Alscalator

Release 0.1.18

Christophe Duong

Jul 24, 2020

CONTENTS:

CHAPTER

ONE

AISCALATOR

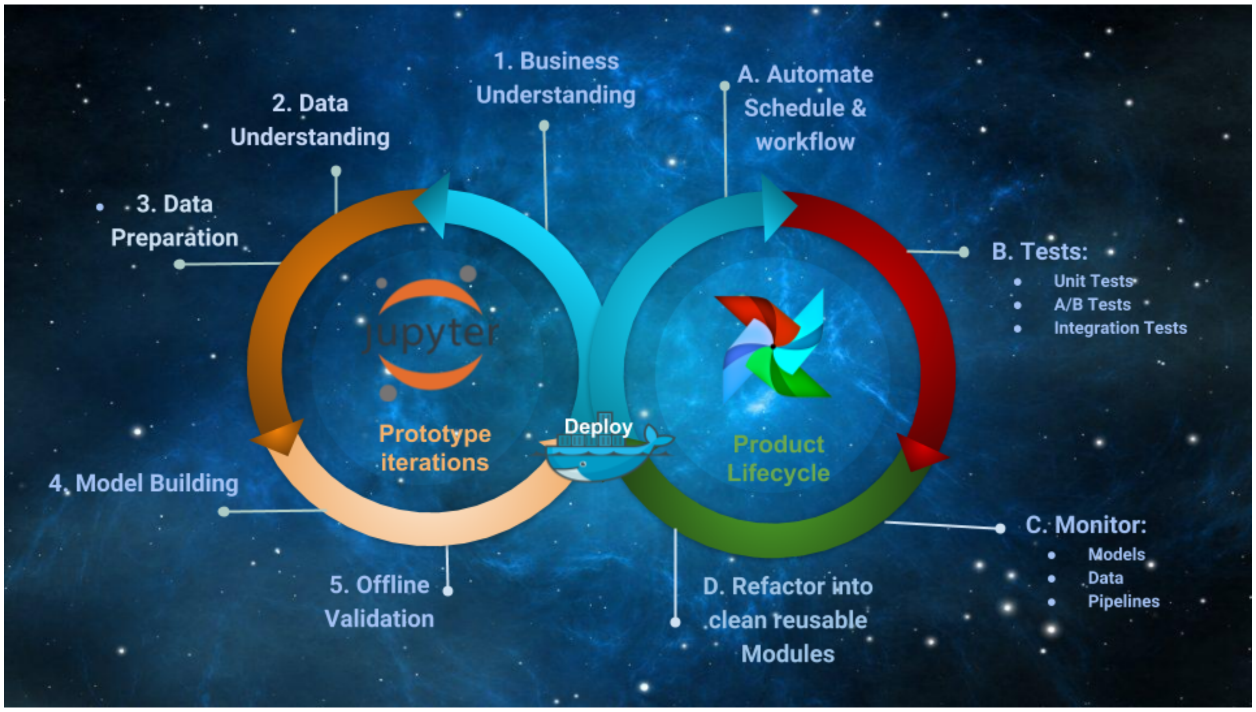
Docs	
Tests	
Package	

- Free software: Apache Software License 2.0
- Website: http://www.aiscalate.com
- Documentation: https://aiscalator.readthedocs.io/en/latest/
- Bugs: https://github.com/aiscalate/aiscalator/issues

1.1 Key Features

Aiscalator is a toolbox to enable your team streamlining processes from innovation to productization with:

- **Jupyter workbench**
  - Explore Data, Prototype Solutions
- **Docker wrapper tools**
  - Share Code, Deploy Reproducible Environments
- **Airflow machinery**
  - Schedule Tasks, Refine Products
- Data Science and Data Engineering best practices


1. Business Understanding
2. Data Understanding
3. Data Preparation
4. Model Building
5. Offline Validation
jupyter
Prototype iterations
Deploy
Product Lifecycle
A. Automate Schedule & workflow
B. Tests:
Unit Tests
A/B Tests
Integration Tests
C. Monitor:
Models
Data
Pipelines
D. Refactor into clean reusable Modules

CHAPTER TWO

QUICK START

2.1 Installation

Test if prerequisite softwares are installed:

```
docker --version
docker-compose --version
pip --version
```

Install AIscalator tool:

```
git clone https://github.com/Aiscalate/aiscalator.git
cd aiscalator/
make install
```

Great, we are now ready to use the AIscalator!

The following setup commands are completely optional because they are dealing with prebuilding Docker images. If you choose not to do it at this point, they will get built later on whenever they are required.

However, since producing a Docker image requires a certain amount of time to download, install packages, and sometimes even compiling them, these installation steps can be initiated right away all at once. Thus, you should be free to go enjoy a nice coffee break!

You might want to customize your environment with the AIscalator, this will ask you few questions:

```
aiscalator setup
```

Build docker images to run Jupyter environments:

```
aiscalator jupyter setup
```

Build docker image to run Airflow:

```
# aiscalator airflow setup <path-to-workspace-folder>
# for example,
aiscalator airflow setup $PWD
```

2.2 Start working

AIscalator commands dealing with jupyter are defining tasks in Airflow jargon; In our case, they are all wrapped inside a Docker container. We also refer to them as Steps.

Whereas AIscalator commands about airflow are made to author, schedule and monitor DAGs (Directed Acyclic Graphs). They define how a workflow is composed of multiple steps, their dependencies and execution times or triggers.

2.3 Jupyter

Create a new Jupyter notebook to work on, define corresponding AIscalator step:

```
# aiscalator jupyter new <path-to-store-new-files>
# For example,
aiscalator jupyter new project
# (CTRL + c to kill when done)
```

Or you can edit an existing AIscalator step:

```
# aiscalator jupyter edit <aiscalator step>
# For example, if you cloned the git repository:
aiscalator jupyter edit resources/example/example.conf
# (CTRL + c to kill when done)
```

Run the step without GUI:

```
# aiscalator jupyter run <aiscalator task>
# For example, if you cloned the git repository:
aiscalator jupyter run resources/example/example.conf
```

2.4 Airflow

Start Airflow services:

```
aiscalator airflow start
```

Create a new AIscalator DAG, define the airflow job:

```
# aiscalator airflow new <path-to-store-new-files>
# For example,
aiscalator airflow new project
# (CTRL + c to kill when done)
```

Or you can edit an existing AIscalator DAG:

```
# aiscalator airflow edit <aiscalator DAG>
# For example, if you cloned the git repository:
aiscalator airflow edit resources/example/example.conf
# (CTRL + c to kill when done)
```

Schedule AIscalator DAG into local airflow dags folder:

```
# aiscalator airflow push <aiscalator DAG>
# For example, if you cloned the git repository:
aiscalator airflow push resources/example/example.conf
```

Stop Airflow services:

```
aiscalator airflow stop
```

CHAPTER THREE

INSTALLATION

3.1 Stable release

To install aiscalator, run this command in your terminal:

```
$ pip install aiscalator
```

This is the preferred method to install aiscalator, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

3.2 From sources

The sources for aiscalator can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/aiscalate/aiscalator
```

Or download the tarball:

```
$ curl  -OL https://codeload.github.com/Aiscalate/aiscalator/legacy.tar.gz/master
```

Once you have a copy of the source, you can install it with:

```
$ make install
```

3.3 Development environment on Mac OS

First, Install docker: https://docs.docker.com/docker-for-mac/install/

Then, we need to have a proper python environment installed: We would recommend using Homebrew to install pyenv. Best practices would also to be using virtualenv and virtualenvwrapper.

```
brew install python

# to successfully run all tests with tox, make sure to have
# each version of python installed
brew install pyenv
brew install pyenv-virtualenv
```

(continues on next page)

(continued from previous page)

```
brew install pyenv-virtualenvwrapper

# Follow their documentations to add the proper environment variables to your
# startup scripts (~/,bash_profile etc)

pyenv install 3.4.9 3.5.6 3.6.7 3.7.1 pypy3.5-6.0.0

# in case pyenv runs into errors when running tox because it
# cannot find python3.X (even though python3.X.Y is installed):
git clone git://github.com/concordusapps/pyenv-implict.git ~/.pyenv/plugins/pyenv-
↪implict


pip install -r requirements_dev.txt

# run all tests
tox
```

CHAPTER

FOUR

USAGE

The project is still in Alpha version, there is still a lot of work to be done. At the moment, you can install using two methods

1) Install from PyPI:

```
pip install aiscalator
```

2) or Download latest version directly from git and install:

```
git clone https://github.com/Aiscalate/aiscalator.git
cd aiscalator/
make install
```

After cloning the project, you can start testing the following commands at the moment:

```
aiscalator jupyter new examples
ls -l examples/
aiscalator jupyter edit resources/example/example.conf
aiscalator jupyter run resources/example/example.conf
aiscalator airflow start
```

CHAPTER

FIVE

FEATURES

5.1 Docker docker_image Configuration

- docker_image support docker_extra_options to be passed as a list of arguments to be appended to the docker execution.

CHAPTER

SIX

AISCALATOR

6.1 aiscalator package

6.1.1 Subpackages

aiscalator.airflow package

Submodules

aiscalator.airflow.command module

Implementations of commands for Airflow

`aiscalator.airflow.command.`**`airflow_cmd`**(*conf:* aiscalator.core.config.AiscalatorConfig, *service='webserver'*, *cmd=None*)

Execute an airflow subcommand

Parameters

- **conf** (*AiscalatorConfig*) – Configuration object for the application
- **service** (*string*) – service name of the container where to run the command
- **cmd** (*list*) – subcommands to run

`aiscalator.airflow.command.`**`airflow_down`**(*conf:* aiscalator.core.config.AiscalatorConfig)

Stop an airflow environment

Parameters **conf** (*AiscalatorConfig*) – Configuration object for the application

`aiscalator.airflow.command.`**`airflow_edit`**(*conf:* aiscalator.core.config.AiscalatorConfig)

Starts an airflow environment

Parameters **conf** (*AiscalatorConfig*) – Configuration object for the application

`aiscalator.airflow.command.`**`airflow_push`**(*conf:* aiscalator.core.config.AiscalatorConfig)

Starts an airflow environment

Parameters **conf** (*AiscalatorConfig*) – Configuration object for the application

`aiscalator.airflow.command.`**`airflow_setup`**(*conf:* aiscalator.core.config.AiscalatorConfig, *config_home: str*, *workspace: list*, *append: bool = True*)

Setup the airflow configuration files and environment

Parameters

- **conf** (*AiscalatorConfig*) – Configuration object for the application
- **config_home** (*str*) – path to the configuration home directory
- **workspace** (*list*) – List of path to directories to mount as volumes to airflow workers to use as workspaces
- **append** (*bool*) – flag to tell if workspace should be appended to the list in the config or replace it.

`aiscalator.airflow.command.`**`airflow_up`**(*conf:* aiscalator.core.config.AiscalatorConfig)

Starts an airflow environment

Parameters **conf** (*AiscalatorConfig*) – Configuration object for the application

aiscalator.airflow.cli module

CLI module for Airflow related commands.

Module contents

aiscalator.core package

Submodules

aiscalator.core.config module

Handles configurations files for the application

class `aiscalator.core.config.`**`AiscalatorConfig`**(*config=None*, *step_selection=None*, *dag_selection=None*)

Bases: `object`

A configuration object for the Aiscalator application.

This object stores:

- global configuration for the whole application
- configuration for a particular context specified in a step configuration file.
- In this case, we might even focus on a particular step.

…

Variables

- **`_app_conf`** – global configuration object for the application
- **`_config_path`** (`str`) – path to the configuration file (or plain configuration as string)
- **`_step_name`** (`str`) – name of the currently processed step
- **`_step`** – configuration object for the currently processed step
- **`_dag_name`** (`str`) – name of the currently processed dag
- **`_dag`** – configuration object for the currently processed dag

Methods

airflow_docker_compose_file()	Return the configuration file to bring airflow services up.
app_config()	**returns** *str* – the configuration object for the aiscalator application
app_config_has(field)	Tests if the applicatin config has a configuration value for the field.
app_config_home()	Return the path to the app configuration folder.
config_path()	**returns** *str* – Returns the path to the step configuration file.
dag_container_name()	Return the docker container name to execute this step
dag_field(field)	Returns the value associated with the field for the currently focused dag.
dag_file_path(string)	Returns absolute path of a file from a field of the currently focused dag.
dag_name()	Returns the name of the currently focused dag
has_dag_field(field)	Tests if the currently focused dag has a configuration value for the field.
has_step_field(field)	Tests if the currently focused step has a configuration value for the field.
redefine_airflow_workspaces(workspaces)	Modify the configuration file to change the value of the airflow workspaces
redefine_app_config_home(config_home)	Modify the configuration file to change the value of the application configuration home directory.
root_dir()	**returns** *str* – Returns the path to the folder containing the
step_container_name()	Return the docker container name to execute this step
step_extract_parameters()	Returns a list of docker parameters
step_field(field)	Returns the value associated with the field for the currently focused step.
step_file_path(string)	Returns absolute path of a file from a field of the currently focused step.
step_name()	Returns the name of the currently focused step
step_notebook_output_path(notebook)	Generates the name of the output notebook
user_env_file([job])	Find a list of env files to pass to docker containers
user_id()	**returns** *str* – the user id stored when the application was first setup
validate_config()	Check if all the fields in the reference config are defined in focused steps too.

airflow_docker_compose_file()
Return the configuration file to bring airflow services up.

app_config()

Returns *str* – the configuration object for the aiscalator application

app_config_has(*field*) → bool
Tests if the applicatin config has a configuration value for the field.

app_config_home() → str
Return the path to the app configuration folder.

config_path()

Returns *str* – Returns the path to the step configuration file. If it was an URL, it will return the path to the temporary downloaded version of it. If it was a plain string, then returns None

dag_container_name() → str
Return the docker container name to execute this step

dag_field(*field*)
Returns the value associated with the field for the currently focused dag.

dag_file_path(*string*)
Returns absolute path of a file from a field of the currently focused dag.

dag_name()
Returns the name of the currently focused dag

has_dag_field(*field*) → bool
Tests if the currently focused dag has a configuration value for the field.

has_step_field(*field*) → bool
Tests if the currently focused step has a configuration value for the field.

redefine_airflow_workspaces(*workspaces*)
Modify the configuration file to change the value of the airflow workspaces

Parameters **workspaces** (*list*) – list of workspaces to bind to airflow

Returns *AiscalatorConfig* – the new configuration object

redefine_app_config_home(*config_home*)
Modify the configuration file to change the value of the application configuration home directory.

Parameters **config_home** (*str*) – path to the new configuration home

Returns *AiscalatorConfig* – the new configuration object

root_dir()

Returns *str* – Returns the path to the folder containing the configuration file

step_container_name() → str
Return the docker container name to execute this step

step_extract_parameters() → list
Returns a list of docker parameters

step_field(*field*)
Returns the value associated with the field for the currently focused step.

step_file_path(*string*)
Returns absolute path of a file from a field of the currently focused step.

step_name()
Returns the name of the currently focused step

step_notebook_output_path(*notebook*) → str
Generates the name of the output notebook

user_env_file(*job=None*) → list
Find a list of env files to pass to docker containers

Parameters **job** – Optional step or dag config

Returns *List* – env files

user_id() → str

Returns *str* – the user id stored when the application was first setup

validate_config()
Check if all the fields in the reference config are defined in focused steps too. Otherwise raise an Exception (either pyhocon.ConfigMissingException or pyhocon.ConfigWrongTypeException)

`aiscalator.core.config.`**convert_to_format**(*file: str*, *output: str*, *output_format: str*)
Converts a HOCON file to another format

Parameters

- **file** (*str*) – hocon file to convert
- **output** (*str*) – output file to produce
- **output_format** (*str*) – format of the output file

Returns *str* – the output file

`aiscalator.core.config.`**generate_user_id**() → str

Returns *str* – Returns a string identifying this user when the setup was run first

aiscalator.core.log_regex_analyzer module

Class to parse output logs from subprocess and catch particular expressions

class `aiscalator.core.log_regex_analyzer.`**LogRegexAnalyzer**(*pattern=None*, *log_level=10*)
Bases: `object`

A regular expression analyzer object to parse logs and extract values from patterns in the logs. . . .

Variables

- **_artifact** (`str`) – Value of the pattern found in the logs
- **_pattern** (`bytes`) – Regular expression to search for in the logs

Methods

artifact()	Returns the artifact extracted from the logs.
grep_logs(pipe)	Reads the logs and extract values defined by the pattern

artifact()
Returns the artifact extracted from the logs.

grep_logs(*pipe*)
Reads the logs and extract values defined by the pattern

Parameters **pipe** – Stream of logs to analyze

aiscalator.core.utils module

Various Utility functions

class aiscalator.core.utils.**BackgroundThreadRunner**(*command*, *log_function*, *no_redirect=False*)

Bases: `object`

Worker Thread to run logging output in the background

...

Variables

- **_process** – Process object of the command running in the background
- **_log_function** (*function(stream -> bool)*) – callback function to log the output of the command
- **_no_redirect** (*bool*) – whether the subprocess STDOUT and STDERR should be redirected to logs
- **_worker** (*Thread*) – Thread object

Methods

process()	Returns the process object.
run()	Starts the Thread, process the output of the process.

process()

Returns the process object.

run()

Starts the Thread, process the output of the process.

aiscalator.core.utils.**check_notebook**(*logger*, *code_path*, *from_format='py:percent'*)

Checks existence of notebook file and regenerates using jupytext from associated .py file if possible. Otherwise, create an empty notebook file.

Parameters

- **code_path** (*str*) – path to the notebook to check
- **from_format** (*str*) – jupytext format of the .py input file

aiscalator.core.utils.**check_notebook_dir**(*logger*, *code_path*, *from_format='py:percent'*)

Check a folder and generate all notebook files that might be required in that folder.

Parameters

- **code_path** (*str*) – path to a file in the folder
- **from_format** (*str*) – jupytext format of potential .py files

aiscalator.core.utils.**copy_replace**(*src*, *dst*, *pattern=None*, *replace_value=None*)

Copies a file from src to dst replacing pattern by replace_value

Parameters

- **src** (*string*) – Path to the source filename to copy from
- **dst** (*string*) – Path to the output filename to copy to

- **pattern** – list of Patterns to replace inside the src file
- **replace_value** – list of Values to replace by in the dst file

`aiscalator.core.utils.`**`data_file`**(*path*)

Utility function to find resources data file packaged along with code

Parameters **path** (*path*) – path to the resource file in the package

Returns *absolute path to the resource data file*

`aiscalator.core.utils.`**`find`**(*collection*, *item*, *field='name'*)

Finds an element in a collection which has a field equal to particular item value

Parameters

- **collection** (*Set*) – Collection of objects
- **item** – value of the item that we are looking for
- **field** (*string*) – Name of the field from the object to inspect

Returns *object* – Corresponding element that has a field matching item in the collection

`aiscalator.core.utils.`**`format_file_content`**(*content*, *prefix=''*, *suffix=''*)

Reformat the content of a file line by line, adding prefix and suffix strings.

Parameters

- **content** (*str*) – path to the file to reformat its content
- **prefix** (*str*) – add to each line this prefix string
- **suffix** (*str*) – add to each line this suffix string

Returns *str* – Formatted content of the file

`aiscalator.core.utils.`**`log_info`**(*pipe*)

Default logging function

`aiscalator.core.utils.`**`notebook_file`**(*code_path*, *from_format='py:percent'*)

Parse a path to return both the ipynb and py versions of the file.

Parameters

- **code_path** (*str*) – path to a file
- **from_format** (*str*) – jupytext format of potential .py files

Returns *(str, str)* – tuple of 2 paths to ipynb and py files

`aiscalator.core.utils.`**`sha256`**(*file: str*)

Reads a file content and returns its sha256 hash.

`aiscalator.core.utils.`**`subprocess_run`**(*command*, *log_function=<function log_info>*, *no_redirect=False*, *wait=True*)

Run command in a subprocess while redirecting output to log_function.

The subprocess either runs synchroneoulsy or in the background depending on the wait parameter.

Parameters

- **command** (*List*) – Command to run in the subprocess
- **log_function** (*function*) – Callback function to log the output of the subprocess
- **no_redirect** (*bool*) – whether the subprocess STDOUT and STDERR should be redirected to logs

- **wait** (*bool*) – Whether the subprocess should be run synchroneously or in the background

Returns

- *int* – return code of the subprocess
- *BackgroundThreadRunner* – the thread running in the background

`aiscalator.core.utils.`**`wait_for_jupyter_lab`**(*commands*, *logger*, *notebook*, *port*, *folder*)

Starts jupyter lab and wait for it to start, returning the url it's running from.

Parameters

- **commands** (*list*) – List of commands to run to start the process
- **logger** (*logging.Logger*) – Logger object
- **notebook** (*str*) – path to the notebook
- **port** – port on which the jupyter lab is listening
- **folder** (*str*) – path in the container to reach the notebook

Returns *str* – url from which it is serving the jupyter lab

Module contents

aiscalator.jupyter package

Submodules

aiscalator.jupyter.command module

Implementations of commands for Jupyter

`aiscalator.jupyter.command.`**`jupyter_edit`**(*conf: aiscalator.core.config.AiscalatorConfig*, *param=None*, *param_raw=None*)

Starts a Jupyter Lab environment configured to edit the focused step

Parameters

- **conf** (*AiscalatorConfig*) – Configuration object for the step
- **param** (*list*) – list of tuples of parameters
- **param_raw** (*list*) – list of tuples of raw parameters

Returns *string* – Url of the running jupyter lab

`aiscalator.jupyter.command.`**`jupyter_new`**(*name*, *path*, *output_format='hocon'*)

Starts a Jupyter Lab environment configured to edit a brand new step

Parameters

- **name** (*str*) – name of the new step
- **path** (*str*) – path to where the new step files should be created
- **output_format** (*str*) – the format of the new configuration file to produce

Returns *string* – Url of the running jupyter lab `aiscalator.jupyter.command.`**`jupyter_run`**`(`*conf: aiscalator.core.config.AiscalatorConfig, prepare_only=False, param=None, param_raw=None*`)`

Executes the step in browserless mode using papermill

Parameters

- **conf** (*AiscalatorConfig*) – Configuration object for the step
- **prepare_only** (*bool*) – Indicates if papermill should replace the parameters of the notebook only or it should execute all the cells too

Returns *string* – the path to the output notebook resulting from the execution of this step

aiscalator.jupyter.cli module

CLI module for Jupyter related commands.

`aiscalator.jupyter.cli.`**`prompt_edit`**`(`*file*`)`

When creating a new step, if it is already defined, ask to edit instead

Parameters **file** (*str*) – existing configuration file

`aiscalator.jupyter.cli.`**`run_auto_update`**`()`

Checks and tries to update Aiscalator itself from Pypi if necessary

`aiscalator.jupyter.cli.`**`update_aiscalator`**`()`

Create and run Thread to execute auto update in the background

Module contents

6.1.2 Submodules

6.1.3 aiscalator.api module

API module of the AIscalator tool.

This module presents the entrypoint to use from a python script. It is intended to be used like this, usually from an airflow DAG scripts defining an AIscalator task:

```
from aiscalator import api

api.jupyter_run("path/to/config.conf", "step_name")
```

`aiscalator.api.`**`jupyter_run`**`(`*config, notebook=None, prepare_only=False, param=None, param_raw=None*`)`

Executes the step in browserless mode using papermill

Parameters

- **config** (*str*) – path to the configuration file
- **notebook** (*str*) – name of node to run, if None, then run the first one
- **parameters** (*list*) – List of parameters and their values
- **prepare_only** (*bool*) – Indicates if papermill should replace the parameters of the notebook only or it should execute all the cells too

Returns *string* – the path to the output notebook resulting from the execution of this step

6.1.4 aiscalator.cli module

Command Line Interface of the AIscalator tool.

Using the python click package (https://click.palletsprojects.com/en/7.x/), this module defines all the entry points to the application.

6.1.5 Module contents

Aiscalator Module: CLI to AIscalate

CHAPTER

SEVEN

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

7.1 Types of Contributions

7.1.1 Report Bugs

Report bugs at https://github.com/aiscalate/aiscalator/issues

If you are reporting a bug, please include:

- aiscalator version:
- Python version:
- Operating System:

Description

Describe what you were trying to get done. Tell us what happened, what went wrong, and what you expected to happen.

What I Did:

```
Paste the command(s) you ran and the output.
If there was a crash, please include the traceback here.
```

Screenshots If applicable, add screenshots to help explain your problem.

Additional context Add any other context about the problem here.

7.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

7.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

7.1.4 Write Documentation

aiscalator could always use more documentation, whether as part of the official aiscalator docs, in docstrings, or even on the web in blog posts, articles, and such.

7.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/aiscalate/aiscalator/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Describe alternatives you've considered
- 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 :)

7.2 Get Started!

Ready to contribute? Here's how to set up *aiscalator* for local development.

1. Fork the *aiscalator* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/aiscalator.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv aiscalator
   $ cd aiscalator/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 aiscalator tests
   $ python setup.py test or py.test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

7.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in FEATURES.rst.
3. The pull request should work for Python 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/Aiscalate/aiscalator/pull_requests and make sure that the tests pass for all supported Python versions.

7.4 Tips

To run a subset of tests:

```
$ py.test tests.test_aiscalator
```

7.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER

EIGHT

CODE OF CONDUCT

Like the technical community as a whole, the AIscalate team and community is made up of a mixture of professionals and volunteers from all over the world, working on every aspect of the mission - including mentorship, teaching, and connecting people.

Diversity is one of our huge strengths, but it can also lead to communication issues and unhappiness. To that end, we have a few ground rules that we ask people to adhere to. This code applies equally to founders, mentors and those seeking help and guidance.

This isn't an exhaustive list of things that you can't do. Rather, take it in the spirit in which it's intended - a guide to make it easier to enrich all of us and the technical communities in which we participate.

This code of conduct applies to all spaces managed by the Read the Docs project. This includes IRC, the mailing lists, the issue tracker, and any other forums created by the project team which the community uses for communication. In addition, violations of this code outside these spaces may affect a person's ability to participate within them.

If you believe someone is violating the code of conduct, we ask that you report it by emailing chris@aiscalate.com.

- **Be friendly and patient.**
- **Be welcoming.** We strive to be a community that welcomes and supports people of all backgrounds and identities. This includes, but is not limited to members of any race, ethnicity, culture, national origin, colour, immigration status, social and economic class, educational level, sex, sexual orientation, gender identity and expression, age, size, family status, political belief, religion, and mental and physical ability.
- **Be considerate.** Your work will be used by other people, and you in turn will depend on the work of others. Any decision you take will affect users and colleagues, and you should take those consequences into account when making decisions. Remember that we're a world-wide community, so you might not be communicating in someone else's primary language.
- **Be respectful.** Not all of us will agree all the time, but disagreement is no excuse for poor behavior and poor manners. We might all experience some frustration now and then, but we cannot allow that frustration to turn into a personal attack. It's important to remember that a community where people feel uncomfortable or threatened is not a productive one. Members of the Read the Docs community should be respectful when dealing with other members as well as with people outside the Read the Docs community.
- **Be careful in the words that you choose.** We are a community of professionals, and we conduct ourselves professionally. Be kind to others. Do not insult or put down other participants. Harassment and other exclusionary behavior aren't acceptable. This includes, but is not limited to:
  - Violent threats or language directed against another person.
  - Discriminatory jokes and language.
  - Posting sexually explicit or violent material.
  - Posting (or threatening to post) other people's personally identifying information ("doxing").
  - Personal insults, especially those using racist or sexist terms.

- – Unwelcome sexual attention.
  - – Advocating for, or encouraging, any of the above behavior.
  - – Repeated harassment of others. In general, if someone asks you to stop, then stop.

- **When we disagree, try to understand why.** Disagreements, both social and technical, happen all the time and Read the Docs is no exception. It is important that we resolve disagreements and differing views constructively. Remember that we're different. The strength of Read the Docs comes from its varied community, people from a wide range of backgrounds. Different people have different perspectives on issues. Being unable to understand why someone holds a viewpoint doesn't mean that they're wrong. Don't forget that it is human to err and blaming each other doesn't get us anywhere. Instead, focus on helping to resolve issues and learning from mistakes.

Original text courtesy of the *Speak Up! project* http://web.archive.org/web/20141109123859/http://speakup.io/coc.html

This version was adopted from the *Django Code of Conduct* https://www.djangoproject.com/conduct

CHAPTER NINE

CREDITS

9.1 Development Lead

- Christophe Duong <chris@aiscalate.com>

9.2 Contributors

None yet. Why not be the first?

9.3 Components

This software uses the following open source components:

- Jupyter-Project notebook, lab, etc under BSD-3-Clause license
- Jupytext under MIT license
- Docker
- Jupyter-Docker-Stacks under BSD-3-Clause license
- Airflow under Apache-2 license
- Docker-Airflow under Apache-2 license
- Papermill under BSD-3-Clause license
- This project was created with Cookiecutter under BSD-3-Clause license
- with the audreyr/cookiecutter-pypackage project template.
- with the ionelmc/cookiecutter-pylibrary project template
- PyHOCON under Apache-2 license

CHAPTER

TEN

HISTORY

10.1 0.1.0 (2018-11-07)

- First Alpha release on PyPI.

10.2 0.1.11 (2019-04-26)

- Added docker_image.docker_extra_options list feature

10.3 0.1.13 (2019-06-23)

- Handle errors in Jupytext conversions
- aiscalator run subcommand exit code propagated to cli
- Concurrent aiscalator run commands is possible

CHAPTER ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX