📦 Python packages

Lucas Borges Ferreira

2024-05-17

Some challenges in Python projects 🐍

- How to structure my project?
  - A single file?
  - Multiple files?
  - Multiple directories?
- How to manage dependencies?
- How to make my code available to others?

Spoiler: Use packages! 📦

Why do I need a 📦?

- Efficient way to structure your code
- Easier to create code that can be easily imported and used anywhere in your system
- Packages can be easily shared and installed using `pip`
- Support for automatic dependency management
- Good fit for geospatial projects

What is a 📦?

- A 📦 is basically a collection of modules and subpackages

General structure:

```
└───mypackage/                  # Top-level package
    │  __init__.py              # Initialize the package
    │  mymodule.py              # Module
    │  mymodule2.py             # Module
    │   └───mysubpackage/       # Subpackage
    │       │  __init__.py      # Initialize the subpackage
    │       │  mysubmodule.py   # Module
```

How to use a 📦?

- To use a 📦, you need to import it!

Based on the previous example, you could import a module from the package like this:

```python
from mypackage import mymodule
# Call a function from the module
mymodule.my_function()
```

To import a module from a subpackage:

```python
from mypackage.mysubpackage import mysubmodule
# Call a function from the module
mysubmodule.my_function()
```

How can a 📦 help me?

> Imagine if you have to read a set of satellite images from a directory (you must support multiple file formats), preprocess the data (different preprocessing algorithm must be provided), and offer tools to visualize and export the data...

> You could write a single python file that does all of this, but it would be a mess. Instead, you could create a package that contains a module or subpackage for each of these task categories.

Example

```
└───mygeopackage/
    │   __init__.py
    │   data_loader.py
    │   export.py
    │   plot.py
    │     └───preprocessing/
    │         │   __init__.py
    │         │   cloud_masking.py
    │         │   smoothing.py
    │         │   mosaicing.py
```

```python
from mygeopackage import data_loader
from mygeopackage.preprocessing import cloud_masking, smoothing
from mygeopackage import export

# Load images
images = data_loader.load_geotif_images('path/to/images')
# Preprocess images
images = cloud_masking.mask_clouds(images)
images = smoothing.smooth_images(images)
# Export images
export.export_to_geotif(images, 'path/to/export')
```

Intra-package references

- Sometimes a module needs to import another module. You can do this by using relative imports:

```
└───mypackage/
    │   __init__.py
    │   mymodule.py
    │   mymodule2.py
    │   └───mysubpackage/
    │       │   __init__.py
    │       │   mysubmodule.py
```

```python
# Inside mymodule.py import mymodule2.py
from . import mymodule2

# Inside mymodule.py import mysubmodule.py
from .mysubpackage import mysubmodule

# Inside mysubmodule.py import mymodule.py
from .. import mymodule
```

Packaging Python projects

- So far you learned the core structure of a package, but to “package” a project you also need some configuration files
- Once you have the required files, you can install your package using `pip`
- `pip` is the package installer for Python. It can install packages from PyPI (Python Package Index) or other sources

Python project example

- In the link below, you can find an example of a geospatial project structured as a package

Example project (Click to view)

Installing a 📦 using `pip`

- Once installed, you can import the package from anywhere in your system
- To install, inside the project directory, run:

```
pip install .
```

- To install in editable mode (changes in the code will be reflected in the installed package), run the command below. This is ideal for development.

```
pip install -e .
```

How to publish a 📦

- ~~To share a package, you can use USPS, FedEx, UPS…~~
- You can use PyPI (Python Package Index), conda-forge, or just install the package directly from GitHub.
- PyPI and conda-forge are great for general distribution of your package, while GitHub is good for sharing with a specific audience (e.g., a package for internal use of GCER members).

Practical example

Let’s download this github repository, and…

- Remove current modules/subpackages and create a single module with an example function
- Adjust `pyproject.toml`
- Create a new conda environment
- Intall the package using `pip`

Practical example

Example module with a function that plots random data

```python
# Inside a module called my_module.py
import numpy as np
import matplotlib.pyplot as plt

def plot_random_data():
    x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
    y = x**2
    plt.plot(x, y)
    plt.show()
```

Practical example

- Create a notebook inside the notebook folder and try to use the function previously created.
- It’s not possible without installing the package first!
- Create and activate a new conda env:

```
conda create --name gcer_training python=3.10
conda activate gcer_training
```

- Intall the package using `pip`:
  - pip install .
  - pip install -e .

Managing conda dependencies

- Some dependencies need to be installed using conda
- Rasterio is typically easier to install using conda (Windows)
- The easiest way to handle it is requesting the user to install conda dependencies manually before installing the main package: `conda install -c conda-forge rasterio`
- An `environment.yml` file can be used to create a conda environment with all dependencies and also to install the package using `pip`

Managing conda dependencies

- What is the `environment.yml` file?
- Check example
- Creating a conda env using the `environment.yml` file:

  ```
  conda env create -f environment.yml
  ```

Intalling the 📦 from GitHub

- To install the package directly from GitHub, use something like:

```
pip install git+https://github.com/user_name/repo_name
```

Example:

```
pip install git+https://github.com/lbferreira/geospatial_project_ex
```

Publishing the package to PyPI

- Publishing to PyPI is generally simple if you have the package structure and configuration files properly prepared
- Tutorial here

Package structure in GitHub repositories

Getting familiar with the package structure is useful to understand other packages on GitHub. Examples:

- segment-geospatial
- Xee
- geopandas

Keep in mind that we learned the core structure of a package/repository. Other files/variations depends on the project requirements and complexity.

Hints for organizing your code inside the package

- A well organized code makes your package easier to understand and maintain
- Use meaningful names for your modules, functions, and variables
- Create functions and classes with a single responsibility / Avoid large functions/classes
- Use type hints and docstrings

Hints for organizing your code inside the package

- Do not place specific configurations inside the package, this should be handled by the user who is calling the functions and classes
- When performing scientific experiments, you can use notebooks or scripts outside the package

Geospatial project example

- Let’s analyze an example of a code initially organized in a single file and then refactored into a package
- Example here

Useful links

- Packages
- Packaging Python Projects

Thank you!