Journal of Geographical Systems
https://doi.org/10.1007/s10109-021-00346-6

ORIGINAL ARTICLE

Geographical Python Teaching Resources: GEOPYTER

Jonathan Reades[1] · Sergio J. Rey[2]

Received: 19 October 2019 / Accepted: 8 January 2021

Abstract
GEOPYTER, an acronym of Geographical Python Teaching Resources, provides a hub for the distribution of 'best practice' in computational and spatial analytic instruction, enabling instructors to quickly and flexibly remix contributed content to suit their needs and delivery framework and encouraging contributors from around the world to 'give back' whether in terms of how to teach individual concepts or deliver whole courses. As such, GEOPYTER is positioned at the confluence of two powerful streams of thought in software and education: the free and open-source software movement in which contributors help to build better software, usually on an unpaid basis, in return for having access to better tools and the recognition of their peers); and the rise of Massive Open Online Courses, which seek to radically expand access to education by moving course content online and providing access to students anywhere in the world at little or no cost. This paper sets out in greater detail the origins and inspiration for GEOPYTER, the design of the system and, through examples, the types of innovative workflows that it enables for teachers. We believe that tools like GEOPYTER, which build on open teaching practices and promote the development of a shared understanding of what it is to *be* a computational geographer represent an opportunity to expand the impact of this second wave of innovation in instruction while reducing the demands placed on those actively teaching in this area.

Keywords Open Source · Spatial Analysis · Education

JEL Classification R3

✉ Sergio J. Rey
sergio.rey@ucr.edu

[1] University College London, London, UK

[2] University of California, Riverside, CA, USA

1 Introduction

Although Donoho (2017) traces the origins of data science back to Tukey's *The future of data analysis* (1962), it is only recently that this set of previously disparate practices for working with large data sets has begun to be formalised in a way that might allow it to be taught in a university context instead of acquired on the job through 'learning-by-doing'. One of the most profound practical impacts of this development has been on the ways in which developers share and execute code: although systems for combining code, commentary, and results have been around for some time (e.g. R-Markdown), these were intended primarily for replicating research outputs, with teaching and interaction as secondary considerations. The emergence of data science—and the overall pace of change in the tools and programming libraries that it employs—gave new impetus to the search for lightweight ways of sharing code and documentation, and an interactive browser-based platform called Jupyter (Kluyver et al. 2016) was the result.

University educators in many disciplines are playing catch-up, but geography—thanks to its long relationship with computation (Arribas-Bel and Reades 2018)—has been acutely aware of these developments: changes in the volume and extent of spatial data available (e.g. Graham and Shelton 2013; González-Bailón 2013; Reades et al. 2016), and in the number and range of methods employed to identify patterns in those data (e.g. Fan et al. 2016; Naik et al. 2017; Santibanez et al. 2015; Stevens et al. 2015; Arribas-Bel et al. 2017), are allowing us to tackle questions thought unanswerable just a few years ago. In academia, there has been a 'turn' towards coding (e.g. Brunsdon and Comber 2020), but this has not been matched by changes in our teaching practice and there is evidence of a 'hollowing out' of once domain-specific skills (Singleton 2014; Singleton et al. 2016): most of what used to be done with specialist Geographic Information Systems (GIS) software can now be done using free online resources (e.g. Google's Fusion Tables, MapBox, Carto, ArcOnline). However, most countries have actually seen an increase in 'geospatial jobs' (Solís et al. 2020) indicating unmet demand for graduates who can *code* using the latest (geo)algorithms and techniques. In this article, we present one means of bringing the benefits of (spatial) data science approaches to instruction using Jupyter 'notebooks' for the teaching of the Python programming language and geospatial analysis.

2 Statement of need

Regardless of whether this is a revolution (Wyly 2014; Torrens 2010) or an evolution (Barnes 2013, 2014), there is no question that teaching students and researchers how to code and how to *think* computationally (Barba 2015) presents new challenges (Etherington 2016; Muller and Kidd 2014; Rey 2009) and that there is little in the way of a tradition of doing so to a high level within geography (see surveys in Bowlick et al. 2017; Bowlick and Wright 2018). Consequently, the majority of spatial data science and analytics resources used in the classroom are being developed from scratch at each institution, often by newly appointed lecturers and assistant professors coming from the relatively small number of elite research facilities active in this domain (see discussion in Ley et al. 2013). Not only has this led to the duplication of effort at multiple sites, but it also has an enormous impact on the productivity of early career researchers, many of whom are in their first teaching post.

The overarching purpose of this software is therefore to address two seemingly contradictory issues: the recognition that no two instructors teach in precisely the same way and the fact that few instructors have the luxury of both time and tenure to develop compelling new course material in splendid isolation. This is where the Geographical Python Teaching Resource (GEOPYTER) comes into play: although developed with geographers in mind, it provides a generic means by which instructors in any discipline can selectively incorporate and remix programming and conceptual content from existing Jupyter notebooks while providing their own 'gloss' for this content. Our approach therefore seeks to develop a rich community of developers and teachers who work *together* on new teaching materials, who make their output entirely *open* to the wider community, and who *support* instructors in quickly adapting existing materials to new delivery formats. It should also be noted that our tool is fully compatible with notebooks written in other languages, such as R and its powerful geospatial analytics framework (see Bivand 2020 for an overview).

3 Origins and inspirations

Like many open-source projects, GEOPYTER grew out of a need to scratch an itch: in our roles as teachers of (geo)computational concepts and methods, we grappled on an annual basis with the demands of developing and updating instructional material with complex interdependencies. And, while the fundamental computing and analytic concepts may remain fairly stable, the field is highly dynamic in terms of both the content (i.e. what should be in the curriculum) and the code (i.e. how applications should be taught). Although, in principle, there should be little interaction between curriculum and code, in practice changes to software libraries can make certain connections much harder, or much easier, to establish in the mind of the learner. For instance, until the widespread adoption of `geopandas` in Python (which itself depended on the widespread adoption of `pandas`) it was common to find that each spatial analysis library had its own data structures[1] and, consequently, students needed to be taught to work with spatial data in different ways depending on the library. Consolidation around a single approach freed up a lot of 'space' in the curriculum for teaching content, not format.

In spite of this, every time a widely used library is updated a protracted sequence of revisions ensues as teachers need to decide whether to resist making use of, or even alerting students to, the new features in order to save updating their materials, or whether to leap in enthusiastically only to struggle with conflicting dependencies and unstable feature sets. But on the basis that 'many hands makes light work', we initially explored with other faculty the potential to 'divide and conquer': we would

[1] A similar effect could be seen in `R` with the `tidyverse` and `sftools`.

each focus on different topics using the latest codebase and then combine these contributions to create a full course. Though received with some enthusiasm, this plan did not survive contact with reality: first, when actually faced with the uncertain promise of receiving a full course in return for our individual efforts, many of us felt more secure developing our own material; second, we had underestimated the value that we each attach to the ability to personalise material to our own style of teaching. Since the shared materials were being developed according to individual 'style', any rationalisation or personalisation would have to be done by the instructor at a later date.

Going back to the drawing board, we looked again at our own experiences of successful programming and instructional efforts. In particular, we took note of two powerful streams of thought in software and education with which we already familiar: the importance attached by the free and open-source software (FOSS) movement to both peer recognition (Raymond 1999) and distributed control; and the rise of virtual learning environments (VLES) and Massive Open Online Courses (MOOCS) with their use of rich, online learning experiences (e.g. Trafford and Shirota 2011; Cabiria 2012). Reflecting on our initial failure, we decided against trying to supply a single set of polished course materials and instead looked for a way to provide teachers with building blocks from which they could assemble an *individualised* course suited to their institutional and pedagogical needs.

We drew inspiration from the original FOSS iPython project (Pérez and Granger 2007)—expanded into the Jupyter project in 2016 (Kluyver et al. 2016)—and its transformative impact on the sharing of integrated code, text, and multimedia resources. And we also looked to the Python Spatial Analysis Library (PYSAL) project (Rey 2019) in which contributors take ownership of particular application areas through managed collaboration instead of a contributory 'free-for-all'. The subsequent growth of the Jupyter project has produced a rich ecosystem of tools to support the kinds of interaction needed to make this an effective teaching resource: multiuser notebooks (`jupyterhub`), automation of grading (`nbgrader`), interactive widgets for visualisation (`ipywidgets`), and packages to facilitate manipulation of the notebook itself (`nbformat` and `notebook`).

With these criteria and resources in mind, we began looking for ways to lower the 'entry costs' for faculty to both take from, and share into, a corpus of instructional material, as well as for a way to address the customisation/localisation challenge. GEOPYTER is our solution: a single, open hub of geographically focussed teaching concepts and practical programming tasks that can be flexibly assembled into classes, or entire courses! GEOPYTER recognises that teachers need to be able to develop their own classroom 'story' at their own pace (and in conformance with their institution's teaching patterns), while supporting them with a library of up-to-date materials from which they can pick and choose. So in much the same way that a computer application is compiled from the contributions of many developers, GEOPYTER allows individual classes and entire courses to be 'compiled' from the contributions of many skilled teachers and developers. As such, this positions our work within contextualised programming initiatives (Guzdial 2010; Lukkarinen and Sorva 2016), of which the CS+X approach is perhaps best known (e.g. Mir et al. 2017); however, GEOPYTER represents an important advance in this area since it not only builds in context at the level of individual modules, but at the level of teaching resources as well!

Without realising it, we were plugging into a larger framework of thinking emerging from the literature on Open Educational Resources (OER), which began as largely 'legal and economic concept' designed to support royalty and license-free access to educational resources (Butcher 2010). It should be clear from this definition that there is no *necessary* link to FOSS, or to open and reproducible research (Brunsdon and Comber 2020), since a free e-book can qualify as an OER. However, these two areas are conceptually aligned in many ways, not least through their shared interest in the '4 Rs': reuse, redistribution, revision, and remixing (Hilton et al. 2010). Within Du's (2017) OER typology GEOPYTER is clearly 'open courseware', but it is conceptually distinct from MIT's eponymous OpenCourseWare or Standford's Coursera since it is designed to promote what Ehlers calls 'Open Educational Practice' (2011) in a manner not altogether dissimilar to that the Geographic Science and Technology Body of Knowledge (University Consortium for Geographic 2018).

In line with Mishra (2017) and with Knox's (2013) call for a role for pedagogy in OER, GEOPYTER envisions an essential role for 'teachers as (co)creators': they not only have an active role in selecting components for a Session or Module, and they can contribute source material as well. The educator and section editors, of which more later, may therefore also use this process to extend their own understanding in an applied workplace setting through collaboration (see Littlejohn and Hood 2017; Eraut 2008 for relevant discussions). Of course, we must also recognise that under Wiley's (2009) ALMS typology, although GEOPYTER provides direct Access to editing tools, is meaningfully editable, and offers source file access, the level of expertise required is not insubstantial though we also expect instructors in this domain to be more comfortable than most with the process. We therefore believe that GEOPYTER is unique in incorporating the instructor's domain expertise and pedagogical strategies by design.

4 System architecture

At least in our experience, the pace of technological and methodological change in spatial data science is such that the gap between what instructors know and what students know is substantial, and in many cases might even be growing. But the differences within individual student cohorts may be greater still, and instructors need to be able to quickly assemble and update a course that meets students where they *are* rather than where the instructor or institution might *wish* them to be. At the same time, there is also a strong need to support more open-ended exploration during 'practicals' (Unwin 1980), particularly by more advanced students who may 'tune out' if progression is overly structured and rigid or just too slow to keep their attention.

The guiding insight behind GEOPYTER is that instruction in programming outside of computer science proceeds from fundamental units of learning typically built around *computing* concepts (variables, lists/arrays, dictionaries/hashes, functions/

Table 1 Overview of system components

Concept	Usage
Atom	A ‘teachable idea’ combining explanation and code for a core, typically singular, concept (e.g. variables, lists, projections, robust rescaling, etc.)
Sessions	A ‘teachable unit’ combining **multiple atoms** for delivery in a particular learning context (e.g. beginner laboratory-based course, intermediate flipped module, online course, etc.)
Modules	A **sequence of sessions** designed to achieve one or more pedagogical objective and/or required for credit in a particular education context

subroutines, etc.) to fundamental units of learning built around *analytic* concepts (cluster analysis, point patterns, spatial autocorrelation, etc.). These units must then be assembled in a way that speaks to the student cohort and its background; we mean this in two ways: first, the examples used must be domain-specific in order to speak to budding geographers, political scientists, historians, etc.; and second, it should be possible to develop courses for different types of cohorts without having to start over from scratch. In other words, how can we enable teachers to reuse many of the building blocks employed in an advanced course for masters students in an introductory course for undergraduates?

4.1 Components

From these constraints, it was clear that our system needed to support a compositional, ‘bottom-up’ approach to instructional design. So although the development of a course or class should obviously start out with a clear set of learning aims and outcomes, at a certain point the instructor will be searching for examples and code with which to teach a particular concept: What *is* a list or dictionary? What *is* *k*-means clustering? We settled on the term ‘Atoms’ to refer to these basic instructional units, and much like entries in the atomic table, we felt that they could be grouped together into sets of related concepts: the fundamentals of programming, point pattern analysis, machine learning, etc (Table 1).

Each Atom would employ *domain-specific* illustrative examples and code so as to anchor learning in problems and applications relevant to the learner.[2] An Atom could start by showing how a list can be used to hold data about cities (e.g. name, country, population), and a subsequent set of ‘cells’ (the basic ‘unit’ of Jupyter a notebook) could build on this with an illustration of how a list-of-lists allows us to add location as a latitude and longitude coordinate pair. We will return to some of the issues that this approach raises in Engagement, but it points to the importance of ensuring a degree of consistency in how the Atoms for a set of closely related topics fit together.

[2] GEOPYTER is intended for geography and planning students, but could quite easily be ‘forked’ to provide a similar set of domain-specific resources for economics, sociology, or literature students.

The purpose of the bottom-up approach is that these units can then be flexibly assembled into Sessions: from the *same ingredients* (i.e. Atoms), the instructor could create quite different modules by organising and presenting the elements in different ways. Quite simply, we do not want to have to rewrite material for each format (e.g. lecture/practical, 'flipped' classroom, or distance learning), but we also need to deal with the fact that different types of scaffolding are required and that the amount of content suitable to a 'class' in each of these formats might differ substantially.

Furthermore, sessions designed for experts (e.g. those pursuing continuing online education) might be able to 'move' students through many more Atoms of instruction in a single Session than a similarly laid out on designed for first-time programmers in an undergraduate programme. So Sessions need to be able to *incorporate* Atoms in a way that minimises the level of effort involved in finding the 'best' way to, for example, explain the concept of recursion while maximising the ability of the instructor to relate this concept to the students' practical experience (e.g. by providing a 'context' that is anchored in a locally relevant 'story' or data).

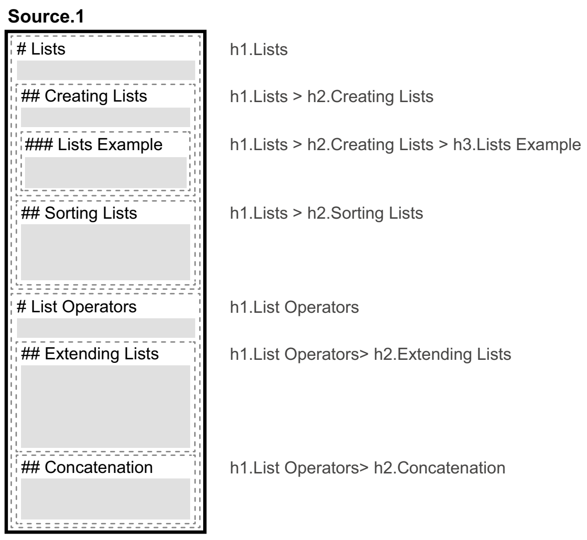
Naturally, Sessions can then be grouped into learning Modules that offer a coherent instructional programme over a period of weeks or months. Modules represent the highest level of abstraction in the proposed system, but they are also obviously the starting point from which instructors can organise their (remixed) atomic and sessional material into something incorporating a set of learning outcomes and a package of assessment appropriate to their students. However, our design reflects the expectation that contributions to geopyter at each *level* of instruction might be made by *different* people: a domain expert in Spatial Bayes might be the right person to develop an Atom on the concept and its application, but not the right person to develop a module tackling advanced spatial analytic concepts where this is just one approach amongst many. Similarly, given the global diversity of delivery formats, a 10-week term in Britain enables students to cover a very different 'volume' of content from a 15-week American semester. geopyter recognises and seeks to respond to that diversity.

4.2 Tools

So we are trying to design a system in which Sessions and Modules are *composed* out of Atoms that can be, optionally, surrounded by the instructor's own 'narrative'. We therefore want to produce a set of teaching materials that are highly portable, easily reused or edited, and that enable the instructor to select only the elements from which they wish to compose their materials. As intimated above, geopyter operationalises this through the Jupyter project and its ability to provide in-browser access to the Python interpreter,[3] and it also takes advantage of the dominance of the Git version control tool—and the GitHub service/website—as a means of tracking authorship across edits.

[3] Other languages are also possible using different kernels; there is no reason that geopyter could not be used to create instructional materials in these languages as well.

Fig. 1 Illustration of a single atom's structure


Source.1
# Lists
## Creating Lists
### Lists Example
## Sorting Lists
# List Operators
## Extending Lists
## Concatenation
h1.Lists
h1.Lists > h2.Creating Lists
h1.Lists > h2.Creating Lists > h3.Lists Example
h1.Lists > h2.Sorting Lists
h1.List Operators
h1.List Operators> h2.Extending Lists
h1.List Operators> h2.Concatenation

In theory, thanks to the combination of Jupyter and GitHub is not even necessary for the novice user to have Python installed on their own computer: since all interaction with Python is via the browser, the environment could be hosted on a server halfway round the world. In practice, however, there are few such services and most users simply download and install a free version of Python (e.g. Anaconda) that will run on their system. In our field, many people are already using this approach: notebooks can be found covering everything from introductory concepts (Millington and Reades 2017) to advanced spatial analysis methods (Arribas-Bel 2016), and combined for both complete courses or workshops (Rey 2016).

Jupyter notebooks are written as a mix of executable code cells and non-executable text formatted with the widely used 'markdown' syntax. Notebook structure is provided through headers in markdown cells: a '#' pre-pended to a line of text is generally taken to be the title of the notebook; '##' at the start of a line provides a second level of structure (i.e. Level 2 headers); '###' indicates Level 3 headers; etc. For our purposes, what is relevant is that these headers naturally yield a semantic hierarchy that corresponds closely to the h1 ... h6 model used by the HTML markup language that lies at the heart of the World Wide Web. This hierarchy allows us to 'abstract out' the problem of inferring the *meaning* of cells in different sections of the notebook since the instructor does it for us through their use of headers.

4.3 Approach

In order to assemble Atoms into Sessions and Modules, GEOPYTER necessarily requires a compositional syntax. We have noted the conceptual mapping between markdown and HTML formatting above, but how do we select some mix of code and markdown material in one notebook to be incorporated into another? And how do we do this in a way that is both simple to express and able to resolve ambiguity?

Fortunately, such a model already exists and was hinted at in Fig. 1: cascading style sheets (css) uses well-understood 'selectors' to specify one or more elements on a web page to which a set of presentational styles should be applied.

In css, an 'h1' in a style sheet indicates that all html Level 1 Headers (e.g. `<h1>A Title</h1>`) should observe the styling rules declared immediately afterwards, while 'h1.important' specifies that only a Level 1 Header of the class 'important' should be selected (e.g. `<h1 class="important">A Title</h1>`) and all other Level 1 headers ignored (e.g. `<h1 class="unimportant">A Title</h1>`). In fact, css also allows for nested selectors in which 'child' element(s) of a 'parent' can be selected in turn. This is normally used to do things like specify mouseover behaviours for a menu: that all anchors (i.e. links) that are within a division of class `menu` should act in *this* way when the mouse passes over them (e.g. `div.menu a.hover`). What is particularly elegant about css is that it provides a means for selecting multiple pieces of content in the document in one declaration (where this is desirable) *and* a means for disambiguating content with the same name, but in different locations within a document hierarchy (where it is not).

Conceptually, geopyter adapts this syntax to allow us to select some or all of a Jupyter notebook using the structure imparted by the instructor: all cells coming after a Level 1 Header are considered to be part of that element's semantic field until another Level 1 Header is encountered or the end of the document is encountered, whichever comes first. And a Level 2 Header coming 'after' (a 'child', if you prefer) a Level 1 Header is considered part of that 'parent' element's semantic field, *but* we can select it uniquely within the notebook using the standard css form of `h1.content h2.subcontent`. This is illustrated in schematic form in Fig. 1, but note that the > is simply make clear the hierarchical relationship. With this, we have essentially repurposed css as a means of selecting and importing content from one notebook into another!

Unfortunately, the nature of Jupyter notebooks does not allow this to happen dynamically at run-time, but it does allow something similar to happen when an instructor is 'compiling' new Sessional and Module content. In short, the instructor writes whatever content they wish but, using syntax similar to the examples below, wherever they want to incorporate contributed content from geopyter (or elsewhere) they have only to 'include' it by specifying both a source and a selection. And this approach works recursively: a notebook can include content from a notebook that itself includes content from another notebook.

4.4 Syntax

To recap, we typically envision an Atom as a short notebook focussing on a core concept or method (e.g. lists, object-oriented design, or spatial autocorrelation); some or all of each Atom can then be selected and imported into a Session, which is itself a notebook; and the sessions can then be selected and managed through a Module, which can *also* be a notebook or a set of notebooks. This process is initiated by the instructor creating a blank text cell in a Jupyter notebook and writing

```
# Session 2: Lists

Module leader : Associate Prof . X
Contact information : prof . x@foo. bar . uk
```

```
@include {
    ' nb'       = ' http :// geopyter . org / atoms/ fundamentals / lists . ipynb ' ,
    ' select '  = ' h1. Lists '
}
```

```
## All Done?

For next week please read the following : ...
```

Fig. 2 Illustrative Jupyter notebook content

an ‘include’ statement. The statement should be the *only* content in the cell since GEOPYTER will be replacing the cell with an unknown number of whole text and code cells from the referenced notebook.

Crucially, `include` statements can be freely intermingled with the instructor’s own content (as shown in Fig. 2, allowing the instructor to ‘frame’ the concepts in a way that suits their teaching style but which saves them having to reinvent the wheel for each class. A Session tackling standardisation could include elements of the relevant Atom from GEOPYTER while still allowing the instructor to interject comments, observations, questions, and additional tasks to ground the learning experience in the local context (individual, institutional, etc.). To illustrate this more clearly, an Atom on Python’s approach to dealing with lists could be incorporated into a longer Session as follows:

Here, `nb` is a path—local or remote—to a valid Jupyter notebook from which the instructor wants to import content. The `select` parameter specifies a selector for which the GEOPYTER tool will search within the source notebook. All content from that point onwards *up to the next selector at the same level* will the then be copied into the compiled notebook. In the above example, if there were a following `h1` covering, for example, ‘List Operators’ then this would *not* be included because, from a structural standpoint, it is at the same level in the hierarchy as ‘Lists’ but has *not* been selected. Furthermore, any `h2` or `h3` subsections within the ‘Lists’ section *would* be included since they are presumed to be providing pedagogical and logical structure to the Lists section and so should be carried over.

Clearly, an instructor might want to import only part of a section, or to suppress a subsection falling in the middle of a larger resource. In anticipation of this need, more complex ‘include’ statements with no equivalent in CSS are also possible:

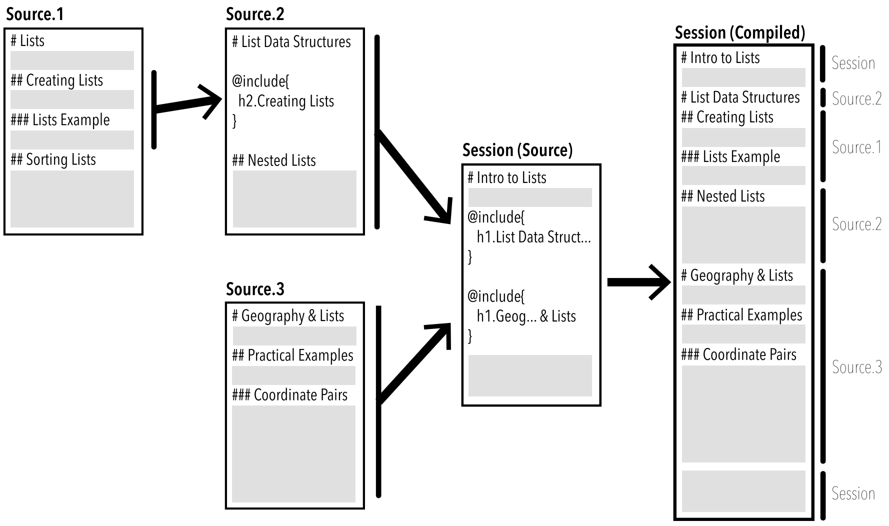


Fig. 3 Illustration of GEOPYTER notebook 'compilation' process

```
@include {
      'resource' = 'http : //geopyter.org/atoms/fundamentals/lists.ipynb'.
      'select' = 'h1.Lists -h3.Lists Examples; h1.List Operators -h2.Concatenation'
}
```

In this second example, two Level 1 sections are imported at the same time and a Level 3 subsection from *within* each of those sections is suppressed using the '-' syntax to indicate that the section should be removed. We diverged from the CSS standard since that selectors are not separated with commas: we wanted to allow for this punctuation to be part of a section heading and felt that semicolons are rather more rare in that context. An additional point of difference from true CSS is that we allow spaces in the 'selector' because we felt that asking teachers to translate between a natural language header ('List Operators') and what CSS would consider a safe header ('List_Operators') would detract from ease of use.

4.5 Putting it all together

Jupyter notebooks use a format called JavaScript Object Notation (JSON) that is not particularly easy for most humans to read, but as it is nonetheless highly structured we can interact with it programmatically. The extensible nature of the JSON format also allows us to read and write both data and metadata not only to each notebook, but also to each and every cell in a notebook. Since metadata that is not understood by Jupyter is simply ignored, we can add our own fields to provide useful information related to instruction such as who should be given credit for contributing and any dependencies or requirements for installed libraries.

Taken together, this provides the foundation for remixing/mashing up content while still enabling to add an institutional or course-specific gloss wherever necessary. Each notebook might start with the instructor's contact information or by providing instructions for setting up the computing environment, but then make use of material developed by others for actual instruction. This process may seem quite abstract—and probably quite convoluted as well—but an illustration (Fig. 3) may help to clarify why this process is so useful.

5 Use cases

To illustrate how this approach offers a substantively new way to think about teaching programming material more generally, we here present two schematic use cases: an online module of eight sessions and an in-person module lasting one semester. It is important to stress that we envision *both* of these modules being built out of the same Atoms and that, because they are drawing from version-controlled source code, the content can be fixed on a particular release (i.e. version).

5.1 Distance learning module

There has been increasing interest at universities in Massive Open Online Courses, or moocs, as vehicles for expanding access to higher education through online delivery. For managers in higher education (he), the mooc promises both enhanced revenue and enhanced access to underserved groups. Our own experience suggests substantial student interest in 'computational social science' (Lazer et al. 2009), with international students being the most keen on such modules as they are seen to provide marketable skills (programming) for graduates from a discipline (geography) with generally high employment rates (see: Royal Geographical Society with IBG n.d.).

Associate Professor X wishes to offer a *Foundations of Spatial Data Science* module to second year undergraduates, but anticipates high demand exceeding her capacity to teach in a computer cluster and high attrition since many students will decide that programming is 'not for them'. Consequently, the decision is made to offer this module on a distance learning basis with pre-recorded video content and other rich media to support student learning. However, the 'remote desktop' environment that will support this module is only updated every 18 months, often in the middle of term.

geopyter significantly reduces the overhead of several of these stages: rather than focussing on tasks such as how to explain or illustrate a particular concept, the instructor can focus on developing the multimedia content. The similarity between learning to code and learning a language is noted, and each 'lesson' is designed to be completed in under an hour so that students can proceed at their own pace without being overwhelmed. As well, using Git tags Prof. X can 'fix' the version of the code and explanations used to one that is appropriate to the computing environment while continuing to update her teaching materials.

5.2 In-class delivery

Associate Professor X *also* teaches a module for Masters students that follows the more traditional lecture+practical format. Although these students are also new to programming, they have already completed a required GIS module. Here, GEOPYTER could be employed differently: since the instructor is able to interactively provide the 'frame' or 'scaffold' (see relevant discussion of 'hypermedia' in Azevedo and Jacobson 2008) upon which the learning is built, there is less need for a narrative around each task or weekly session.

The instructor might therefore simply import the same group of Atoms as above, but use a Session and Module template that leaves out the rich media and more detailed explanations since these will be discussed in class with the students. The Sessions then wrap up with an additional mini-project or mini-assessment that requires the students to translate the concepts into a new problem domain or investigate the process in more detail: 'we've seen how we can use `pandas` and `bokeh` to explore and compare the distribution of demographic groups in London, here's a link to equivalent open data for Phoenix, Arizona...'.[4]

In addition, since students are working on their own machines, the instructor can update to the latest-and-greatest much more rapidly. To enable Professor X to manage these competing requirements, we make use of Git and the GitHub web platform not only to monitor, approve, and roll back alterations to any submitted revisions, but also to provide release 'tags' to which an instructor can bind a particular instance of a Session or Module. This 'fixes' their course to a particular version of an Atom such that development of the Atom by others can continue without the instructor having to worry that the explanation or code upon which they rely will suddenly change!

6 Engagement

In effect, in both of the example use cases the role of the instructor is to provide an integrative narrative that guides the students and contextualises their choice of components. We think that this has the potential to free up instructors to focus on where they can most effectively 'add value'; not in developing yet *another* way to show how a list or dictionary works, but in explaining why they matter to a geography, political science, or literature major (see Bort et al. 2015 for an application in the literature).

[4] Interactive examples that demonstrate these use cases are available online at https://mybinder.org/v2/gh/pysal/GeoPyTeR/master.

Table 2 Indicative groups of atoms

Atomic group	Indicative content
`auto`	Approaches to spatial autocorrelation analysis
`context`	Why learn to program? Example applications. Interviews
`networks`	Working with network data and costs
`describe`	Simple descriptive statistics
`statistics`	More advanced methods of comparing data
`clustering`	Non-spatial and spatial clustering analysis
`foundations`	Foundations of programming and computer science
`os`	Basic aspects of interacting with operating systems programmatically
`viz`	Visualising data and making maps
`ml`	Approaches to Machine Learning in a spatial context
`points`	Point pattern generation and analysis
`zones`	Zonal statistics and relationship
`models`	From regression to GWR

And because it builds on FOSS approaches to software development, we expect GEOPYTER to benefit from network effects: the more people use it, the more useful it becomes, and the more people use it. The open-source, peer-generated approach is also, however, likely to present something of a challenge over time: as more people seek both to use and to contribute to GEOPYTER we would expect to see the emergence and use of divergent norms, examples, and data across Atoms, Sessions, and Modules. Although the emergence of difference styles of coding is actually quite natural in programming and could be seen as a benefit to students in terms of teaching them about this aspect of programming, it also the case that GEOPYTER could become a victim its own success if it ceases to be coherent.

6.1 Coordination

Here, we believe that the PYSAL project offers a useful template. Although PYSAL is a FOSS project, it is *not* a free-for-all: domain specialists tend to gravitate towards those parts of the project to which they have the most to contribute and, over time, those who coordinate and enable the most substantive contributions to the codebase in terms of features and performance are 'invited' to help manage individual components of the tool (e.g. `lib`, `model`, `explore`, `viz`). Overall coherence is maintained via regular calls and online discussion boards, as well as a synchronised release schedule so that changes can be coordinated, tested, and knock-on effects resolved.

In general, we would expect to see a relatively small number of committed educators and developers creating and maintaining groups of Atoms that align with their areas of expertise, interests, and teaching responsibilities. However, unlike a traditional software project there is an important role here for *teachers*, not just developers. We think that this represents a really exciting opportunity for innovative new approaches to rise to the surface. There may be only a few who can write the Python code to conduct a geographically weighted regression analysis, but it will be interesting to see how many creative, insightful ways there are to explain it!

So although we also expect interested educators to begin almost immediately picking holes in the organisational structure proposed in Table 2, some kind of starting point is needed. Moreover, as we have mentioned elsewhere, in the event of serious disagreement other instructors are free 'fork' the repository and begin changing material as they see fit. Indeed, there is nothing to prevent geopyter Sessions and Modules drawing on content spread across multiple repositories following different organisational and developmental strategies: all that's need is a url!

The educational focus of geopyter implies that it may well be the most committed teachers who end up as 'section editors' who coordinate and review contributions. The editorial approach also aligns with the obvious benefits of grouping sets of Atoms together into basic sections such as `foundations` (the basics of variables and data structures); `describe` (describing data). The section editor ensures that examples, style, and other features are consistent across individual Atoms to make it easy to generate a set of Sessions introducing the Unix file system or Local Indicators of Spatial Autocorrelation.

geopyter therefore seeks to balance the benefits of code sharing with those of local expertise: the instructor is free to write their own exegesis, if you will, of the code and its relevance to a particular Session or Module, but the burden of developing a cogent, domain-specific demonstration can be shared with others and compelling examples more widely adopted without the effort of reinventing the wheel. Where irreconcilable differences arise between pedagogical approaches or models, then we might expect to see small groups of collaborators 'fork' the codebase and offer their own models; this is, of course, a valid approach with open-source code and one from which all instructors can ultimately benefit!

6.2 Credit

The use of Git/GitHub also gives us access to contributor information in the 'commit' (i.e. editing) logs that we can propagate into notebooks so that all contributors are recognised in the final output. Although the open nature of the project means that we cannot strictly enforce attribution, geopyter seeks to make this the easier to do 'by default' through the insertion of metadata into the compiled notebook which is then used to append a list of contributors to the end of each notebook, along with any other relevant acknowledgments or copyright notices.

To facilitate reuse while protecting contributions from unacknowledged exploitation textual content in geopyter is covered by a creative commons license; however, to deal with the fact that geopyter relies extensively on open-source contributions which are incompatible with some cc licenses (see discussion in Wilson 2013), code blocks are licensed under the mit license. The manner in which contributions from authors at different institutions can be combined also 'pollutes' the materials in ways that inhibit institutional assertions of ownership over geopyter content.

geopyter also recognises the reality of the need for peer and professional recognition by incorporating attribution mechanisms directly into the compilation process.

Digital object identifiers (DOIS) can be created and curated by the 'section editors', but authorship of GEOPYTER components—be they Atoms, Sessions, or Modules—provides the contributors with peer-evaluated, impactful materials to add to promotion applications. Our intention is that both users and institutions come to better understand the extent to which open educational resources can be a *joint* project relying on the contributions of many teachers and developers.

7 Limitations

As we noted above in connection with Table 2, it is rather unlikely that our first attempt to divide up the entire field into discrete units of instruction will be entirely successful. We would also expect to draw on reference documents such as the *Body of Knowledge* (University Consortium for Geographic 2018) and *Subject Benchmark Statement* (QAA, 2014) for the 'why' and 'what' of instruction, leaving GEOPYTER to deal with the 'how'. Moreover, the open, contributory nature of the project positions us to build GEOPYTER on top of the shared understanding of many specialists with a range of ideas about how to break apart, and put back together, the constituent elements of our domain's knowledge in ways that speak to different types of students.

For the time being, we have also deliberately hobbled GEOPYTER in one important way: an `include` command must be in a cell that does not contain any other text or code. This was done primarily for simplicity: it is a lot easier to look for whole (text) cells that match a target pattern than to have to try to parse long blocks of text or code on the off-chance that an `include` might be found; it also avoids any ambiguity as to whether the `include` is a GEOPYTER or 'native' command. Not coincidentally, it is also a good deal easier to replace an entire cell (the one containing an `include`) with one or more entire cells, than to try to work out if a cell needs to be 'closed out' first.

8 Conclusion and future directions

From practical experience, conference presentations, and code, we tend to already know who is a good *programmer* or *theorist*, GEOPYTER provides a mechanism for discovering who is a good *teacher*. Sometimes these abilities may reside in the same person, but more often we expect that they will not: the strongest developers tend to be people who have been practicing software development for many years and, consequently, may have difficulty communicating their ideas to beginner- or intermediate-level students or teachers (see: Chapman 2010)! For this reason we see the broad-based community-of-practice aspect of GEOPYTER as integral at all stages of the project: system enhancement, content development, expanding coverage, and instructional design.

Consequently, GEOPYTER has a lot in common—both philosophically and practically—with the Software Carpentry movement (Wilson 2016), and although we seek to tackle a slightly narrower set of issues with a more reusable set of resources, we can take both inspiration and warning from their experience. The benefit, we think, is that while it is possible to design Sessions and Modules that follow the popular 'bootcamp' approach to instruction (though see critique in Feldon et al. 2017), we want to enable the *same* content to be employed in a carpentry format as well as a 'normal' classroom or MOOC as required, or even to enable the instructor to mash all of those formats together such that they use a 'bootcamp' format for the introduction to Unix and the command line, an online-formatted resource for foundational concepts in computer science, and a traditional course format for the (geo)data analysis instruction. All pulled from the same set of source Atoms!

Ultimately, although GEOPYTER was developed with teaching needs in mind there is, of course, no reason why it could not be put to other uses: in combination with with `nteract` (https://github.com/nteract/nteract) it would allow developers or researchers to build full-fledged applications as scripts assembled from a collection of notebooks; or as an addition to Netflix's notebook ecology to allow for enhanced resource sharing and standardisation during the development phase before features and interfaces are 'fixed' as libraries (Ufford et al. 2018). Nonetheless, our focus for the time being remains the cohort of university teachers at all levels tasked with introducing programming material to their students and wondering where to begin. We hope that GEOPYTER makes a valuable contribution to this application domain and look forward to working with others to roll out a rich, reusable teaching framework.

9 Supplemental

Demonstration notebooks can be found in the 'sessions' directory of the GEOPYTER project on GitHub: github.com/pysal/GeoPyTeR/tree/master/sessions.

References

Arribas-Bel D (2016) Geographic data science'15. Retrieved 2016-02-19, from http://darribas.org/gds15. https://doi.org/10.5281/zenodo.46313

Arribas-Bel D, Patino J, Duque J (2017) Remote sensing-based measurement of living environment deprivation: improving classical approaches with machine learning. PLoS ONE 12(5):e0176684

Arribas-Bel D, Reades J (2018) Geography and computers: past, present, and future. Geogr Compass. https://doi.org/10.1111/gec3.12403

Azevedo R, Jacobson MJ (2008) Advances in scaffolding learning with hypertext and hypermedia: a summary and critical analysis. Educ Technol Res Dev 56(1):93–100

Barba LA (2015) Computational thinking and the pedagogy of computable content. Lecture Berkely Institute for Data Science. Retrieved from https://bids.berkeley.edu/resources/videos/computational-thinking-and-pedagogy-computable-content

Barnes TJ (2013) Big data, little history. Dial Hum Geogr 3(3):297–302

Barnes TJ (2014) What's old is new, and new is old: History and geography's quantitative revolutions. Dial Hum Geogr 4(1):50–53

Bivand RS (2020) Progress in the R ecosystem for representing and handling spatial data. J Geogr Syst. https://doi.org/10.1007/s10109-020-00336-0

Bort H, Czarnik M, Brylow D (2015) Introducing computing concepts to non-majors: a case study in gothic novels. In: Proceedings of the 46th ACM technical symposium on computer science education, pp 132–137

Bowlick FJ, Goldberg DW, Bednarz SW (2017) Computer science and programming courses in geography departments in the United States. Profess Geogr 69(1):138–150

Bowlick FJ, Wright DJ (2018) Digital data-centric geography: implications for geography's frontier. Profess Geogr 70(4):687–694

Brunsdon C, Comber A (2020) Opening practice: supporting reproducibility and critical spatial data science. J Geogr Syst. https://doi.org/10.1007/s10109-020-00334-2

Butcher N (2010) Open educational resources and higher education (Tech. Rep.). OER Africa. Retrieved 22 February 2020, from https://www.oerafrica.org/FTPFolder/understanding/OER%20in%20HE%20concept%20paper.pdf

Cabiria J (2012) Connectivist learning environments: massive open online courses. In: The 2012 world congress in computer science computer engineering and applied computing, pp 16–19

Chapman L (2010) Dealing with maths anxiety: How do you teach mathematics in a geography department? J Geogr Higher Educ 34(2):205–213

Donoho D (2017) 50 years of data science. J Comput Graph Stat 26(4):745–766. https://doi.org/10.1007/978-3-642-23430-9_71

Du Y (2017) Knowledge creation and information sharing through open education resources. Technical report. University of North Texas. Retrieved 22 February 2020, from https://digital.library.unt.edu/ark:/67531/metadc1036562/

Ehlers U (2011) Extending the territory: from open educational resources to open educational practices. J Open Flex Dist Learn 15(2):1–10

Eraut M (2008) How professionals learn through Work. Technical report. University of Surry. Retrieved 22 February 2020, from http://surreyprofessionaltraining.pbworks.com/w/file/fetch/11505951/How%20Professionals%20Learn%20through%20Work.pdf

Etherington TR (2016) Teaching introductory GIS programming to geographers using an open source python approach. J Geogr Higher Educ 40(1):117–130

Fan C, Rey SJ, Myint S (2016) Spatially filtered ridge regression (SFRR): a regression framework to understanding impacts of land cover patterns on urban climate. Trans GIS

Feldon DF, Jeong S, Peugh J, Roksa J, Maahs-Fladung C, Shenoy A, Oliva M (2017) Null effects of boot camps and short-format training for PhD students in life sciences. Proc Natl Acad Sci 114(37):9854–9858

González-Bailón S (2013) Big data and the fabric of human geography. Dial Hum Geogr 3(3):292–296

Graham M, Shelton T (2013) Geography and the future of big data, big data and the future of geography. Dial Hum Geogr 3(3):255–261

Guzdial M (2010) Does contextualized computing education help? ACM Inroads 1(4):4–6

Hilton J, Wiley D, Stein J, Johnson A (2010) The four 'r's of openness and alms analysis: frameworks for open educational resources. Open Learn J Open Dist e-Learn 25(1):37–44. https://doi.org/10.1080/02680510903482132

Kluyver T, Ragan-Kelley B, Pérez F, Granger B, Bussonnier M, Frederic J, . Jupyter Development Team (2016) Jupyter notebooks—a publishing format for reproducible computational workflows. In: Loizides F, Schmidt B (eds) Positioning and power in academic publishing: players, agents and agendas, pp 97–90. IOS Press, London

Knox J (2013) Five critiques of the open educational resources movement. Teach Higher Educ 18(8):821–832. https://doi.org/10.1080/13562517.2013.774354

Lazer D, Pentland A, Adamic L, Aral S, Barabási AL, Brewer D, Van Alstyne M (2009) Life in the network: the coming age of computational social science. Science 323(5915):721–723

Ley D, Braun B, Domosh M, Elliott S, Le Heron R, Peake L, Yeoh B (2013) International Benchmarking Review of UK Human Geography. Online. Retrieved 19 September 2018, from https://esrc.ukri.org/files/research/research-and-impact-evaluation/international-benchmarking-review-of-uk-human-geography/

Littlejohn A, Hood N (2017) How educators build knowledge and expand their practice: the case of open education resources. Br J Educ Technol 48(2):499–510. https://doi.org/10.1111/bjet.12438

Lukkarinen A, Sorva J (2016) Classifying the tools of contextualized programming education and forms of media computation. In: Proceedings of the 16th koli calling international conference on computing education research, pp 51–60

Millington J, Reades J (2017) Python lessons: code Camp. Retrieved from https://kingsgeocomputation.org/teaching/code-camp/code-camp-python/lessons/

Mir DJ, Mishra S, Ruvolo P, Pollock L, Engen S (2017) How do faculty partner while teaching interdisciplinary cs+ x courses: models and experiences. J Comput Sci Coll 32(6):24–33

Mishra S (2017) Open educational resources: removing barriers from within. Dist Educ 38(3):369–380. https://doi.org/10.1080/01587919.2017.1369350

Muller CL, Kidd C (2014) Debugging geographers: teaching programming to non-computer scientists. J Geogr Higher Educ 38(2):175–192

Naik N, Kominers S, Raskar R, Glaeser E, Hidalgo C (2017) Computer vision uncovers predictors of physical urban change. Proc Natl Acad Sci 114(29):7571–7576

Pérez F, Granger BE (2007) IPython: a system for interactive scientific computing. Comput Sci Eng 9(3):21–29

QAA (2014) Subject benchmark statement for geography. Online. Retrieved 24 September 2018, from https://www.qaa.ac.uk/docs/qaa/subject-benchmark -statements/sbs-geography-14.pdf

Raymond ES (1999) The cathedral and the bazaar. O'Reilly

Reades J, Zhong C, Manley E, Milton R, Batty M (2016) Finding pearls in London's oysters. Built Environ 42(3):365–381

Rey SJ (2009) Show me the code: spatial analysis and open source. J Geogr Syst 11:191–207

Rey SJ (2016) Spatial data analysis with PySAL. Retrieved from https://github.com/sjsrey/narsc16

Rey SJ (2019) PySAL: the first 10 years. Spat Econ Anal 14(3):273–282. https://doi.org/10.1080/17421772.2019.1593495

Royal Geographical Society with IBG. (n.d.) Careers with geography: employability. Retrieved 19 September 2018, from https://www.rgs.org/geography/studying-geography-and-careers/careers/employability/

Santibanez S, Kloft M, Lakes T (2015) Performance analysis of machine learning algorithms for regression of spatial variables: a case study in the real estate industry. In: Geocomputation papers, pp 292–297. Dallas

Singleton AD (2014) Learning to code, Geogr Mag 77

Singleton AD, Spielman S, Brunsdon C (2016) Establishing a framework for o1pen geographic information science. Int J Geogr Inf Sci 30(8):1507–1521

Solís P, Anderson J, Rajagopalan S (2020) Open geospatial tools for humanitarian data creation, analysis, and learning through the global lens of youthmappers. J Geogr Syst. https://doi.org/10.1007/s10109-020-00339-x

Stevens F, Gaughan A, Linard C, Tatem A (2015) Disaggregating census data for population mapping using random forests with remotely-sensed and ancillary data. PloS One 10(2):e0107042

Torrens P (2010) Geography and computational social science. GeoJournal 75(2):133–148

Trafford P, Shirota Y (2011) An introduction to virtual learning environments. Gakushuin Econ Pap 48(10):143–51

Tukey J (1962) The future of data analysis. Ann Math Stat 33(1):1–67

Ufford M, Pacer M, Seal M, Kelley K (2018) Beyond interactive: notebook innovation at Netflix. Retrieved 19 September 2018, from https://medium.com/netflix-techblog/notebook-innovation-591ee3221233

University Consortium for Geographic Information Science. (n.d.). Gis&t body of knowledge. Retrieved 24 September 2018, from http://gistbok.ucgis.org/

Unwin D (1980) Make your practicals open-ended. J Geogr Higher Educ 4(2):39–42

Wiley D (2009) Creating open educational resources. Technical report. Materials prepared for an independent study class on open educational resources

Wilson G (2016) Software carpentry: lessons learned [version 2; referees: 3 approved]. F1000Res 3(62), 1–24. https://doi.org/10.12688/f1000research.3-62.v2

Wilson R, Wilson S (2013) Creative commons and open content. Retrieved 19 September 2018, from http://oss-watch.ac.uk/resources/cclicensing

Wyly E (2014) The new quantitative revolution. Dial Hum Geogr 4(1):26–38