MPHYG001 First Continuous Assessment: Packaging Greengraph

Gavin Dold
Student number 15081723

January 2, 2016

1 Usage and entry point

This program may be run as-is by invoking `python ./greengraph.py`, or installed with `sudo python setup.py install`, in which case on most systems it can be called simply with `greengraph`. When invoked, it generates a graph of the proportion of green pixels in a series of satellite images between two points.

By default it will use matplotlib to display a graph showing 20 data points from New York to Chicago, if given no arguments. However, the program can take 5 optional arguments: `--from`; `--to`; `--steps`; `--out`; `--format`.

The `--from` and `--to` arguments allow specification of the start and end points of the plot. These default to New York and Chicago respectively, but if for example one wanted to plot Amsterdam to Berlin this could be done with `greengraph --from Amsterdam --to Berlin`.

The `--steps` argument allows one to choose the amount of data points in the graph. This defaults to 20. For example, we can plot the greengraph from New York to Chicago in 50 steps with `greengraph --from New York --to Chicago --steps 50`

Specifying `--out` will make the program output the graph to a file instead of displaying on the screen, for example `greengraph --from New York --to Chicago --out Greengraph.png` will save to `Greengraph.png`. Valid output formats depend on the backend used by each system, but most backends support png, pdf, ps, eps and svg. If a valid file extension is found it will output in that format, otherwise it will default to png and append `.png` to the end (i.e. `--out Greengraph` creates the file `Greengraph.png`).

One can force an output format with `--format`. As detailed above this is system-dependent but most backends support png, pdf, ps, eps, and svg, and no checks are performed to check the validity of the format.

Finally, the program prints a help message if called with `-h` or `--help`.

2 Problems encountered

I had some issues with Google's API limiting the number of API calls I could make in a certain period of time. In this circumstance instead of receiving a 400x400 satellite image, the program receives an image indicating API calls are being made too fast, resulting in garbage data being displayed without the program noticing.

I got around this by registering for a Google API key, which I have included in the program code in `map.py`. This prevented any further limits from affecting my running of the program, though if it were implemented more widely one single API key may not be sufficient and users may be required to obtain their own.

I had difficulty in writing a test for the `Map.show_green()` method given in the original code as it does not pass `self` as an argument. However, I quickly realised that this method is not in fact used at any point in the program – hence it has been commented out in `map.py`.

3 Discussion of packaging

The major advantage in packaging code for installation through package managers is the vast simplification of the installation process, and the greater attention your program would get from being part of a package index. Without proper packaging it becomes more of a pain to install software and resolve dependencies, putting users off your program. Packaging also removes the need for the developer to ensure their install code works on every possible OS and system, by allowing the user's package manager to install the software in a platform-dependent way, while the published code remains platform-independent.

For a small software package like this the cost of packaging is minimal as it only requires the creation of a few folders and `__init__.py` files, and writing `setup.py` according to a standard template. However, as software packages get older, larger, and more developed, the effort in maintaining the code in a platform-independent manner and keeping track of dependencies sensibly can become a significant overhead.

Once prepared properly, packages can be installed through package managers such as pip, allowing for automated and (usually) hassle-free installation. These package managers connect to a database of software (a package index) to allow software to be installed by simply giving its name `pip install packagename`. Pip is also capable of installing from a git repository, if the repository contains packaged code, with `pip install git+git://github.com/user/repository`.

Pip's package index is PyPI which contains over 72000 packages. Uploading software to PyPI would allow your program to be installed by users simply with `pip install greengraph`. Getting packages indexed on PyPI is relatively simple, involving registering an account and uploading your package.

4 Towards a community

The major step towards building a community around open source software such as this is getting it noticed. Packaging it for easy installation through pip and such package managers is vital for getting users, as very few people would be willing to install packages manually. Other steps toward gaining users would depend on the project in question, but could involve posting on forums, talking to colleagues, or for scientific work bringing it up at conferences.

Once a userbase has been established, having a public github repository will be a major step toward making it a community, as having forums and a bugtracker encourages people to discuss the issues and propose solutions without necessarily needing input from the project's founder.