MPHYG001 - Packaging Greengraph

Thomas Hird
Student Number: 16070048

MPHYG001 - Packaging Greengraph

Author: Thomas Hird
Student Number: 16070048

1 Usage

The greengraph package produces a plot of a graph that creates a rudimentary measure of the amount of green land between two cities. This is achieved by taking satellite images from google between the two locations, over a given number of steps. The software may be installed by either running from the command line by using the syntax `>python setup.py install` or by using the python installation package pip and running:
`>pip install git+git://github.com/ucaptmh/GreenGraph`.

Once installed the software may then be run from the command line by the following:
`>greengraph [-h] --start START --end END --steps STEPS --out OUT`

The input (capitalised) arguments are:

- `START`: The starting location
- `END`: The final location
- `STEPS`: The number of steps to use between the two locations
- `OUT`: The output file for the graph.

Hence example from command line would be:
`>greengraph --start Sydney --end Melbourne --steps 20 --out GreenAus.png`

This would produce a graph png file named GreenAus.png showing the number of green pixels 20 satellite images taken in equal steps between Sydney and Melbourne.

To perform the automated tests include simply run `>nosetests` in the command line when in the main directory.

2 Problems Encountered

I encountered a number of problems throughout the project, the largest of these was centred around my unfamiliarity with testing. I specifically had issues regarding the implementation of mocking. To overcome these difficulties I made use of the course lecture notes and the online documentation regarding "mock". For example the testing the function `green_between` in the `Greengraph` class required the internet access of the `Maps` class to be considered and then appropriately mocked.

I also found devising tests themselves difficult, predominantly due to lack of exposure to them before now. Considering each method within a class and how to compare it to a desired output was a thought-provoking challenge. A more specific issue during the completion of the task was the inability to have the script be run from the terminal in the desired manner, despite it seeming to match examples given. This problem was resolved by renaming the `command.py` file `__main__.py` and adding an entry point to the set-up file. Completing the project using PyCharm, the additional features of which (auto complete, and error-highlighting to name but a few), were extremely useful.

3 Preparing Work for Release

Packaging a program takes a considerable amount of time and/or experience. It involves skills that may not be required in writing and implementing the program. The benefits of packaging however far outweigh the disadvantages.

Packaging enables software to be easily shared and used among others, and is ideally non-platform specific. The ability to easily share packages means that existing software can be used instead of users writing their own to perform the same function. The net result of this is better software being available.

Sharing of packages also allows for peer review and constructive critiquing of code. This will hopefully serve to improve the program and lead to a community of users and contributors. Sharing is further enhanced by enabling packages to be installed via package managers such as allowing users to "pip install" from, for example (as in my Greengraph package), Github. Package indexes such as PyPI allow for users to easily search and choose relevant libraries.

4 Further Considerations

Once a project is ready for release, steps can be taken to build a community of users. This may be achieved by making the package desirable. As well as making it easy to access, this may be done by keeping the library or package updated, using version control software and hosting on a sharing platform (e.g. Github). Being responsive to comments, suggestions and errors from users, gives confidence that the software is being maintained and is reliable to use. Having a community also would allow users to suggest their own solutions to bugs encountered by others.

Projects can also be made desirable to use by having a clear README file, documenting the installation and usage. A LICENSE and CITATION file make it clear under what scenarios the software may be used for. For example as well as a comprehensive README file I have licensed it under an open MIT licence allowing anyone to freely use/copy and modify the software. I have also included a number of automated test that collaborators may run and contribute to allow easy confirmation that the software is working as expected when used with test parameters.

5 File Structure

The file tree for the package is as follows:

```
GreenGraph
├── CITATION.txt
├── LICENSE.txt
├── README.md
├── setup.py
├── report.pdf
├── greengraph
│   ├── graph.py
│   ├── PlotGraph.py
│   ├── __init__.py
│   ├── __main__.py
│   └── test
│       ├── test_graph.py
│       ├── test_map.py
│       ├── __init__.py
│       └── fixtures
│           ├── build.yaml
│           ├── london_green_bool.npy
│           ├── london_numpy.npy
│           ├── london_png.png
│           ├── london_show_green.npy
│           ├── oxford_png.png
│           └── ox_london_seq.npy
└── scripts
    └── greengraph
```