[PD-05-011] Python for GIS

Abstract

Python is a popular language for geospatial programming and application development. This entry provides an overview of the different development modes that can be adopted for GIS programming with Python and discusses the history of Python adoption in the GIS community. The different layers of the geospatial development stack in Python are examined giving the reader an understanding of the breadth that Python offers to the GIS developer. Future developments and broader issues related to interoperability and programming ecosystems are identified.

Keywords: development, geocomputation, geoprocessing, open source, spatial analysis, workflow

Author & citation

Rey, S.J. (2017). Python for GIS. The Geographic Information Science & Technology Body of Knowledge (3rd Quarter 2017 Edition), John P. Wilson (ed). DOI: 10.22224/gistbok/2017.3.4

Explanation

1. Definitions
2. Python for GIS
3. Geospatial development modes with Python
4. The Python geospatial development stack
5. Future developments

1. Definitions

development stack: a group of software programs that work together to achieve a specific result or to carry out a particular analytical task

interpreted language: a programming language in which its instructions are executed directly without previous compilation into machine-language instructions

open source software: software code that is made freely available and may be modified and redistributed

2. Python for GIS

Python is an open-source, object oriented interpreted language created by Guido von Rossum in 1991. Since its release, Python has seen widespread adoption across many problem domains. This popularity arises from the attractive features of the language. First, as a multi-paradigm language Python supports both structured programming and object orientation. Second, Python is an interpreted language which lends itself to rapid

prototyping and development cycles. Core Python is itself written in the C-language and this has enabled implementations of Python across many platforms and operating systems. The rich and versatile standard library makes Python well suited for many projects.

GIScience is one of the many scientific disciplines where Python has found a receptive audience. From a scientific perspective, the emphasis on readability, cross-platform support, and low startup costs have made Python an excellent language for collaboration between GIScientists. Some of these same features also position Python as a wonderful language for teaching GIScience concepts, indeed Python is increasingly used as a first language in computer science curricula. Python is an excellent first-language, however, this should not be taken to mean it is limited in scope, as many large scale and widely used applications have been implemented in Python, a prime example being Dropbox.

Given its growing popularity, this entry first provides an overview of the different development modes that can be adopted for GIS programming with Python and discusses the history of Python adoption in the GIS community. This is followed by a canvassing of the GIS programming stack to give the reader an understanding of the breadth that Python offers to the GIS developer. The conclusion discusses likely future developments and broader issues related to interoperability and programming ecosystems.

3. Geospatial development modes with Python

As a scripting language Python offers flexibility in the different modes of development for geospatial programming. These can take the form of using Python to extend the functionality of a desktop GIS, to the development of a complete standalone desktop application for specialized geospatial analysis, and the use of Python for scientific scripting and interactive computation in a shell environment.

3.1 Desktop applications

Python plays a central role for desktop Geospatial applications. From version 9.0 of ArcGIS, Esri included Python as a core scripting language. Its **ArcPy** package provides an interface to geoprocessing tools, functions, classes, and modules. An arcpy function defines a particular piece of functionality, for example, functions exist to list certain datasets, access a dataset's properties and update geodatabases. Moreover, Python scripts can be used within ArcGIS or from outside of ArcGIS, and additional Python toolboxes can be developed and distributed to ArcGIS users by leveraging Python's **distutils** module.

The open source GIS package QGIS uses Python as a scripting language in a variety of ways. A Python console is available through the QGIS graphical user interface (GUI) to provide an interactive shell that can be used to script the existing session. This supports experimentation with the QGIS API and allows the user to build up a workflow that ultimately can be saved as a script for later reuse.

Python has also been used to develop the **processing** framework as a part of QGIS. This is a geoprocessing environment that can be employed to run native or external/third-party algorithms from within QGIS. Figure 1 shows an example of using the PySAL tool from within the Processing toolbox to carry out a hot-spot analysis of a resource deprivation variable in 1990 for southern US counties.

Figure 1. PySAL within QGIS Processing Toolbox: Hot-spot analysis of Homicide Rates in Southern US Counties.

In addition to these two desktop GIS packages, Python has been used to develop other standalone geospatial applications. Prime examples are related to the PySAL project and include: GeoDaSpace, a package for spatial regression analysis; CAST: Crime Analytics in Space-Time; and STARS (Space-Time Analysis of Regional Systems). These packages are developed using Python and wrap advanced geospatial functionality inside a GUI. These applications are possible due to the existence of multiple scientific libraries available for Python that are summarized below.

3.2 Interative computational geospatial programming

Many times in scientific computing, researchers write code to express their ideas, prototype, or to explore data. This sort of opportunistic programming requires a flexible computing environment to facilitate open-ended exploration. The Python ecosystem offers a rich set of tools for this type of interactive scientific computing.

IPython expands upon the interactive Python interpreter to provide a comprehensive shell environment for interactive and exploratory computing. Feature-rich, IPython offers tab-completion, object introspection, command history and access to the operating system shell resulting in a superior interpreter for scientific computing over that offered by the built-in Python interpreter.

The IPython project has evolved and given rise to Jupyter**.** The Jupyter notebook extends the console-based approach to interactive computing by providing a web-based application that can encompass the entire scientific computing process from code development, documentation, and execution to the actual presentation and dissemination of the results. Jupyter consists of two components. The first is a browser-based tool that combines text, code, and rich media to support interactive authoring of scientific documents. The second component is the notebook document which is the actual encoding of all the content that is visible in the web-application. The notebook documents are implemented as JavaScript Object Notation (JSON) which facilitates version control and sharing with collaborators.

4. The Python Geospatial Development Stack

Geospatial development covers a wide diversity of tasks. To provide an overview of the geospatial programming stack in Python, Table 1 summarizes selective packages from each layer in the stack. The selected packages are the more popular or commonly encountered for each type of functionality, and it should be kept in mind that the listing is not exhaustive in its coverage but intended to provide a first-stop pointer for Python developers looking to tackle a particular task.

Layer	Package	Description	WWW
Spatial Data IO	gdal	Interface to Geospatial data abstraction library	https://pypi.python.org/pypi/GDAL/

	fiona	API to OGR (Vector) layer of GDAL	http://toblerity.org/fiona/
	rasterio	Reading and writing geospatial raster data	https://github.com/mapbox/rasterio
Geoprocessing	shapely	Deterministic spatial analysis	http://toblerity.org/shapely
	rasterstats	Summarizing rasters using vector geometries	https://github.com/perrygeo/python-rasterstats
	geopandas	Pandas-like spatial operations on geometric types	http://geopandas.org
	pyproj	PROJ4 Interface for cartographic transformations	https://github.com/jswhit/pyproj
Geovisualization	basemap	Plotting 2D data on maps	https://github.com/matplotlib/basemap
	cartopy	Cartographic tools	http://scitools.org.uk/cartopy
	folium	Visualization via interactive Leaflet maps	https://github.com/python-visualization/folium
	bokeh	Interactive visualization library browsers	https://github.com/bokeh
	datashader	Big data visualization graphics pipeline	https://github.com/bokeh/datashader
Spatial Statistical Analysis	PySAL	Spatial data analysis	http://pysal.org
	pykriging	Geostatistics	http://pykriging.com/
Spatial Modeling	mesa	Agent based modeling	https://github.com/projectmesa
	spint	Spatial interaction modeling	https://github.com/pysal/pysal/tree/master/pysal/contrib/spint
	clusterpy	Spatially constrained clustering	http://www.rise-group.org/zonificando-la-ciudad-por-perfil-de-clientes-copy/
Web and Distributed	OWSlib	Client programming interface to OGC web service	https://geopython.github.io/OWSLib/
	Stetl	Streaming ETL for geospatial data	http://www.stetl.org/en/latest/

Table 1. The Python Geospatial Programming Stack

4.1 Spatial Data Input/Output

All spatial analysis begins with the reading of geospatial data. One of the key distinguishing features of doing spatial data analysis is the rich variety of spatial data formats. This presents a major challenge to the geospatial developer, as one-size-fits-all packages are simply not to be found.

As is common for many of the packages in the geospatial stack, the approach taken to implement file input (and output) has been to wrap existing C libraries that have been widely used for the purpose at hand. In this case the target library is the Geospatial Data Abstraction Library (GDAL) that provides translators to read and write both raster and vector spatial data. In the Python stack, two different libraries have been developed that target these components. **Fiona** focuses on the OGR (vector) functionality, while **rasterio** provides a similar wrapper to expose the raster functionality of GDAL. This allows for the reading and writing of formats such as GeoTIFF via a Python API that relies on **numpy** N-D arrays for efficient computation.

4.2 Geoprocessing

Once geospatial data is read into memory, a variety of geometric operations and manipulations are available for subsequent processing. For vector data these take the form of set theoretic operations and manipulation of planar features. The package **Shapely** wraps the **geos** library for buffering, intersection, dilation, differencing and a host of other types of spatial operators on vector objects. Functionality for focal, zonal, and summarization of rasters is offered by the **rasterstats** package. **Rasterstats** can also support the querying of rasters using vector geometries. For example a developer can find the elevation from a DEM stored in a TIF file based on a point vector object, or generate summary statistics for elevation within the bounds of a polygon object (i.e., mean, max, std of elevation within a zone).

Both sets of these spatial operations are commonly encountered in the use case of deriving new spatial variables. These are then utilized higher up in the geospatial stack for statistical analysis. On their own, the packages at the geoprocessing stack (i.e., fiona and shapely) are not intended to cover the subsequent statistical analysis of the derived objects/layers. Rather, they are viewed as critical components of a geospatial pipeline in which different packages are chained together to implement a particular workflow. The package **geopandas** can be viewed as a way to facilitate this type of workflow and pipeline. It relies on functionality from **fiona** and **shapely** for geoprocessing, but draws inspiration from the popular general data manipulation package **pandas** which is the premier library for processing data in the Python computational stack.

In order for data from different formats and sources to be properly harmonized and integrated, functionality to convert between different coordinate reference systems is provided through the **pyproj** package. **pyproj** is a **Cython** wrapper that provides a Python interface to the **PROJ.4** library for cartographic transformations and geodetic computations.

4.3 Geovisualization and Mapping

The visualization and mapping of geospatial data in Python had its origins in global scale mapping implemented in the package **basemap**. **Basemap** does not implement the actual plotting per se but relies on **Proj4.C** to transform coordinates to a specific map projection and then employs **matplotlib**, the primary visualization library in Python, to do the actual plotting of contours, images, or vector objects in the projected coordinates. **Basemap's** origins were in the support of oceanography and meteorology, and over time its functionality has evolved to support other disciplines from biology to geology and geophysics.

cartopy is a package targeted at easy drawing of maps for data analysis and visualization. It adopts an object oriented approach to defining map projections and provides a simple and intuitive interface to visualization via **matplotlib**. **cartopy** relies on both PROJ.4 for its projection functionality, and **shapely** for reading shapefiles. Similar to **basemap, Cartopy** was originally created to support meteorological research, but has expanded to support mapping in a wide array of scientific domains.

While both **basemap** and **cartopy** interface with **matplotlib** for visualization, a number of Python mapping packages target web browsers as the platform for visualization. **Folium** is one of the earliest of these, providing a Python interface to the **leaflet** javascript library for interactive web-based mapping. **Folium** also includes enhancements from the

Vincent/Vega framework to bring augmented markers and plots into the geospatial visualization. Folium interfaces with a number of popular tileset services including **OpenStreetMap**, **Mapbox**, and **Stamen,** with support for **GeoJSON** and **TopoJSON** spatial data formats.

Although they are not designed specifically for mapping and geovisualzation two recent Python visualization packages warrant inclusion here. **Bokeh** is similar to **Folium** in that it targets modern web browsers for visualization. It does so through the style of the **D3.js** library for data driven visualization, with an emphasis on high performance interactivity for large data sets. **Datashader** is a component of **Bokeh** that implements a graphics pipeline for rendering representations of massive datasets through binning, aggregation and transformations. It is notable that the impressive capabilities of **Datashader** often feature geospatial applications (https://github.com/bokeh/datashader).

Before leaving the visualization layer of the Python geospatial stack, two additional points need to be made. First, there is an active focus in the Python scientific community on effective color maps for scientific visualization. This work intersects with that of cartographers designing color schemes for choropleth mapping. In the Python world the popular packages are **palettable** and **colormap**. Second, the visualization component of the Python scientific stack is rapidly evolving so readers are encouraged to use this section as an entry point not a final destination.

4.4 Spatial Statistical Analysis

PySAL is a library of spatial analysis functions for spatial data. It consists of modules that cover exploratory spatial data analysis which includes global and local measures of spatial autocorrelation - popular methods to detect spatial clustering and hot spots. **PySAL** also provides functionality for spatial regression, space-time analysis, regionalization, map classification and a host of geocomputational modules. **PySAL** began at a time when very little of the geospatial Python stack existed and, as such, it also includes functionality for lower levels of the stack out of necessity. Over time, **PySAL** has added modules that interface with many other newer packages in the stack (shapely, geopandas, cartopy, etc).

PySAL focuses primarily on vector spatial formats and covers statistical analysis for attributes associated with polygons, points and networks. Geostatistical data analysis is associated with the statistical analysis of fields commonly encountered in the geosciences where phenomena such as temperature, precipitation, air quality are modeled as a surface using methods such as kriging and various spatial interpolation approaches. The objective in much of this work is to generate predictions for a continuum of locations based on a discrete set of observed locations for the process at hand. In the Python stack, geostatistics is an area that has been underdeveloped to date as there have been a few starts of projects but intermittent development - one example being **pykriging**.

4.5 Spatial Modeling

Mesa is a package that implements functionality to develop and apply agent-based models. These models can be used to simulate autonomous agents embodied with behavioral rules on a grid layout to assess the impacts of their behaviors and interactions on the system as a whole. The classic example of ABM is the Schelling model of segregation featured as a key demonstration in the Mesa package.

clusterpy provides a library of spatially constrained clustering algorithms that can be used to group primitive geographic areas or points into a smaller number of regions for subsequent analysis. For example, in geodemographics, census tracts can be defined based on both socioeconomic similarity as well as geographical contiguity. The inclusion of the latter constraint differentiates regionalization from more general multivariate clustering.

4.6 Web and Distributed

Increasingly, scientific workflows have been moving towards web based and distributed frameworks, and spatial analysis is no exception. Here libraries that provide for middleware linking different analytical and processing functionality together as individual services are required. In the Python geospatial stack a growing number of packages implement specifications from the Open Geospatial Consortium to facilitate web based geoprocessing and analysis. **Stetl** offers a toolkit for extraction, transformation, and loading (ETL) of geospatial data. **OWSlib** implements key OGC standards via a Pythonic interface including, web mapping service (WMS), web feature service (WFS), sensor model language (SensorML), and WaterML among some 17 other services. Both **Stetl** and **OWSlib** are part of the **GeoPython** organization which develops numerous geospatial projects.

4.7 Package Considerations

The packages listed in Table 1 represent only a fraction of what is available to the geospatial developers working in Python. It should also be kept in mind that the placing of a package in a specific layer is somewhat fluid since a particular package may afford functionality that spans multiple layers of the stack. For example, because **Geopandas** has fiona and **PySAL** among its dependencies, it can provide access to the core functionality of those packages via its namespace. The richness and diversity of the geospatial packages are testaments to the core strengths of the underlying scientific computing stack in Python where packages such as **numpy** and **scipy** together with the development affordances of the Python ecosystem itself provide for an excellent development paradigm. As well, there are packages in the broader Python scientific stack that can be called upon to implement geospatial functionality. For example, **network-x** provides a wealth of methods for analyzing networks and graphs, while scikit-image is a powerful library for image analysis.

There are a few issues that GIS developers need to keep in mind, particularly when developing GIS applications with Python for different platforms. A key issue is the management of the Python installation alongside third-party packages. Both **ArcGIS** and **QGIS** install their own Python interpreters in directories that are specific to those applications. This can be a source of confusion if the developer also has other Python installations on the same machine, particularly when it comes to installing additional packages. Python offers a number of approaches towards package management (**pip** being the main one), and Python distributions such as Anaconda and Enthought also provide their own package managers.

5. Future Developments

Python and the related GIS software ecosystem are very active and undergoing constant growth. To enhance one’s proficiency, it is important to keep up with these developments so that workflows can remain current by taking advantage of advances in the language

itself and packages in the broader ecosystem. While predictions about the future are always challenging, the seeds of a few developments can already be seen emerging. The first is an increasing emphasis on the importance of interoperability between Python and other languages. Packages such as **RPy** allow for bidirectional interfaces between Python and R so that a developer can leverage the strengths of each language in the same workflow. This also serves to widen the scope of the GIScience programming stack.

A related area to keep abreast of is the set of tools that allow for extension of Python by wrapping external libraries written in lower level languages such as C, C++, and Fortran. **Cython**, **numba**, and **swig** provide methods to target bottlenecks in a Python code base to be replaced with more computationally efficient libraries. The extensibility of Python using these tools means that initial prototyping of a GIScience application can rely on pure Python in a developer-efficient manner. Only after the full functionality of the application is implemented should attention turn to optimizations and, in many cases, this turns out to be a small fraction of the code base. In these instances, the extension frameworks can prove invaluable.

The world of parallel and distributed high performance computing has advanced remarkably in the past decade and the Python ecosystem exemplifies this. Although a comprehensive coverage of related packages is beyond the current scope (packages for cloud, distributed and high performance computing in Python can be found at https://wiki.python.org/moin/ParallelProcessing), one exemplar package to note is **Dask.** A flexible parallel library for analytic computing, Dask is built around two components: dynamic task scheduling that optimizes interactive computational workloads, and big data collections that enable out-of-memory or distributed processing of parallel data arrays and dataframes.

A fourth important thread is the rise of open science. Stressing the need for improved transparency, reproducibility, and replication, the open science movement is leading to new types of development paradigms such as Docker containers, conda environments and the notebook viewer that enable sharing of snippets of code and even complete workflows that reproduce the analysis underlying a particular scientific paper. Familiarity with these tools will be increasingly important for Python GIS development.

Open science also has parallels in the open government and open data movements that are driving transparency, sharing, and reuse of government collected data. These have fostered collaborative projects bringing together academic researchers, government and industry to fashion computing solutions targeting social and environmental issues. Given that many of these projects involve geospatial data and analysis, Python capable GIScientists can play a central role in these efforts.