

Day II - Part II - GitHub exercise

December 3, 2020

1 GitHub exercise

The final part of the workshop focuses on implementing a software package for calculating different distance measures. The exercise uses the hosting service GitHub.

This part of the workshop covers:

- Creating GitHub repositories
- Packaging and testing software
- Creating pull request
- Lincense
- Package documentation
- Continuous integration

GitHub is one hosting service but several other exists, for example:

- GitLab: <https://about.gitlab.com>

2 Log in GitHub

Initially make sure that you are logged in your GitHub account: <https://github.com>.

3 Create a repository for the project

A software repository, or “repo” for short, is a storage location for software packages.

To create a repository on GitHub click on “New” located at the left side of the home page.

Here we are going to create the project’s repository.

- **Repository name:** distances
- **Description:** A package for calculating different distance measures.
- **Private.** We are going to create a private repository as this is an artificial exercise.
- **Initialize this repository.** GitHub allows us to initialize a repository with a series of standard files. We want to include:
 - A README. A README file contains information about other files in a directory or archive of computer software.
 - A .gitignore. Select the template Python
 - A lincense. Select the MIT License.

We click `create repository` and this creates our repository.

4 Clone the repository

When we create a repository on GitHub, we create a project that exists as a remote repository. We can clone our repositories to create a local copy on our computers and sync between the two locations.

To clone a GitHub repository we need to copy it's address:

and run the following command in a terminal/command prompt:

```
$ git clone <the_address_we_just_copied>
```

Make sure that you run this command while on the location you want to clone the repository.

For example if I wanted to clone the repository on my Desktop I would run the following commands:

```
$ cd Desktop
$ git clone https://github.com/Nikoleta-v3/distances.git
```

5 Brach

When working on a project we hardly ever work on the `master` branch. Instead we create different branches for working on different parts of a project.

So let's create a branch called `implement-distances-package`.

```
$ git branch implement-distances-package
$ git checkout implement-distances-package
```

The repository's structure right now is as follows:

```
|--- .gitignore
|--- LICENSE
|--- README.md
```

Spend some time now to familiarie yourself with the structure and the current files of the repository.

6 Implement the distance function

We are now ready to write a package that calculates distances.

1. We will create a folder called `distances`. This can be done with the command:

```
$ mkdir distances
```

2. In the folder we just created we are going to add a file called `euclidean.py`. This file will contain the code needed to calculate the euclidean distance of two vectors.

Alter the file `euclidean.py` to contain the following lines of code:

```
import math

def euclidean_distance(u, v):
    """
    Computes the Euclidean distance between two vectos `u` and `v`.
```

The Euclidean distance between `u` and `v`, is defined as:

$$\sqrt{(u_1 - v_1)^2 + \dots + (u_n - v_n)^2}$$

Parameters

u : list

Input vector.

v : list

Input vector.

Returns

euclidean : double

The Euclidean distance between vectors `u` and `v`.

"""

distance = 0

for u_i, v_i in zip(u, v):

distance += (u_i - v_i) ** 2

return math.sqrt(distance)

3. Now that the function is implemented we need to commit the change.

```
$ git add distances/euclidean.py
```

```
$ git commit
```

You can use the following commit message:

```
implement euclidean distance
```

The structure of the repository should now be the following:

```
|--- distances
|   |--- euclidean.py
|--- .gitignore
|--- LICENSE
|--- README.md
```

7 Test the function

Now that we have written the function it is time to use it. Creating a test for an implemented function is a great way to:

1. Demonstrate it's usage
2. Test it's implementation

Lets's create a file `test_euclidean.py` at the root of the repository such as the structure now is:

```
|--- distances
```

```
|--- euclidean.py
|--- .gitignore
|--- LICENSE
|--- README.md
|--- test_euclidean.py
```

Open `test_euclidean.py` with your editor and alter it so that it looks like:

```
import distances

u = (2, -1)
v = (-2, 2)

print(distances.euclidean_distance(u, v))
```

Trying to run this file using the command:

```
$ python test_euclidean.py
```

should return the following error:

In order to let python know that files under the module `distances` we need to create a file called `__init__.py` which will be under `distances` and will include the following line:

```
from .euclidean import euclidean_distance
```

Now run the command:

```
$ python test_euclidean.py
```

again.

Now let's alter the code to include an `assert` command:

```
import distances

u = (2, -1)
v = (-2, 2)

assert distances.euclidean_distance(u, v) == 5
```

and run `python test_euclidean.py`.

Fantastic. Now, let's commit this change:

```
$ git add test_euclidean.py
$ git add distances/__init__.py
```

with the following commit:

```
add test for the euclidean distance
```

8 Pytest

Currently the command:

```
$ python test_euclidean.py
```

does not give any feedback. In our case that is because the `assert` command is `True` and thus there is nothing to report.

Several packages already exist that can make tests output more useful. An example of such a Python package is `pytest`. `pytest` has been installed on your computers with Anaconda.

Alter the code in `test_euclidean.py` to:

```
import distances

def test_euclidean():
    u = (2, -1)
    v = (-2, 2)

    assert distances.euclidean_distance(u, v) == 5
```

and now use the following command to run the tests:

```
$ pytest test_euclidean.py
```

This should return something like:

Now commit the changes with a message: `use the library pytest to run tests`.

The commands are:

```
$ git add test_euclidean.py
$ git commit
```

9 Document the package

When you are developing a software package, you want to make available for other people to use. Thus, you need to document your project.

This include, letting people know how to install your project, its purpose and functionality, its license and how to test it.

All these details can be included on a projects `README.md`.

The current `README.md` of our project looks like this:

10 distances

A package for calculating different distance measures.

Alter the file to include details of your project.

An example of a `README.md`:

11 distances

A package for calculating different distance measures. # Installation The project can be cloned locally using the following command:

```
$ git clone <path>
```

12 Usage

Currently the following distance measures are implemented in the package: - Euclidean distance. # Tests

The package is being tested using `pytest`. To run the test suite use the command:

```
$ pytest test_euclidean
```

13 License

The package is under the MIT license.

Once you are done altering the `README.md` remember to commit your changes.

14 Open a pull request

Our project is now ready. We have implemented a package that can calculate the euclidean distance and we want to share it with the world.

Before we update our copy on GitHub run the command:

```
$ git status
```

We can see that there are a few files that have not been committed, but also that we don't reconfig these files. You don't need to worry about these files. Let's add them to our `.gitignore`.

So our `.gitignore` should include the lines:

```
__pycache__/  
distances/__pycache__/
```

Add the changes, commit and run `git status` again.

Now that everything has been committed we are ready to update the copy of our project on GitHub.

To do that you need to run the following command:

```
$ git push origin implement-distances-package
```

and then you should see something like this:

Once you have run the command open GitHub and navigate your project.

There you should see the following:

Click on **Compare & pull request**.

GitHub then transfers you to **Pull request**. Here you can review the changes you have made and **request** for your code to become part of the projects main branch.

Take sometime to familirize yourself with the **Pull request** page and then click on **Create pull request**.

Congratulations you have created your first pull request!!!!!!

[]: