MAPQTools Documentation

Release 0.0.1a0

Alyssa Morrow

Nov 04, 2020

Installation and Configuration

This documentation includes information and tutorials for creating MAPQTools, a python package for generating MAPQ distribution plots. This is a sample python package, and is intended to be a guide for creating and deploying python packages.

CHAPTER 1

Installing MAPQTools

1.1 Requirements

- conda
- python 3.6

1.2 Installation from Pip

1. Create and activate a pytion 3.6 conda venv:

```
conda create --name MAPQTools python=3.6 pip
source activate MAPQTools
```

2. Install MAPQTools from Pypi:

```
pip install -i https://test.pypi.org/simple/ MAPQTools
```

1.3 Installation from Source

1. Create and activate a pytion 3.6 conda venv:

```
conda create --name MAPQTools python=3.6 pip
source activate MAPQTools
```

2. Get the code:

```
git clone https://github.com/akmorrow13/CompBIO_Seminar_2020.git
cd CompBIO_Seminar_2020
```

3. Install MAPTools and its requirements

```
pip install -e .
```

CHAPTER 2

API Documentation

2.1 PlotMAPQ module

PlotMAPQ allows you to count and plot MAPQ distributions from bam files.

2.1.1 PlotMAPQHelper functions

PrintDictionaryToTab(keyHeader, valueHeader, …)	Write out dictIn to a tab-delimited file.
SaveMAPQHistogram(dictIn, filePath[, title])	Plots and saves dictIn to a histogram.

PlotMAPQ.functions.PrintDictionaryToTab

PlotMAPQ.functions.**PrintDictionaryToTab**(*keyHeader*, *valueHeader*, *dictIn*, *filePath*)

Write out dictIn to a tab-delimited file.

Args:

param keyHeader title of header for keys

param valueHeader title of header for values

param dictIn dictionary of keys and values

param filePath filePath to save results to

PlotMAPQ.functions.SaveMAPQHistogram

PlotMAPQ.functions.**SaveMAPQHistogram**(*dictIn*, *filePath*, *title='MAPQ'*)

Plots and saves dictIn to a histogram.

Args:

param dictIn dictionary of (k,v) where k indicates mapq and v indicates frequency.

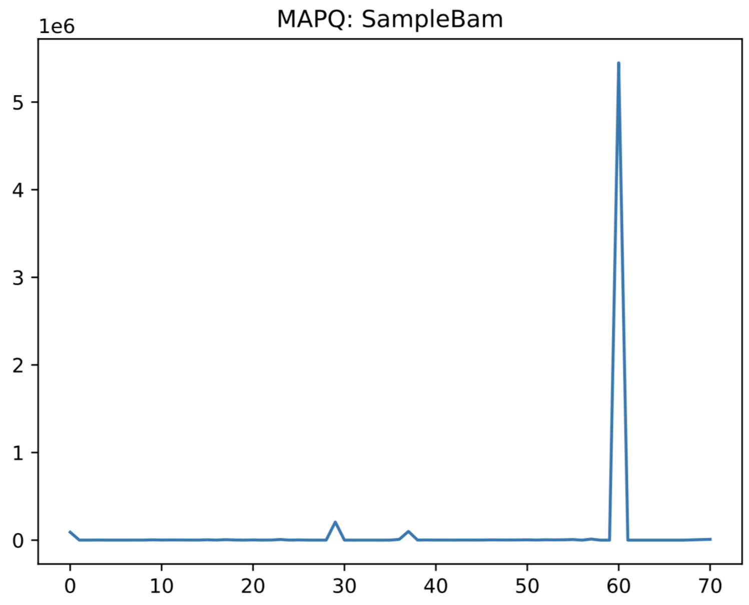
param filePath filePath to save results to

CHAPTER 3

Tutorial

This tutorial shows you how to build a python package with the minimum components for a clean and well engineered package. In this tutorial, you will learn how to setup a project, how to build and implement unit tests, how to create documentation, and how to publish your tool to bioconda and PyPI.

In this tutorial, we will build a simple project that plots mapping quality distribution from an input bam file. Below is a sample plot that our tool produces:



3.1 Prerequisites

First, we require some prerequisites before getting started with this tutorial. This tutorial assumes that you have some understanding of python, and have it installed on your computer.

3.1.1 Requirements

- conda. This is optional, but will make it easier to manage project requirements.
- python 3.6
- git
- a TestPyPI account. This is optional, but is required if you want upload a test package to TestPyPI.

3.1.2 Creating a conda environment

We use conda to manage requirements for different projects. You may be working on multiple python projects that require different packages and versions. To manage this, we can create different conda environments for each project. Although this step is not necessary, it will help you manage your projects so they do not conflict.

Let's create a conda environment for our project:

```
conda create -n PlotMAPQ python=3.6 pip
source activate PlotMAPQ
```

3.1.3 Getting the code

Next, we will want to pull the code for this tutorial. This is a finished project and includes all the code you need. You can use this repository as a reference for future projects.

Clone the repository:

```
git clone https://github.com/akmorrow13/CompBIO_Seminar_2020.git
cd CompBIO_Seminar_2020
```

You now have everything you need. Let's get started!

3.2 Project Setup

In this project, we want to build a package that allows users to plot MAPQ statistics from an alignment file. A `package` is a collection of python files that contain functions, constants, and class definitions. These python files are usually referred to as `modules`. Modules are files that contains definitions that can be imported into other modules. A file's name is the module name with the suffix .py.

In our package, we have the following items:

- a collection of modules
- build and installation scripts
- scripts to run from the command line
- unit tests
- test data
- documentation

Below, we describe these items in detail.

3.2.1 Basic Directory Structure

Let’s look at the structure of our project. Here, we have:

```
PlotMAPQ
        - __init__.py
        - tests
        - functions.py
        - PlotMAPQ.py
```

In this example, `PlotMAPQ` is our package. Within this package, we have two modules: `functions.py` and `PlotMAP`. Once we install our package, we will be able to import these modules.

Note another file `__init__.py`. The `__init__.py` file is required in package directories for python to treat directories as packages. `__init__.py` files can be empty, but can also contain initialization code for the submodule it defines. Here, we only have two `__init__.py` files, one in our package and one in our test directory. However, if your project has multiple subdirectories, you would have more.

3.2.2 Setup.py

The `setup.py` file is the script used for building and installing the package. This file uses setuptools, a library for facilitating the packaging of python projects. `setup.py` defines the package version, dependencies, metadata, and other build information. Let’s take a look at a part of our setup file:

```
import setuptools

...

setuptools.setup(
    name="MAPQTools_myname",                                    # name of project
    install_requires=REQUIRED_PACKAGES,                         # all requirements␣
↪used by this package
    version=this_version,                                       # project version,␣
↪read from version.py
    author="Author name",                                           # Author, shown␣
↪on PyPI
    scripts = ['bin/plot-mapq'],                                # command line␣
↪scripts installed
    author_email="email@email.edu",                                     # Author␣
↪email
    description="Plots MAPQ distributions from BAM files",      # Short description␣
↪of project
    long_description=long_description,                          # Long description,␣
↪shown on PyPI
    long_description_content_type="text/markdown",              # Content type. Here,␣
↪we used a markdown file.
    url="https://github.com/akmorrow13/CompBIO_Seminar_2020",    # github path
    packages=setuptools.find_packages(),                        # automatically finds␣
↪packages in the current directory. You can also explicitly list them.
    classifiers=[                                               # Classifiers give␣
↪pip metadata about your project. See https://pypi.org/classifiers/ for a list of␣
↪available classifiers.
        "Programming Language :: Python :: 3",
        "License :: OSI Approved :: Apache Software License",
        "Operating System :: OS Independent",
    ],
```

(continues on next page)

(continued from previous page)

```
    python_requires='>=3.6',                                    # python version␣
↪requirement
)
```

Note that we have also defined package requirements and a long description for PyPI in `setup.py`.

Now, we will use our setup to build a distribution. The distribution consists of a tar archive of all the files needed to build and install the package.

```
python setup.py sdist
```

This will build a distribution under `dist/` which can be uploaded to PyPI. For now, we want to play around with our project in the development mode, so we will install our project. In your project directory, install MAPQTools in editable mode:

```
pip install -e .
```

This will install our project in "editable" mode, meaning the package will be updated as we make modifications to our files.

Now that we have installed our package in editable mode, we can access the package from python:

```
python
>>> import PlotMAPQ
```

3.2.3 Scripts

Note that we also include the `scripts` directive in `setup.py`. This specifies a list of scripts that setuptools installs to be accessible by the command line. To add this functionality, we create a `bin` directory that contains scripts the user can run. We discuss this more in *Command Line Scripts*.

3.2.4 Unit tests

All good projects have unit tests. Luckily, there are a ton of great resources to add unit tests and run them regularly. We have added unit tests under `PlotMAPQ/test`. This directory contains unit tests and data required to run the unit tests. We discuss this in more detail in *Unit tests and Continuous Integration*.

3.2.5 Documentation

Under the directory `docs`, we have a bunch of files that create the documentation you are reading right now! We will discuss this in detail later in *Project Documentation*.

3.2.6 Other Important Files

In addition to the files mentioned, we have some other files that are important to a project:

1. `README.md`: the README gives information for users on github, and is also used to provide information about your package on PyPI. In this example, we read the README.md file in setup.py to provide a long description for our package. Our read me is a markdown (md) file.

2. `LICENSE`: A license tells users of your package the terms under which they can use it. choosealicense provides information for picking out a license. In this tutorial, we use the Apache Foundation license, as it provides no restrictions for users.
3. `Makefile`: A Makefile is completely optional, but makes it easier to build and develop projects. Here, we have added a makefile that contains objectives that allow developers to quickly run tests, develop, and push to PyPI.

3.2.7 Next Steps

For more information on package python projects, see python-packaging-tutorial. Next, we will take a look at our command line script.

3.3 Command Line Scripts

We would like to add a command line tool that allows users to run PlotMAPQ from the command line. As mentioned in *Project Setup*, we have included a *bin* directory that contains scripts that can be run from the command line. This allows users to call our modules from a script, as well as python. To do so, all we have to do is add a script that calls PlotMAPQ, and then declare the script in our setup script.

1. First, make a file called `bin/plot-mapq`. This file will call PlotMAPQ.
2. Next, modify the setup.py file to include your new script in the package:

```
setup(
        ...
        scripts = ['/bin/plot-mapq']
        ...
)
```

For more information on command line tools, see the python packaging documentation.

Now, we can run PlotMAPQ from the command line:

```
>> plot-mapq
usage: plot-mapq [options] alignment_bam sample_name out_directory
plot-mapq: error: the following arguments are required: bamIn, sampleName, dirOut
```

We can also just print the version:

```
>> plot-mapq -v
0.0.1a0
```

3.4 Unit tests and Continuous Integration

Testing your code is a great way to make sure your code is working as expected and that new additions to your code do not break existing functionality. The first thing we need to test our code is unit tests. Unit tests are small tests that isolate certain functionality of your code. Ideally, your unit tests comprehensively cover a majority of the functionality in your code base.

We have added unit tests in our GitHub under `PlotMAPQ/tests`. There are three unit tests that test core functions and our command line tool. These tests use unittest to run tests.

To run the tests, we simply run:

```
make test
```

This triggers pytest to run using the command:

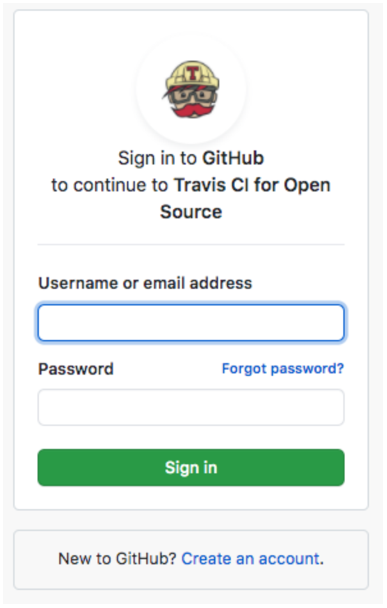
```
$(python) -m pytest -vv $(tests)
```

3.4.1 Continuous Integration with Travis

Once we have added unit tests, it would be helpful to automatically run tests every time our code changes. To do this, we can use existing continuous integration tools to test our code. Here, we choose to use travis-ci to run our tests. Travis-ci is free for public repositories, and can run unit tests every time code is updated or a pull request is made. You can find documentation for travis here.

Note that every repository that uses travis requires a `.travis.yml` file. You can find our travis file here. This file contains information on what build requirements your project has and how to run tests.

1. To use Travis, first make an account. You will have to link this with your GitHub so it can access your repositories.


Sign in to GitHub
to continue to Travis CI for Open Source

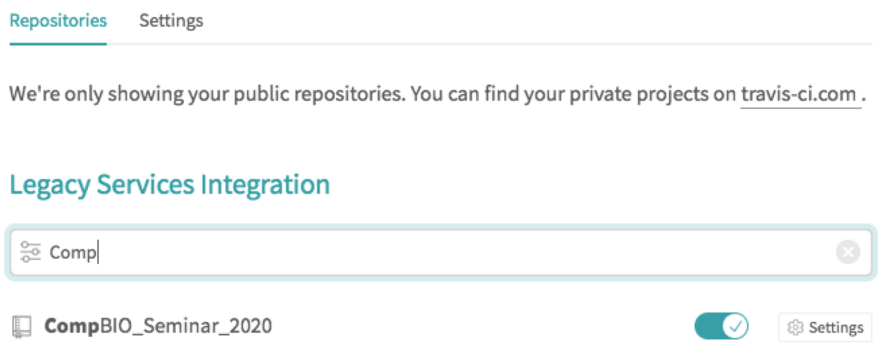
Username or email address

Password Forgot password?

Sign in

New to GitHub? Create an account.

2. Next, navigate to your user settings. You can toggle projects from your GitHub on and off.


Repositories Settings

We're only showing your public repositories. You can find your private projects on travis-ci.com .

Legacy Services Integration

Comp

CompBIO_Seminar_2020 Settings

To see our unit tests in action, navigate to our Travis tests for this project.

3.5 Uploading your package to PyPI

Our ultimate goal is to upload our package so it can be used by anyone. The Python Package Index (PyPI) is a repository of software for Python. It contains numerous packages that can all be installed in the following format:

```
pip install <package_name>
```

You can search packages in PyPI. In this section of the tutorial, we will go over how to upload projects to PyPI.

3.5.1 Building your package

We will use setuptools to build our package. To build our distribution, use the following command:

```
python setup.py sdist bdist_wheel
```

This generates the following files in the `dist` output directory:

```
dist/
        - PlotMAPQ-py3-none-any.whl
        - PlotMAPQ-0.0.1.tar.gz
```

Here, `tar.gz` is a source archive and `.whl` is a build distribution. Depending on the pip version, pip will choose one of these files to install your package.

3.5.2 Uploading to PyPI

The next step is to upload our package to PyPI so others can use it! However, this is a tutorial, so we don't really want to upload it to PyPI. To get an idea of how PyPI works, we will use test PyPI to test uploading our package.

1. First, make an account on test PyPI.
2. Verify your email address.
3. If you want to test uploading this project to PyPI, you will have to change the name of the project to be unique, because I already am using the name MAPQTools. To do so, change `name="MAPQTools"` to `name="MAPQTools-YOUR-USERNAME"` in `setup.py` to create a unique project name.

We will use twine to upload distribution packages we created in the previous section. First, let's install twine:

```
pip install twine
```

Next we can run twine to upload all files under dist:

```
python -m twine upload --repository-url https://test.pypi.org/legacy/ dist/*
```

Note: Normally, I use my Makefile to prepare and publish packages on PyPI. See the `make pypi` command in the Makefile. This command automatically builds your distribution and uploads to PyPI.

Here, enter your testPyPI username and password. Once uploaded, the package should be available at https://test.pypi.org/project/MAPQTools-YOUR-USERNAME. My package is available at https://test.pypi.org/project/MAPQTools/.

Now, you can install your package from PyPI:

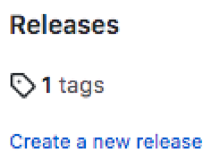
```
pip install -i https://test.pypi.org/simple/ MAPQTools-YOUR-USERNAME
```

Note: testPyPI is a great way to test publishing packages. However, when publishing a package, you will normally publish to https://pypi.org/. You can make an account for PyPI and upload to this repository for future projects.

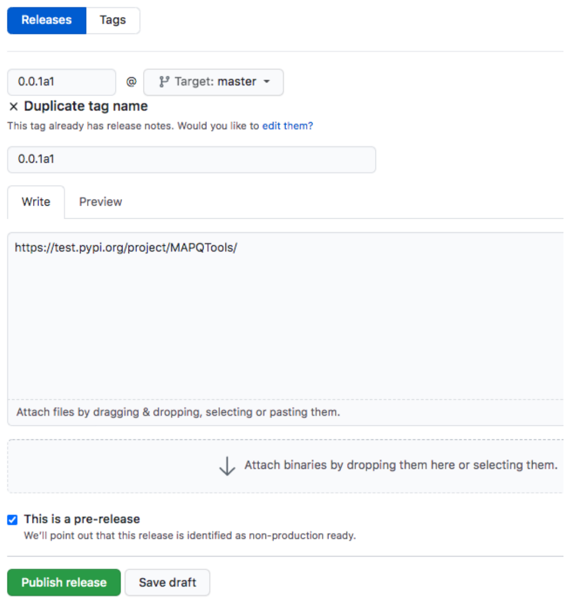
3.5.3 Tagging a release on GitHub

Now that we have uploaded to PyPI, we want to create a release. This allows us to tag our repository at a certain point in time. It also allows users who visit our GitHub to see which releases are available.

1. To tag a release, click **Create a new release** in the right sidebar on your GitHub home package.



2. Fill in information regarding the release. Note that you can check **This is a pre-release** if your release is not yet stable.



× Duplicate tag name
This tag already has release notes. Would you like to edit them?

0.0.1a1

https://test.pypi.org/project/MAPQTools/

Attach files by dragging & dropping, selecting or pasting them.

Attach binaries by dropping them here or selecting them.

☑ This is a pre-release
We'll point out that this release is identified as non-production ready.

3.6 Adding your Package to Bioconda

If PyPI can package python projects, then conda can package almost anything. Conda is great for packaging complicated projects. Let's save you have some java code, which connects to python. Additionally, you could have very complicated dependencies that are hard to manage with PyPI. We do biology, so we often use bioconda, which is a channel for the conda package manager that specializes in bioinformatics. You can browse available packages at the bioconda package index.

An additional benefit of using bioconda is that it automatically builds Docker Biocontainers that are updated everytime you update the conda recipe.

Creating a conda recipe requires you to create activation scripts and submit a pull request to the bioconda recipes. These scripts specify how to build your project and which dependencies are required.

Let's take a look at pyranges, a python package that is similar to bedtools. Go to the pyranges bioconda recipe. You will see a `meta.yaml` file. This file contains information about the package name and version, and where to download source code from. In this case, we are downloading the `tar.gz` distribution from PyPI. This file also contains information of other conda dependencies required for this conda recipe.

Once you push your conda recipe to bioconda, users will then be able to install your package with conda:

```
conda install pyranges
```

Additionally, your package will be available in biocontainers to be used with docker.

For more information about bioconda, see Contributing to Bioconda.

3.7 Project Documentation

Every good project has even better documentation! Luckily, it is now very easy to publish documentation for python projects. You can make an account at readthedocs and link a project in your Github to readthedocs.

To publish our documentation, we use Sphinx to generate python documentation. Sphinx automatically builds documentation and can also add automatic API documentation.

We have provided some documentation in the `docs/` directory. This directory contains reStructuredText (rst) files that contain documentation we would like the public to see. Additionally, there are some other files:

- `conf.py`: configuration file that is required by Sphinx. Documentation for conf.py can be found here.
- `Makefile`: This is an optional file that makes it easy to build and view your docs locally before pushing them to readthedocs.
- `requirements.txt`: This file contains python requirements needed to build docs locally.

3.7.1 Building documentation locally

First, install requirements:

```
cd docs
pip install -r requirements.txt
```

Next, you can generate the documentation to view locally:

```
make html
```

You can view the documentation locally by opening `docs/_build/index.html` in your favorite web browser.

3.7.2 Autogenerating API Documentation

Navigate to `_build/html/` and open `index.html` in your web browser to view docs locally. Here, you will see the documentation that looks just like these docs. Additionally, you will see API Documentation. We use Sphinx's autosummary feature to automatically generate documentation for our package. This also requires respective file headers in our package files (functions.py, __init__.py, and PlotMAPQ.py) so autosummary can find and generate the API documentation for the designated functions.

3.7.3 Adding documentation to readthedocs

Finally, we want to publish our documentation so it is visible to anyone who wants to use our tool.

1. Navigate to readthedocs. If you do not have an account, create one. You will have to link your Github account with your readthedocs account to access your github repositories.
2. Click `Import a Project`

3. Find the project you would like to import. Readthedocs will automatically pull docs from your `docs` directory.

You can view our documentation at readthedocs.

3.8 Summary

Congratulations! You have finished the tutorial. In summary, you have learned how to build a minimal python package with documentation and integration tests. Furthermore, you learned how to add your package to PyPI and Bioconda to be accessed by the community.

3.8.1 Resources

Below, we list a set of resources that can help you get started with future projects. Feel free to use this as a reference when building projects in the future.

- MAPQTools source code
- Making python packages and uploading to PyPI
- Getting started with bioconda
- Getting started with readthedocs
- Getting started with Travis Continous integration
- Using TestPyPI

Let's get started!

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index