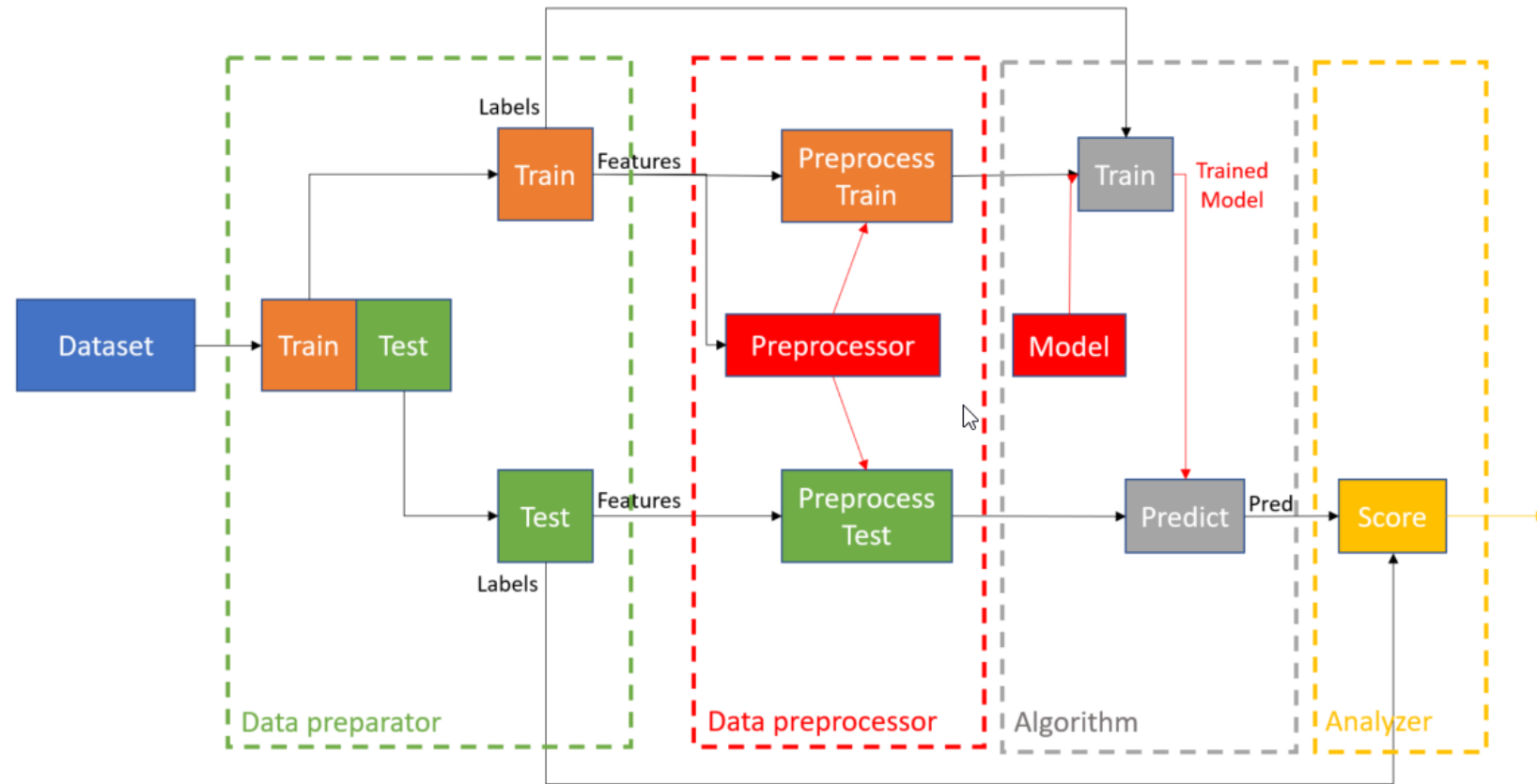


# M05 Project

Gaetan Breton & Quentin Cretenet

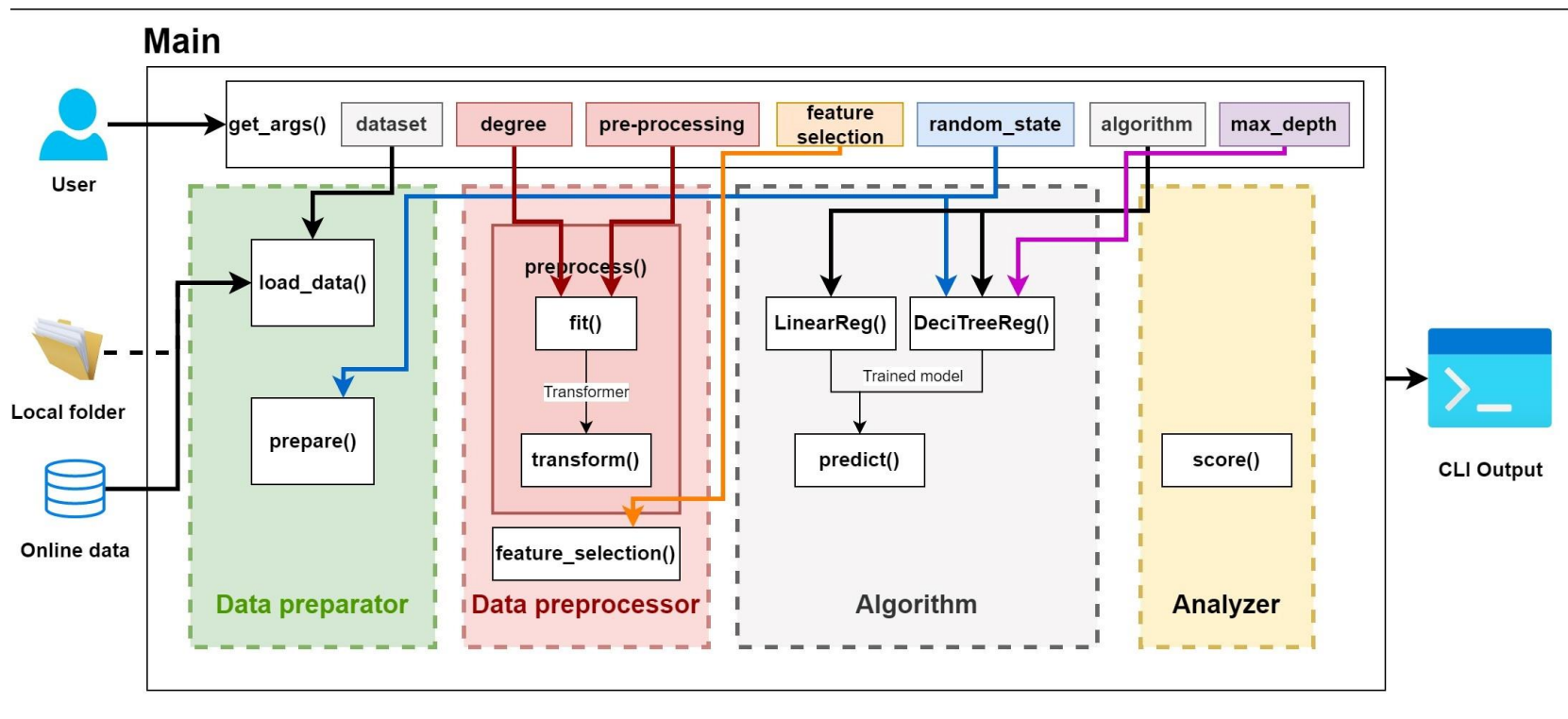
# Data and Workflow Management



# Data and Workflow Management

## Reproducibility - CLI Arguments

For reproducibility, all the parameters are accessible from a CLI (Command-Line Interface) including the value of `random_state`, `max_depth`...



# Data and Workflow Management

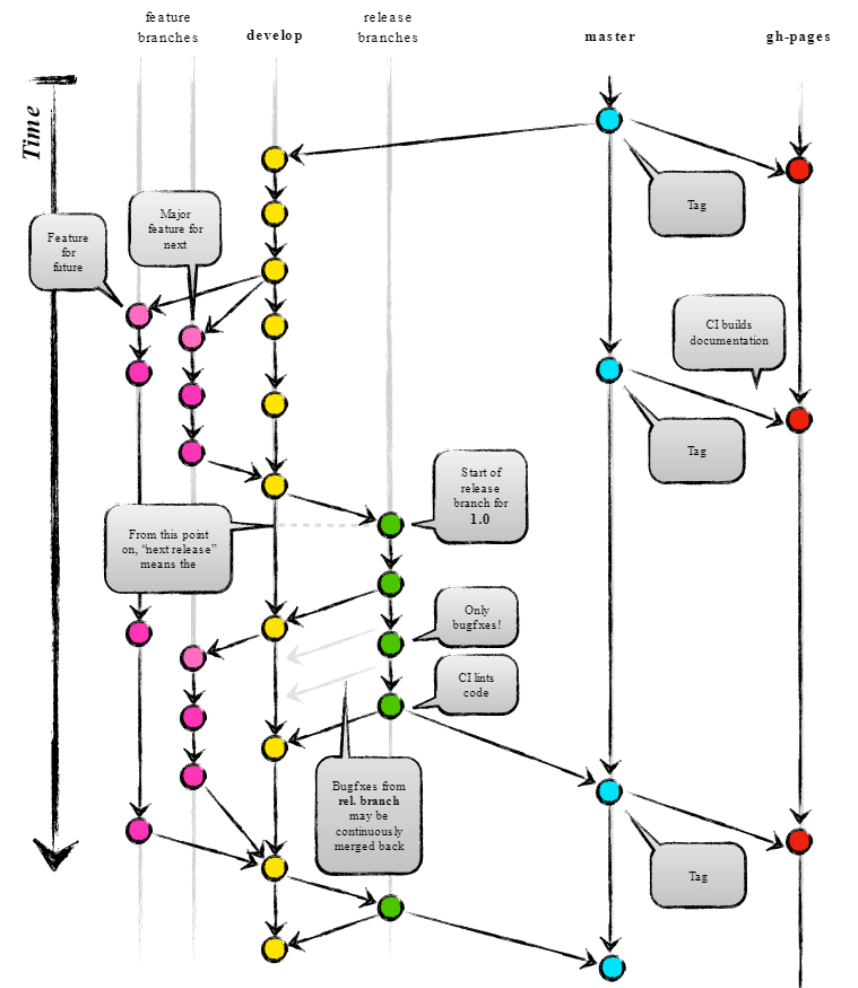
- The experiment is built in a sequential way, only one experiment at a time, no parallel programming
  - Next step: Using multiprocessing python built-in and cross validation in scikit-learn libraries could allow it
- Treat with glass-box approach:
  - Something special to implement for one of the tasks
    - => do not modify, add a step.
    - E.g. deal with the possibility of having one or two wines.
  - Respects DRY (Don't repeat yourself)

# Distribution of work

- Gaëtan : Algorithm + Analyzer
- Quentin : Preparator + Preprocessor
- Mix : CLI + CI + Documentation
- Philosophy :
  - Each person fixes the bugs in their own code (e.g. Quentin never touches the analyzer)
  - Always use GitHub issues to communicate bugs, to keep logs

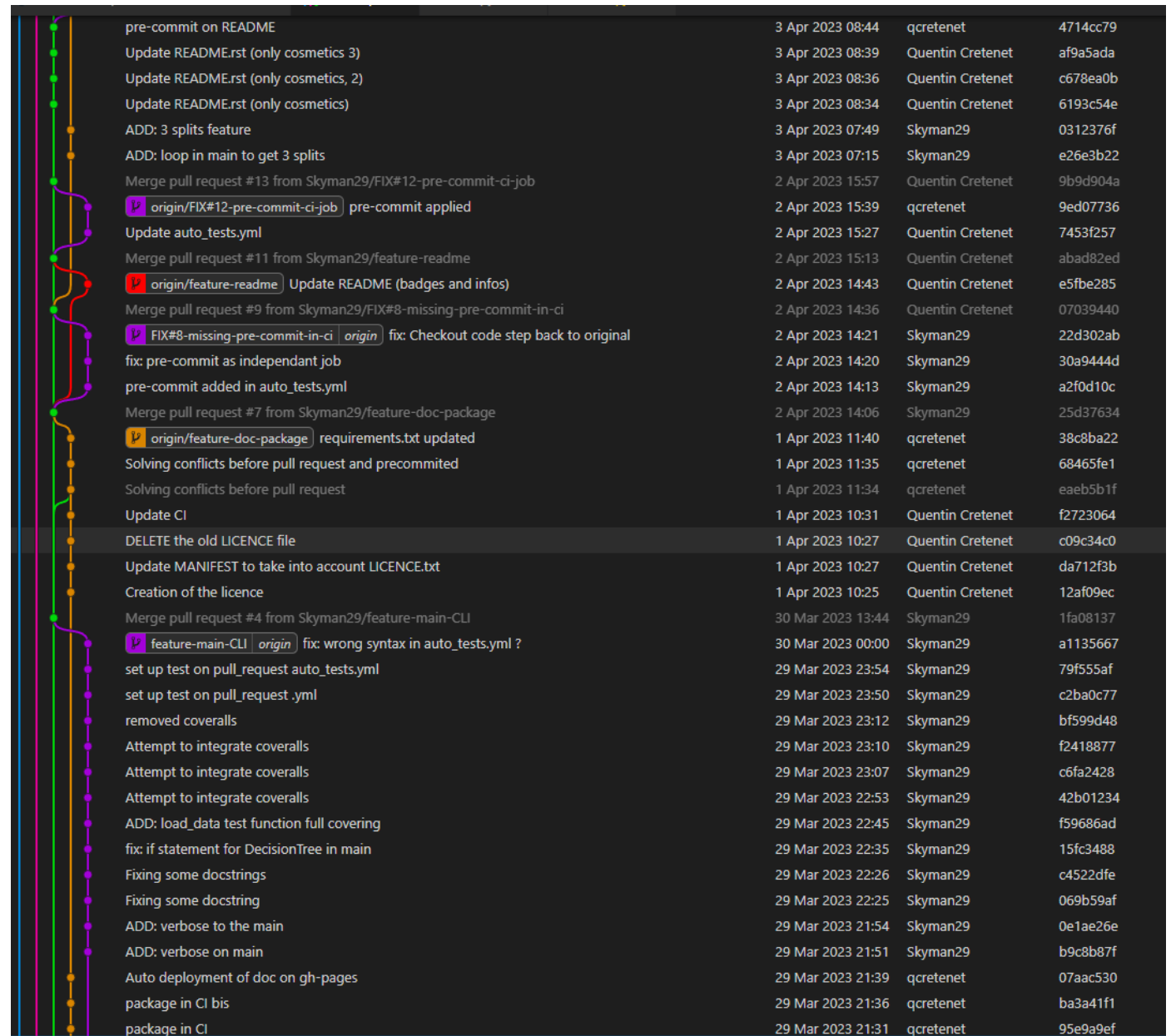
# Version Control with Git

- Work on the *dev* branch (not main), then merge dev to main when finished and stable.  
After mergin, create a release and tag (e.g. v1.0.0)
- New feature => new branch named :  
*feature-name-of-the-feature*
- Bug to fix :
  - If still working on the feature branch, just a commit
  - Otherwise, create issue (next slide github), and the branch to fix is named *FIX#(id\_of\_issue)-feature*



# Version Control with Git

## Example of the git tree

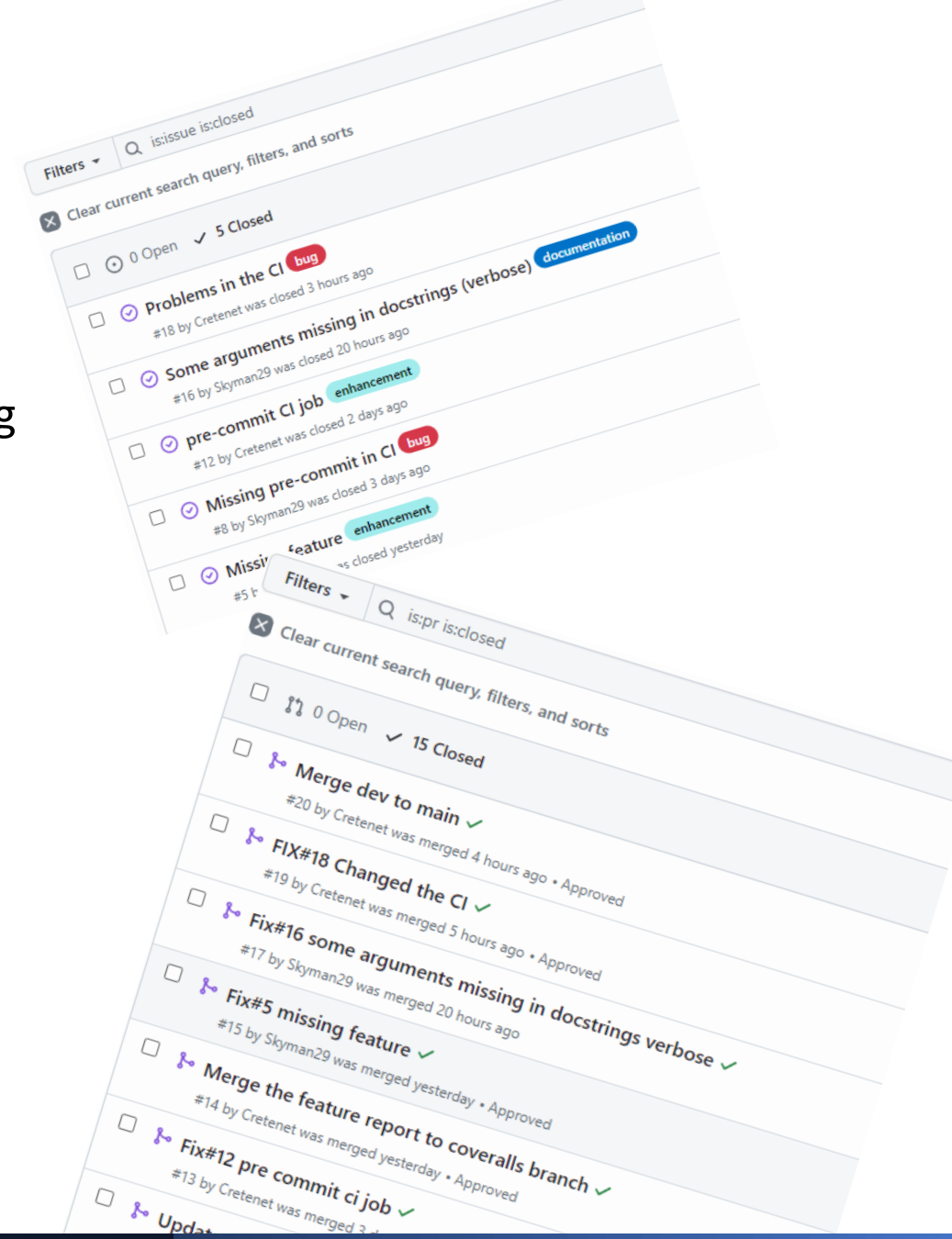


The image displays a Git commit history visualization. On the left, a vertical timeline shows the sequence of commits, with branches represented by colored lines (green for main, yellow for feature, purple for fix, orange for pre-commit). The main branch is shown as a vertical line of green dots, while feature branches are shown as horizontal lines branching off and merging back. The commit messages are listed in the center, and the commit hash, date, time, author, and commit ID are listed on the right.

pre-commit on README	3 Apr 2023 08:44	qcretenet	4714cc79
Update README.rst (only cosmetics 3)	3 Apr 2023 08:39	Quentin Cretenet	af9a5ada
Update README.rst (only cosmetics, 2)	3 Apr 2023 08:36	Quentin Cretenet	c678ea0b
Update README.rst (only cosmetics)	3 Apr 2023 08:34	Quentin Cretenet	6193c54e
ADD: 3 splits feature	3 Apr 2023 07:49	Skyman29	0312376f
ADD: loop in main to get 3 splits	3 Apr 2023 07:15	Skyman29	e26e3b22
Merge pull request #13 from Skyman29/FIX#12-pre-commit-ci-job	2 Apr 2023 15:57	Quentin Cretenet	9b9d904a
origin/FIX#12-pre-commit-ci-job pre-commit applied	2 Apr 2023 15:39	qcretenet	9ed07736
Update auto_tests.yml	2 Apr 2023 15:27	Quentin Cretenet	7453f257
Merge pull request #11 from Skyman29/feature-readme	2 Apr 2023 15:13	Quentin Cretenet	abad82ed
origin/feature-readme Update README (badges and infos)	2 Apr 2023 14:43	Quentin Cretenet	e5fbe285
Merge pull request #9 from Skyman29/FIX#8-missing-pre-commit-in-ci	2 Apr 2023 14:36	Quentin Cretenet	07039440
FIX#8-missing-pre-commit-in-ci origin fix: Checkout code step back to original	2 Apr 2023 14:21	Skyman29	22d302ab
fix: pre-commit as independant job	2 Apr 2023 14:20	Skyman29	30a9444d
pre-commit added in auto_tests.yml	2 Apr 2023 14:13	Skyman29	a2f0d10c
Merge pull request #7 from Skyman29/feature-doc-package	2 Apr 2023 14:06	Skyman29	25d37634
origin/feature-doc-package requirements.txt updated	1 Apr 2023 11:40	qcretenet	38c8ba22
Solving conflicts before pull request and precommitted	1 Apr 2023 11:35	qcretenet	68465fe1
Solving conflicts before pull request	1 Apr 2023 11:34	qcretenet	eaeb5b1f
Update CI	1 Apr 2023 10:31	Quentin Cretenet	f2723064
DELETE the old LICENCE file	1 Apr 2023 10:27	Quentin Cretenet	c09c34c0
Update MANIFEST to take into account LICENCE.txt	1 Apr 2023 10:27	Quentin Cretenet	da712f3b
Creation of the licence	1 Apr 2023 10:25	Quentin Cretenet	12af09ec
Merge pull request #4 from Skyman29/feature-main-CLI	30 Mar 2023 13:44	Skyman29	1fa08137
feature-main-CLI origin fix: wrong syntax in auto_tests.yml ?	30 Mar 2023 00:00	Skyman29	a1135667
set up test on pull_request auto_tests.yml	29 Mar 2023 23:54	Skyman29	79f555af
set up test on pull_request .yml	29 Mar 2023 23:50	Skyman29	c2ba0c77
removed coveralls	29 Mar 2023 23:12	Skyman29	bf599d48
Attempt to integrate coveralls	29 Mar 2023 23:10	Skyman29	f2418877
Attempt to integrate coveralls	29 Mar 2023 23:07	Skyman29	c6fa2428
Attempt to integrate coveralls	29 Mar 2023 22:53	Skyman29	42b01234
ADD: load_data test function full covering	29 Mar 2023 22:45	Skyman29	f59686ad
fix: if statement for DecisionTree in main	29 Mar 2023 22:35	Skyman29	15fc3488
Fixing some docstrings	29 Mar 2023 22:26	Skyman29	c4522dfe
Fixing some docstring	29 Mar 2023 22:25	Skyman29	069b59af
ADD: verbose to the main	29 Mar 2023 21:54	Skyman29	0e1ae26e
ADD: verbose on main	29 Mar 2023 21:51	Skyman29	b9c8b87f
Auto deployment of doc on gh-pages	29 Mar 2023 21:39	qcretenet	07aac530
package in CI bis	29 Mar 2023 21:36	qcretenet	ba3a41f1
package in CI	29 Mar 2023 21:31	qcretenet	95e9a9ef

# Code Sharing with GitHub

- The code is shared via a repository on GitHub.
- GitHub is used to raise issues, suggest modifications (bug fix or improvement).
- 2 keys methods to protect the main branch.
- Perform branch merging through Github Pull Requests reviewed by the other project member.
- Github actions allow us to perform CI (see next).





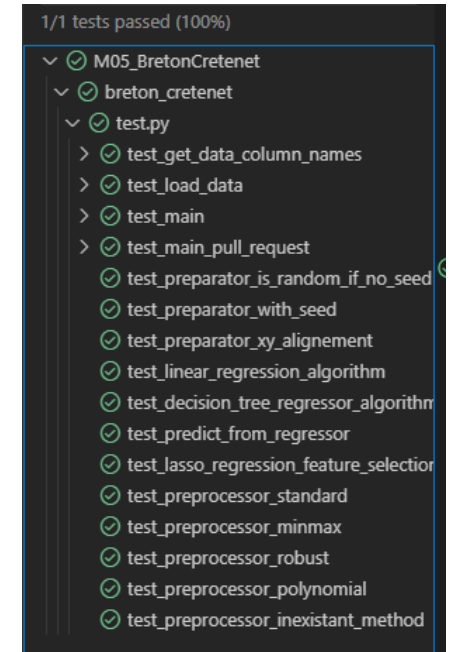
# Unit Testing and Continuous Integration

## Unit Testing

- Unit tests are all in test.py, named test\_name\_of\_the\_test
- We perform tests with pytest-cov to check the coverage

## CI – Continuous integration

- Pre-commit : Black formatting, isort and then flake8
- Automatic actions are triggered when pushing to Github or when make a pull request:
  - Run the tests with pytest-cov
    - On push, perform a set of "basic" tests
    - On Pull requests, perform push tests + explore all possible parameters (because of light dataset)
  - Send the coverage report to coveralls.io



Example of our unit tests



Current project's coverage

# Documentation

- Documentation generated with sphinx (each function has a docstring)
- The documentation is built and published to our GitHub page via the CI
- GitHub README with badges that explains:
  - How to install the package
  - How to run the program with default values
  - Where to find the documentation
  - Where to find the coverage report and how to run it yourself

```
def function_with_types_in_docstring(param1, param2):  
    """Example function with types documented in the docstring.  
  
    :pep:`484` type annotations are supported. If attribute, parameter, and  
    return types are annotated according to `PEP 484`, they do not need to be  
    included in the docstring:  
  
    Parameters  
    -----  
    param1 : int  
        The first parameter.  
    param2 : str  
        The second parameter.  
  
    Returns  
    -----  
    bool  
        True if successful, False otherwise.  
    """
```

README.rst

coverage 98% docs latest github project pypi project

## Mini project of reproducible science

This project is a simple toy project. Its only purpose is to explore the most important concepts of reproducible science and apply them.

To install the package, type :

```
pip install .
```

You can also download it directly from test Pypi with the following command :

```
pip install --extra-index-url https://test.pypi.org/simple breton_cretenet
```

# Packaging, Deployment and Licensing

## Licence :

- Permissive licence MIT

Permissions	Limitations
✓ Commercial use	✗ Liability
✓ Modification	✗ Warranty
✓ Distribution	
✓ Private use	

## Python package :

- Downloadable from test pypi
- Automatically built and uploaded **if tagged**
- Once the package is installed, you can :
  - Call the main function with : `breton_cretenet_results`
  - See all the possibilities with the main function : `breton_cretenet_results --help`

