to the FAB-managed templates, which are not built the same way as the newer, React-powered front-end assets, you need to start the app without the `--with-threads` argument like so:

`FLASK\_ENV=development superset run -p 8088 --reload --debugger`

#### Logging to the browser console

This feature is only available on Python 3. When debugging your application, you can have the server logs sent directly to the browser console using the [ConsoleLog](https://github.com/betodealmeida/consolelog) package. You need to mutate the app, by adding the following to your `config.py` or `superset\_config.py`:

```python

from console\_log import ConsoleLog

def FLASK\_APP\_MUTATOR(app):

app.wsgi\_app = ConsoleLog(app.wsgi\_app, app.logger)

```

Then make sure you run your WSGI server using the right worker type:

```bash

FLASK\_ENV=development gunicorn "superset.app:create\_app()" -k "geventwebsocket.gunicorn.workers.GeventWebSocketWorker" -b 127.0.0.1:8088 --reload

```

You can log anything to the browser console, including objects:

```python

from superset import app

app.logger.error('An exception occurred!')

app.logger.info(form\_data)

```

### Frontend Assets

Frontend assets (TypeScript, JavaScript, CSS, and images) must be compiled in order to properly display the web UI. The `superset-frontend` directory contains all NPM-managed front end assets. Note that there are additional frontend assets bundled with Flask-Appbuilder (e.g. jQuery and bootstrap); these are not managed by NPM, and may be phased out in the future.

#### nvm and node

First, be sure you are using recent versions of NodeJS and npm. Using [nvm](https://github.com/creationix/nvm) to manage them is recommended. Check the docs at the link to be sure, but at the time of writing the following would install nvm and node:

```bash

curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.34.0/install.sh | bash

nvm install node

```

#### Prerequisite

#### Installing Dependencies

Install third-party dependencies listed in `package.json`:

```bash

# From the root of the repository

cd superset-frontend

# Install dependencies from `package-lock.json`

npm ci

```

#### Building

You can run the Webpack dev server (in a separate terminal from Flask), which runs on port 9000 and proxies non-asset requests to the Flask server on port 8088. After pointing your browser to `http://localhost:9000`, updates to asset sources will be reflected in-browser without a refresh.

```bash

# Run the dev server

npm run dev-server

# Run the dev server on a non-default port

npm run dev-server -- --devserverPort=9001

# Run the dev server proxying to a Flask server on a non-default port

npm run dev-server -- --supersetPort=8081

# Or proxy it to a remote backend so you can test frontend changes without

# starting the backend locally

npm run dev-server -- --superset=https://superset-dev.example.com

```

Alternatively you can use one of the following commands.

```bash

# Start a watcher that recompiles your assets as you modify them (but have to manually reload your browser to see changes.)

npm run dev

# Compile the TypeScript/JavaScript and CSS in production/optimized mode for official releases

npm run prod

```

If you run this service from somewhere other than your local machine, you may need to add hostname value to webpack.config.js at .devServer.public specifying the endpoint at which you will access the app. For example: myhost:9001. For convenience you may want to install webpack, webpack-cli and webpack-dev-server globally so that you can run them directly:

```bash

npm install --global webpack webpack-cli webpack-dev-server

```

#### Docker (docker-compose)

See docs [here](docker/README.md)

#### Updating NPM packages

Use npm in the prescribed way, making sure that

`superset-frontend/package-lock.json` is updated according to `npm`-prescribed

best practices.

#### Feature flags

Superset supports a server-wide feature flag system, which eases the incremental development of features. To add a new feature flag, simply modify `superset\_config.py` with something like the following:

```python

FEATURE\_FLAGS = {

'SCOPED\_FILTER': True,

}

```

If you want to use the same flag in the client code, also add it to the FeatureFlag TypeScript enum in `superset-frontend/src/featureFlags.ts`. For example,

```typescript

export enum FeatureFlag {

SCOPED\_FILTER = 'SCOPED\_FILTER',

}

```

`superset/config.py` contains `DEFAULT\_FEATURE\_FLAGS` which will be overwritten by

those specified under FEATURE\_FLAGS in `superset\_config.py`. For example, `DEFAULT\_FEATURE\_FLAGS = { 'FOO': True, 'BAR': False }` in `superset/config.py` and `FEATURE\_FLAGS = { 'BAR': True, 'BAZ': True }` in `superset\_config.py` will result

in combined feature flags of `{ 'FOO': True, 'BAR': True, 'BAZ': True }`.

## Git Hooks

Superset uses Git pre-commit hooks courtesy of [pre-commit](https://pre-commit.com/). To install run the following:

```bash

pip3 install -r requirements/integration.txt

pre-commit install

```

## Linting

Lint the project with:

```bash

# for python

tox -e flake8

# for frontend

cd superset-frontend

npm ci

npm run lint

```

The Python code is auto-formatted using [Black](https://github.com/python/black) which

is configured as a pre-commit hook. There are also numerous [editor integrations](https://black.readthedocs.io/en/stable/editor\_integration.html).

## Conventions

### Python

Parameters in the `config.py` (which are accessible via the Flask app.config dictionary) are assummed to always be defined and thus should be accessed directly via,

```python

blueprints = app.config["BLUEPRINTS"]

```

rather than,

```python

blueprints = app.config.get("BLUEPRINTS")

```

or similar as the later will cause typing issues. The former is of type `List[Callable]` whereas the later is of type `Optional[List[Callable]]`.

## Typing

### Python

To ensure clarity, consistency, all readability, \_all\_ new functions should use

[type hints](https://docs.python.org/3/library/typing.html) and include a

docstring using Sphinx documentation.

Note per [PEP-484](https://www.python.org/dev/peps/pep-0484/#exceptions) no

syntax for listing explicitly raised exceptions is proposed and thus the

recommendation is to put this information in a docstring, i.e.,

```python

import math

from typing import Union

def sqrt(x: Union[float, int]) -> Union[float, int]:

"""

Return the square root of x.

:param x: A number

:returns: The square root of the given number

:raises ValueError: If the number is negative

"""

return math.sqrt(x)

```

### TypeScript

TypeScript is fully supported and is the recommended language for writing all new frontend components. When modifying existing functions/components, migrating to TypeScript is appreciated, but not required. Examples of migrating functions/components to TypeScript can be found in [#9162](https://github.com/apache/incubator-superset/pull/9162) and [#9180](https://github.com/apache/incubator-superset/pull/9180).

## Testing

### Python Testing

All python tests are carried out in [tox](https://tox.readthedocs.io/en/latest/index.html)

a standardized testing framework.

All python tests can be run with any of the tox [environments](https://tox.readthedocs.io/en/latest/example/basic.html#a-simple-tox-ini-default-environments), via,

```bash

tox -e <environment>

```

For example,

```bash

tox -e py36

```

Alternatively, you can run all tests in a single file via,

```bash

tox -e <environment> -- tests/test\_file.py

```

or for a specific test via,

```bash

tox -e <environment> -- tests/test\_file.py:TestClassName.test\_method\_name

```

Note that the test environment uses a temporary directory for defining the

SQLite databases which will be cleared each time before the group of test

commands are invoked.

### Frontend Testing

We use [Jest](https://jestjs.io/) and [Enzyme](https://airbnb.io/enzyme/) to test TypeScript/JavaScript. Tests can be run with:

```bash

cd superset-frontend

npm run test

```

### Integration Testing

We use [Cypress](https://www.cypress.io/) for integration tests. Tests can be run by `tox -e cypress`. To open Cypress and explore tests first setup and run test server:

```bash

export SUPERSET\_CONFIG=tests.superset\_test\_config

superset db upgrade

superset init

superset load\_test\_users

superset load\_examples

superset run --port 8081

```

Run Cypress tests:

```bash

cd superset-frontend

npm run build

cd cypress-base

npm install

npm run cypress run

# run tests from a specific file

npm run cypress run -- --spec cypress/integration/explore/link.test.js

# run specific file with video capture

npm run cypress run -- --spec cypress/integration/dashboard/index.test.js --config video=true

# to open the cypress ui

npm run cypress open

```

See [`superset-frontend/cypress\_build.sh`](https://github.com/apache/incubator-superset/blob/master/superset-frontend/cypress\_build.sh).

### Storybook

Superset includes a [Storybook](https://storybook.js.org/) to preview the layout/styling of various Superset components, and variations thereof. To open and view the Storybook:

```bash

cd superset-frontend

npm run storybool

```

When contributing new React components to Superset, please try to add a Story alongside the component's `jsx/tsx` file.

## Translating

We use [Babel](http://babel.pocoo.org/en/latest/) to translate Superset.

In Python files, we import the magic `\_` function using:

```python

from flask\_babel import lazy\_gettext as \_

```

then wrap our translatable strings with it, e.g. `\_('Translate me')`.

During extraction, string literals passed to `\_` will be added to the

generated `.po` file for each language for later translation.

At runtime, the `\_` function will return the translation of the given

string for the current language, or the given string itself

if no translation is available.

In TypeScript/JavaScript, the technique is similar:

we import `t` (simple translation), `tn` (translation containing a number).

```javascript

import { t, tn } from '@superset-ui/translation';

```

### Enabling language selection

Add the `LANGUAGES` variable to your `superset\_config.py`. Having more than one

option inside will add a language selection dropdown to the UI on the right side

of the navigation bar.

```python

LANGUAGES = {

'en': {'flag': 'us', 'name': 'English'},

'fr': {'flag': 'fr', 'name': 'French'},

'zh': {'flag': 'cn', 'name': 'Chinese'},

}

```

### Extracting new strings for translation

```bash

flask fab babel-extract --target superset/translations --output superset/translations/messages.pot --config superset/translations/babel.cfg -k \_ -k \_\_ -k t -k tn -k tct

```

You can then translate the strings gathered in files located under

`superset/translation`, where there's one per language. You can use [Poedit](https://poedit.net/features)

to translate the `po` file more conveniently.

There are some [tutorials in the wiki](https://wiki.lxde.org/en/Translate\_\*.po\_files\_with\_Poedit).

For the translations to take effect:

```bash

# In the case of JS translation, we need to convert the PO file into a JSON file, and we need the global download of the npm package po2json.

npm install -g po2json

flask fab babel-compile --target superset/translations

# Convert the en PO file into a JSON file

po2json -d superset -f jed1.x superset/translations/en/LC\_MESSAGES/messages.po superset/translations/en/LC\_MESSAGES/messages.json

```

If you get errors running `po2json`, you might be running the Ubuntu package with the same

name, rather than the NodeJS package (they have a different format for the arguments). If

there is a conflict, you may need to update your `PATH` environment variable or fully qualify

the executable path (e.g. `/usr/local/bin/po2json` instead of `po2json`).

If you get a lot of `[null,\*\*\*]` in `messages.json`, just delete all the `null,`.

For example, `"year":["?"]` is correct while `"year":[null,"?"]`is incorrect.

### Creating a new language dictionary

To create a dictionary for a new language, run the following, where `LANGUAGE\_CODE` is replaced with

the language code for your target language, e.g. `es` (see [Flask AppBuilder i18n documentation](https://flask-appbuilder.readthedocs.io/en/latest/i18n.html) for more details):

```bash

pip install -r superset/translations/requirements.txt

pybabel init -i superset/translations/messages.pot -d superset/translations -l LANGUAGE\_CODE

```

Then, [extract strings for the new language](#extracting-new-strings-for-translation).

## Tips

### Adding a new datasource

1. Create Models and Views for the datasource, add them under superset folder, like a new my\_models.py

with models for cluster, datasources, columns and metrics and my\_views.py with clustermodelview

and datasourcemodelview.

1. Create DB migration files for the new models

1. Specify this variable to add the datasource model and from which module it is from in config.py:

For example:

```python

ADDITIONAL\_MODULE\_DS\_MAP = {'superset.my\_models': ['MyDatasource', 'MyOtherDatasource']}

```

This means it'll register MyDatasource and MyOtherDatasource in superset.my\_models module in the source registry.

### Improving visualizations

To edit the frontend code for visualizations, you will have to check out a copy of [apache-superset/superset-ui](https://github.com/apache-superset/superset-ui):

```bash

git clone https://github.com/apache-superset/superset-ui.git

cd superset-ui

yarn

yarn build

```

Then use `npm link` to create symlinks of the plugins/superset-ui packages you want to edit in `superset-frontend/node\_modules`:

```bash

cd incubator-superset/superset-frontend

npm link ../../superset-ui/plugins/[PLUGIN NAME]

# Or to link all core superset-ui and plugin packages:

# npm link ../../superset-ui/{packages,plugins}/\*

# Start developing

npm run dev-server

```

When `superset-ui` packages are linked with `npm link`, the dev server will automatically load a package's source code from its `/src` directory, instead of the built modules in `lib/` or `esm/`.

Note that every time you do `npm install`, you will lose the symlink(s) and may have to run `npm link` again.

### Visualization Plugins

The topic of authoring new plugins, whether you'd like to contribute

it back or not has been well documented in the

[So, You Want to Build a Superset Viz Plugin...](https://preset.io/blog/2020-07-02-hello-world/) blog post

To contribute a plugin to Superset-UI, your plugin must meet the following criteria:

\* The plugin should be applicable to the community at large, not a particularly specialized use case

\* The plugin should be written with TypeScript

\* The plugin should contain sufficient unit/e2e tests

\* The plugin should use appropriate namespacing, e.g. a folder name of `plugin-chart-whatever` and a package name of `@superset-ui/plugin-chart-whatever`

\* The plugin should use them variables via Emotion, as passed in by the ThemeProvider

\* The plugin should provide adequate error handling (no data returned, malformatted data, invalid controls, etc.)

\* The plugin should contain documentation in the form of a populated `README.md` file

\* The plugin should have a meaningful and unique icon

\* Above all else, the plugin should come with a \*commitment to maintenance\* from the original author(s)

Submissions will be considered for submission (or removal) on a case-by-case basis.

### Adding a DB migration

1. Alter the model you want to change. This example will add a `Column` Annotations model.

[Example commit](https://github.com/apache/incubator-superset/commit/6c25f549384d7c2fc288451222e50493a7b14104)

1. Generate the migration file

```bash

superset db migrate -m 'add\_metadata\_column\_to\_annotation\_model.py'

```

This will generate a file in `migrations/version/{SHA}\_this\_will\_be\_in\_the\_migration\_filename.py`.

[Example commit](https://github.com/apache/incubator-superset/commit/d3e83b0fd572c9d6c1297543d415a332858e262)

1. Upgrade the DB

```bash

superset db upgrade

```

The output should look like this:

```

INFO [alembic.runtime.migration] Context impl SQLiteImpl.

INFO [alembic.runtime.migration] Will assume transactional DDL.

INFO [alembic.runtime.migration] Running upgrade 1a1d627ebd8e -> 40a0a483dd12, add\_metadata\_column\_to\_annotation\_model.py

```

1. Add column to view

Since there is a new column, we need to add it to the AppBuilder Model view.

[Example commit](https://github.com/apache/incubator-superset/pull/5745/commits/6220966e2a0a0cf3e6d87925491f8920fe8a3458)

1. Test the migration's `down` method

```bash

superset db downgrade

```

The output should look like this:

```

INFO [alembic.runtime.migration] Context impl SQLiteImpl.

INFO [alembic.runtime.migration] Will assume transactional DDL.

INFO [alembic.runtime.migration] Running downgrade 40a0a483dd12 -> 1a1d627ebd8e, add\_metadata\_column\_to\_annotation\_model.py

```

### Merging DB migrations

When two DB migrations collide, you'll get an error message like this one:

```

alembic.util.exc.CommandError: Multiple head revisions are present for

given argument 'head'; please specify a specific target

revision, '<branchname>@head' to narrow to a specific head,

or 'heads' for all heads`

```

To fix it:

1. Get the migration heads

```bash

superset db heads

```

This should list two or more migration hashes.

1. Create a new merge migration

```bash

superset db merge {HASH1} {HASH2}

```

1. Upgrade the DB to the new checkpoint

```bash

superset db upgrade

```

### SQL Lab Async

It's possible to configure a local database to operate in `async` mode,

to work on `async` related features.

To do this, you'll need to:

- Add an additional database entry. We recommend you copy the connection

string from the database labeled `main`, and then enable `SQL Lab` and the

features you want to use. Don't forget to check the `Async` box

- Configure a results backend, here's a local `FileSystemCache` example,

not recommended for production,

but perfect for testing (stores cache in `/tmp`)

```python

from cachelib.file import FileSystemCache

RESULTS\_BACKEND = FileSystemCache('/tmp/sqllab')

```

- Start up a celery worker

```shell script

celery worker --app=superset.tasks.celery\_app:app -Ofair

```

Note that:

- for changes that affect the worker logic, you'll have to

restart the `celery worker` process for the changes to be reflected.

- The message queue used is a `sqlite` database using the `SQLAlchemy`

experimental broker. Ok for testing, but not recommended in production

- In some cases, you may want to create a context that is more aligned

to your production environment, and use the similar broker as well as

results backend configuration

## Chart Parameters

Chart parameters are stored as a JSON encoded string the `slices.params` column and are often referenced throughout the code as form-data. Currently the form-data is neither versioned nor typed as thus is somewhat free-formed. Note in the future there may be merit in using something like [JSON Schema](https://json-schema.org/) to both annotate and validate the JSON object in addition to using a Mypy `TypedDict` (introduced in Python 3.8) for typing the form-data in the backend. This section serves as a potential primer for that work.

The following tables provide a non-exhausive list of the various fields which can be present in the JSON object grouped by the Explorer pane sections. These values were obtained by extracting the distinct fields from a legacy deployment consisting of tens of thousands of charts and thus some fields may be missing whilst others may be deprecated.

Note not all fields are correctly catagorized. The fields vary based on visualization type and may apprear in different sections depending on the type. Verified deprecated columns may indicate a missing migration and/or prior migrations which were unsucessful and thus future work may be required to clean up the form-data.

### Datasource & Chart Type

| Field | Type | Notes |

| ----------------- | -------- | ----------------------------------- |

| `database\_name` | \_string\_ | \_Deprecated?\_ |

| `datasource` | \_string\_ | `<datasouce\_id>\_\_<datasource\_type>` |

| `datasource\_id` | \_string\_ | \_Deprecated?\_ See `datasource` |

| `datasource\_name` | \_string\_ | \_Deprecated?\_ |

| `datasource\_type` | \_string\_ | \_Deprecated?\_ See `datasource` |

| `viz\_type` | \_string\_ | The \*\*Visualization Type\*\* widget |

### Time

| Field | Type | Notes |

| ---------------------- | --------------- | ------------------------------------- |

| `druid\_time\_origin` | \_string\_ | The Druid \*\*Origin\*\* widget |

| `granularity` | \_string\_ | The Druid \*\*Time Granularity\*\* widget |

| `granularity\_sqla` | \_string\_ | The SQLA \*\*Time Column\*\* widget |

| `time\_grain\_sqla` | \_string\_ | The SQLA \*\*Time Grain\*\* widget |

| `time\_range` | \_string\_ | The \*\*Time range\*\* widget |

### GROUP BY

| Field | Type | Notes |

| ------------------------- | --------------- | --------------------------- |

| `metrics` | \_array(string)\_ | See Query section |

| `order\_asc` | - | See Query section |

| `row\_limit` | - | See Query section |

| `timeseries\_limit\_metric` | - | See Query section |

### NOT GROUPED BY

| Field | Type | Notes |

| --------------- | --------------- | ----------------------- |

| `order\_by\_cols` | \_array(string)\_ | The \*\*Ordering\*\* widget |

| `row\_limit` | - | See Query section |

### Y Axis 1

| Field | Type | Notes |

| --------------- | ---- | -------------------------------------------------- |

| `metric` | - | The \*\*Left Axis Metric\*\* widget. See Query section |

| `y\_axis\_format` | - | See Y Axis section |

### Y Axis 2

| Field | Type | Notes |

| ----------------- | -------- | --------------------------------------------------- |

| `metric\_2` | - | The \*\*Right Axis Metric\*\* widget. See Query section |

### Query

| Field | Type | Notes |

| ------------------------------------------------------------------------------------------------------ | ------------------------------------------------- | ------------------------------------------------- |

| `adhoc\_filters` | \_array(object)\_ | The \*\*Filters\*\* widget |

| `columns` | \_array(string)\_ | The \*\*Breakdowns\*\* widget |

| `groupby` | \_array(string)\_ | The \*\*Group by\*\* or \*\*Series\*\* widget |

| `limit` | \_number\_ | The \*\*Series Limit\*\* widget |

| `metric`<br>`metric\_2`<br>`metrics`<br>`percent\_mertics`<br>`secondary\_metric`<br>`size`<br>`x`<br>`y` | \_string\_,\_object\_,\_array(string)\_,\_array(object)\_ | The metric(s) depending on the visualization type |

| `order\_asc` | \_boolean\_ | The \*\*Sort Descending\*\* widget |

| `row\_limit` | \_number\_ | The \*\*Row limit\*\* widget |

| `timeseries\_limit\_metric` | \_object\_ | The \*\*Sort By\*\* widget |

The `metric` (or equivalent) and `timeseries\_limit\_metric` fields are all composed of either metric names or the JSON representation of the `AdhocMetric` TypeScript type. The `adhoc\_filters` is composed of the JSON represent of the `AdhocFilter` TypeScript type (which can comprise of columns or metrics depending on whether it is a WHERE or HAVING clause). The `all\_columns`, `all\_columns\_x`, `columns`, `groupby`, and `order\_by\_cols` fields all represent column names.

### Chart Options

| Field | Type | Notes |

| --------------------- | --------- | ------------------------------------------------ |

| `color\_picker` | \_object\_ | The \*\*Fixed Color\*\* widget |

| `label\_colors` | \_object\_ | The \*\*Color Scheme\*\* widget |

| `normalized` | \_boolean\_ | The \*\*Normalized\*\* widget |

### Y Axis

| Field | Type | Notes |

| ------------------- | --------------- | ---------------------------- |

| `y\_axis\_2\_label` | \_N/A\_ | \_Deprecated?\_ |

| `y\_axis\_format` | \_string\_ | The \*\*Y Axis Format\*\* widget |

| `y\_axis\_zero` | \_N/A\_ | \_Deprecated?\_ |

Note the `y\_axis\_format` is defined under various section for some charts.

### Other

| Field | Type | Notes |

| -------------- | -------- | ------------ |

| `color\_scheme` | \_string\_ | |

### Unclassified

| Field | Type | Notes |

| ------------------------------- | ----- | ----- |

| `add\_to\_dash` | \_N/A\_ | |

| `code` | \_N/A\_ | |

| `collapsed\_fieldsets` | \_N/A\_ | |

| `comparison type` | \_N/A\_ | |

| `country\_fieldtype` | \_N/A\_ | |

| `default\_filters` | \_N/A\_ | |

| `entity` | \_N/A\_ | |

| `expanded\_slices` | \_N/A\_ | |

| `extra\_filters` | \_N/A\_ | |

| `filter\_immune\_slice\_fields` | \_N/A\_ | |

| `filter\_immune\_slices` | \_N/A\_ | |

| `flt\_col\_0` | \_N/A\_ | |

| `flt\_col\_1` | \_N/A\_ | |

| `flt\_eq\_0` | \_N/A\_ | |

| `flt\_eq\_1` | \_N/A\_ | |

| `flt\_op\_0` | \_N/A\_ | |

| `flt\_op\_1` | \_N/A\_ | |

| `goto\_dash` | \_N/A\_ | |

| `import\_time` | \_N/A\_ | |

| `label` | \_N/A\_ | |

| `linear\_color\_scheme` | \_N/A\_ | |

| `new\_dashboard\_name` | \_N/A\_ | |

| `new\_slice\_name` | \_N/A\_ | |

| `num\_period\_compare` | \_N/A\_ | |

| `period\_ratio\_type` | \_N/A\_ | |

| `perm` | \_N/A\_ | |

| `rdo\_save` | \_N/A\_ | |

| `refresh\_frequency` | \_N/A\_ | |

| `remote\_id` | \_N/A\_ | |

| `resample\_fillmethod` | \_N/A\_ | |

| `resample\_how` | \_N/A\_ | |

| `rose\_area\_proportion` | \_N/A\_ | |

| `save\_to\_dashboard\_id` | \_N/A\_ | |

| `schema` | \_N/A\_ | |

| `series` | \_N/A\_ | |

| `show\_bubbles` | \_N/A\_ | |

| `slice\_name` | \_N/A\_ | |

| `timed\_refresh\_immune\_slices` | \_N/A\_ | |

| `userid` | \_N/A\_ | |