ROYAL SOCIETY
OPEN SCIENCE

The PySAL ecosystem: philosophy and implementation

Journal:	*Royal Society Open Science*
Manuscript ID	Draft
Article Type:	Research
Date Submitted by the Author:	n/a
Complete List of Authors:	Rey, Sergio; University of California Riverside Anselin, Luc; University of Chicago Amaral, Pedro; Universidade Federal de Minas Gerais Arribas-Bel, Daniel; University of Liverpool, Cortes, Renan; University of California Riverside Gaboardi, James; Pennsylvania State University Kang, Wei; University of California Riverside Knaap, Elijah; University of California Riverside Li, Ziqi; Arizona State University LumnitzU, Stefanie; The University of British Columbia Oshan, Taylor; University of Maryland at College Park Shao, Hu; Environmental Systems Research Institute Inc Wolf, Levi; University of Bristol
Subject:	Statistics < MATHEMATICS
Keywords:	Spatial analysis, geocomputation, spatial data science
Subject Category:	Earth and Environmental Science

Author-supplied statements

Relevant information will appear here if provided.

Ethics

Does your article include research that required ethical approval or permits?:
This article does not present research with ethical considerations

Statement (if applicable):
CUST_IF_YES_ETHICS :No data available.

Data

It is a condition of publication that data, code and materials supporting your paper are made publicly available. Does your paper present new data?:
Yes

Statement (if applicable):
The source code for PySAL is fully open source and available on the github organizational account: https://github.com/pysal

Conflict of interest

I/We declare we have no competing interests

Statement (if applicable):
CUST_STATE_CONFLICT :No data available.

Authors' contributions

This paper has multiple authors and our individual contributions were as below

Statement (if applicable):
Explained in author contribution section of the manuscript.

ROYAL SOCIETY OPEN SCIENCE

rsos.royalsocietypublishing.org

Research

Article submitted to journal

Keywords:
Spatial analysis, open-source computation, spatial data science

Author for correspondence:
Sergio J. Rey
e-mail: sergio.rey@ucr.edu

The **PySAL** ecosystem: philosophy and implementation

Sergio J. Rey[1], Luc Anselin[2], Pedro Amaral[3], Dani Arribas-Bel[4], Renan Xavier Cortes[1], James David Gaboardi[5], Wei Kang[1], Elijah Knaap[1], Ziqi Li[6], Stefanie Lumnitz[7], Taylor M. Oshan[8], Hu Shao[9], Levi John Wolf[10]

[1]University of California, Riverside
[2]University of Chicago
[3]Universidade Federal de minas Gerais
[4]University of Liverpool
[5]The Pennsylvania State University
[6]Arizona State University
[7]University of British Columbia
[8]University of Maryland
[9]Environmental Systems Research Institute (Esri)
[10]University of Bristol

PySAL is a library for geocomputation and spatial data science. Written in Python, the library has a long history of supporting novel scholarship and broadening methodological impacts far afield of academic work. Recently, many new techniques, methods of analyses, and development modes have been implemented, making the library much larger and more encompassing than that previously discussed in the literature [68]. As such, we provide an introduction to the library as it stands now, as well as the scientific and conceptual underpinnings of its core set of components. Finally, we provide a prospective look at the library's future evolution.

THE ROYAL SOCIETY PUBLISHING

1. Introduction

In recent years, it has become increasingly important for scientists to adopt open science practices [59,60], especially for junior or early-career researchers [1]. An open approach enhances reproducibility, transparency, and speed of scientific workflows and discovery. One critical part of open science practices is the development, improvement, maintenance, and use of open science tooling [23]. Alongside the broader trends in quantitative research towards computation-driven inquiry [19], geography has provided a fertile ground for open science [69,77]. Large-scale collaborations on technical and scientific infrastructure have long been a requirement in geography, owing to distinctive spatial data representations, statistical concerns, and computational requirements. But, in the past, many of these large-scale, open collaborations have been outpaced in functionality and computational performance by closed source, proprietary platforms. This led to widespread awareness of the challenges of "disabling technologies" in the field [29], where the implementation of a specific suite of analytical capabilities limited the conceptual and practical reach of spatial science. During the past decade, however, the situation has begun to change, as progress in methodology of spatial analysis has been aided by the availability of open source (and thus verifiable) software, in contrast to the closed source black box implementations of proprietary software, where the underlying assumptions were often not made explicit.

As a result, the influence of dependence on proprietary software has been waning, as there is now a strong case to be made for open science in geography [30,64]. Treating scientific code as text, enmeshed and integral to the scientific work, has pedagogic, scientific, and societal benefits. As part of this process, packages such as **spdep** in R [10] or **PySAL** in Python [68] serve as open libraries in two senses. First, in terms of computation, they are open libraries that support scientists *doing spatial science* through the analyses they make possible. Second, in terms of literature, they are open libraries that support students *learning spatial science* through the algorithms they make explicit. Thus, it is important to ensure long-term contributions, development, and maintenance to open scientific libraries.

Since its initial public release in 2010 [67], PySAL has demonstrated the benefits of an open source geographic science library and seen widespread adoption[1] across a diverse set of applications. As a software library, PySAL is relied upon by a number of upstream packages to develop specialized tools for spatial analysis, prominent examples include **geopandas**, **geoplot**, **momepy**, and **geosnap**. PySAL is also used by researchers in the analysis of a wide array of topics across many disciplines including political science [35], criminology [37], economics [21], planning [52], public health [38], engineering [20], environmental science [32], chemistry [78], physics [36], religion [22], biology [51], neuroscience [12], epidemiology [34], technology forecasting [44], climate change [56], organizational dynamics [83], information visualization [15], ecology [79], and sociology [48], among others.

PySAL has significantly evolved since its original inception, both technologically and as a collaborative research endeavour. This paper frames recent changes against the backdrop of the project's history, and presents the ecosystem model that was recently adopted as a solution to some of the challenges posed by its own success. The remainder of the paper is organised as follows: Section 2 reviews the process of growth and change experienced by the project since its early years to the recent move to a federated model; Section 3 introduces the new structure of the package and, in doing so, reviews the current set of functionality available in PySAL; Section 4 considers non-technical aspects of the project, including governance practices, the approach to community-building, and pedagogy; and Section 5 concludes with some reflections on the future challenges and next steps.

[1]PySAL has been downloaded over 1 million times from pypi: `https://pepy.tech/project/pysal`.

2. PySAL 2.0: Original Design, Evolution, and Current Model

(i) Original design principles

To understand the new model recently adopted, we first need to frame it under the evolution the library has experienced since its inception, now over ten years ago. At the time, the Python scientific ecosystem was largely devoid of any packages covering geospatial analysis. PySAL was conceived as an initial attempt to fill this void. Our target audience was data scientists who wanted to engage with spatial analysis using Python; as well as developers who could leverage the library to build new applications across the growing number of delivery platforms including desktop, plugins to standard geographic information systems, and web-based applications. To support those users, and fuel the dissemination of the library, we wanted to ensure that installation of PySAL was streamlined.[2] We also stressed the importance of interoperability with the wider geospatial stack outside of Python, such as the proliferation of spatial analysis packages in R, MATLAB, and STATA, brought with it language-specific implementations of spatial data formats and handlers that were beginning to limit collaboration between user and developer communities. Python had already been widely recognized as an excellent "scientific glue" [82] that could be used to leverage disparate scientific code. In designing PySAL, we wanted to leverage this feature of Python.

The original model to develop PySAL was community-driven and centralised. The code was structured as a single package with several submodules closely interrelated. Since there was little existing Python code within the domain, the first versions were focused on covering the minimum functionality required to start a spatial analysis workflow. This included functionality such as file readers and writers, and foundational data structures on which several techniques relied, e.g. spatial weights matrices. Once these building blocks were created, with development mainly driven through volunteered time, subsequent functionality focused on areas related to ongoing grants or research interest of the then developers. At the time, the development team was formed by five to ten people who already had a history of collaboration through other research projects, all based within the same academic department. These circumstances allowed for direct communication, rapid iteration of ideas, and agile progress. However, it also meant that efforts to establishing more formal channels of communication and scaffolding to integrate new contributors external to the PySAL project remained aspirational.

(ii) Growth in a time of (technical) debt

While this first model of growth and community-driven development was innovative for the field of spatial sciences and spatial econometrics it also resulted in "technical debt" [43]. Though it succeeded in getting the project off the ground, choices about the structure of the software affected the software's subsequent growth, maintenance, and future stability. First, designing the package as a monolithic distribution of spatial analysis functions meant that some parts of the library were very tightly coupled [57], causing changes introduced into these components of the library to immediately and substantially affect other parts. In some cases, this resulted in "cascades" of faults: changes in the "stack", the numerical and computational libraries upon which PySAL relied (e.g. **numpy**, 80; **scipy**, 84) could introduce software faults in one part of PySAL and the entire package would thus be compromised. Additionally, attempts to resolve such faults would sometimes introduce new issues in other parts of the library. Triage of these problems and their fixes was difficult; especially alongside accepting new contributions and continuing the development of the package as new scientific advances were made. This situation made the library confusing to use for those not directly involved in the development process and complicated the process of accepting outside contributions.

Second, the history of the package significantly guided how the package was distributed and discussed. Since many in the original development group understood most of the functionality

[2] At the time, there were no dedicated package managers such as Anaconda, Inc.'s **conda** or container technologies such as Docker, and installing certain dependencies was notably more involved than currently.

across the library, all functionality was distributed together and exposed at the same Application Program Interface (API) endpoint. For example, users interested in conducting spatial regression analyses would also be exposed to a set of statistics and tools for the analysis of Markov Chains in the program's interface. This API design also meant that explanations of what PySAL could do were difficult to focus, further complicating user experience. Beyond PySAL being perceived as a large toolbox filled with many tools used for very different purposes, for a new user it remained unclear as to how the pieces all fit together.

Third, the maintenance required by tight coupling hindered the efforts of new contributors. Because parts of the package were tightly coupled, new contributions often required large amounts of editorial work by a team of maintainers, before novel functionality or enhancements could be integrated. This caused a situation akin to the Matthew effect [49]: a new contributor may make a significant and novel addition, but this addition would be credited to experienced contributors. Significant contributions from new community members would require integration work elsewhere in the package. This integration work would normally be done by senior maintainers. The whole contribution would get "credited" to the senior maintainer that made it possible to include the new functionality. This is in spite of the fact that the new contributor led the effort and that the maintainers were adamant about sharing credit. While this is primarily a social problem, the tight coupling of the software exacerbated the integration effort required to include new contributions.

Finally, the tight coupling between library components seriously limited the library's ability to grow, refactor, and integrate with new *dependencies* as the Python ecosystem grew.[3] As a design principle, the Python language adopts a loose collection of statements set forth in Peters [58]. The thirteenth statement, "There should be one—and preferably only one—obvious way to do [a task]," became particularly challenging to obey due to the tight internal coupling in PySAL. Some parts of PySAL had been written before Python was mainstream in science. Critically, Python's widespread adoption in spatial sciences meant that new packages often replicated and improved the infrastructure that PySAL built in service of its main point: cutting-edge spatial analytics. But, it was difficult to justify the inclusion of these new community projects; since the packages of interest only provided computational infrastructure, the work needed to re-tool the infrastructure was less important than that implementing scientifically novel algorithms. This meant that new packages were not adopted even when they significantly improved upon existing functionality. As the number of new packages replicating or improving basic functionality increased, PySAL's internally-consistent structure also lead users to think that the package was not integrated in the wider discipline. As the tide of geospatial packages in Python continued to rise, PySAL needed to cut this tight-coupling tether, relying on the growing geospatial infrastructure in Python to hone the library's core competencies in spatial analytics.

(iii) A "federated" solution

The solution proposed and implemented to the growing challenge of maintaining and expanding PySAL was to move from a tightly integrated to a *federated* model. Rather than contributing all code to a monolithic package that holds all functionality, the project moved to a model where functionality was split into several, smaller packages, each having a clearly delimited area of focus. Each of these packages are now independent Python packages in their own right. As such, they may have different maintainers, release cycles, and sets of dependencies. In this context, the library **PySAL** becomes an "aggregator", or a meta-package, that brings together all of these packages under a common brand and interface with a single install: every six months, PySAL collects the latest release of each federated package, wraps them under a common API, and releases it in a bundle.

This model brings together benefits from a monolithic and a fully distributed approach. Because the functionality is split across independent, self-contained packages, development is

[3] *Dependencies* are other Python packages that provide algorithms or computational objects that form the foundations upon which PySAL's analytical functionality was built.

faster and more agile. Developers can rely on official versions of other packages to develop their own, and can focus on expanding functionality rather than ensuring their changes do not affect other ends of the library. Equally, testing any single package and catching faults is faster since each package's tests are now isolated from other packages. Furthermore, since packages are independent, releases of each sub-package can take place as soon as the developers agree to, without having to coordinate with a larger team. Users interested in the functionality contained in one package can install only that package, bringing a smaller footprint and a more limited set of dependencies. Finally, it is easier to explain the purpose and functionality of each smaller package, as they focus on and contain only related functionality.

These independent packages with more focused functionality also provide a venue to spread the credit and enhance contributions from outside. As discussed above, the monolithic approach lends itself more to focus attention on a smaller set of developers and maintainers, even though a larger group might be contributing functionality. A federated approach opens the option to include more developers in lead roles as package maintainers, and provides more opportunities to disseminate the functionality in independent papers (e.g. 47) or other venues, such as citable software releases through **zenodo** [50]. This ensures the community is healthy, broad, well-integrated and provides incentives to grow in diversity and functionality [87]. At the same time, the meta-package offers several of the benefits of the monolithic approach. Users with less specific needs, can rely on the six-month release to provide a stable, one-install version that requires all the dependencies and installs the entire set of functionality provided by the federation. This "aggregator" also acts as a platform with higher visibility that makes it easier to discover functionality.

3. Current Analytical Capabilities

The new federated approach means **PySAL** is a meta-package that re-distributes several independent smaller packages. The purpose of grouping several packages loosely connected thematically is to bundle, organise, and assure quality, so that the result is a consistent platform for spatial analytics. In this sense, the project has de-coupled the structure of software development from the issues of providing a platform that is easy to access, learn, and deploy. This way, the distribution and development issues are now resolved within each federated package, while the concerns about consistency and pedagogical clarity are addressed in the meta-package. Since the number of packages that **PySAL** encompasses is relatively large[4], and is expected to grow over time, the team decided to organise functionality in more general thematic categories, such as visualization or data exploration, and re-wrote the API to reflect such change. The result is PySAL 2.0, released first in 2019.

The **PySAL** 2.*X* series organizes functionality around four main areas or domains: **lib** - core data structures and foundational algorithms-, **explore** - spatial data exploration -, **model** - explicitly-spatial modelling -, and **viz** - tools for visualization of spatial statistical analysis. Each of these domains is broadly aligned with different components of a spatial analysis workflow, and accordingly houses packages providing related functionality. To reflect this feature, each federated package is imported from within its own domain. The remainder of this section briefly describes the packages present in each domain for the original 2.0 release.

(a) Foundational Algorithms: **libpysal**

Underpinning the three domains, **libpysal** provides foundational algorithms and data structures that support the rest of the library. This currently includes the following modules: input/output (**io**), which provides readers and writers for common geospatial file formats[5]; weights (**weights**), which provides the main class to store spatial weights matrices, as well as several utilities to

[4]The first meta-package version of PySAL (2.0) consisted of 14 pakcages.
[5]Much of these are provided in a legacy mode to avoid breaking backwards compatibility. However, the consensus among the development team is to offload much of this area to related packages such as **geopandas** or **rasterio**.

manipulate and operate on them; computational geometry (**cg**), with several algorithms, such as Voronoi tessellations and alpha shapes [18] that efficiently transform geometric shapes; and an additional module with example data sets (**examples**). This domain is also a single stand-alone package due to its core importance to other domains.

(b) Exploratory Spatial Data Analysis: **explore**

The **explore** layer of PySAL includes modules to conduct exploratory analysis of spatial and spatio-temporal data. At a high level, packages in **explore** are focused on enabling the user to better understand patterns in the data and suggest new interesting questions rather than answer existing ones. They include methods to characterize the structure of spatial distributions (either on networks, in continuous space, or on polygonal lattices). In addition, this domain offers methods to examine the *dynamics* of these distributions, such as how their composition or spatial extent changes over time.

(i) **esda**

Exploratory spatial data analysis (ESDA) involves the interrogation of patterns in spatial data. Common topics in ESDA include the analysis of *spatial dependence*, where realizations from a random spatial process depend on other nearby realizations and *spatial heterogeneity* where a process may exhibit different behavior in different areas. In exploratory spatial data analysis, *spatial autocorrelation*, statistical dependence of a given variable with other nearby measurements of that same variable, is often critical to identify and understand. The **esda** package implements methods for the analysis of both global (map-wide) and local (focal) spatial autocorrelation [3], for both continuous and binary data. In addition, the package offers new statistics about boundary strength [86] and measures of aggregation error in statistical analyses [17].

(ii) **giddy**

Geospatial Distribution Dynamics (**giddy**) is an extension of ESDA to spatio-temporal data. The package hosts state-of-the-art methods that explicitly consider the role of space in the dynamics of distributions over time [39]. A full set of spatially-extended discrete Markov chain models, including Spatial Markov, LISA Markov, Full Rank Markov, and Geographic Rank Markov models [62,65] are available for users who are interested in the underlying transitional dynamics of a process as well as how the spatial structure shapes such dynamics. Global and Local Indicators of Mobility Association, GIMA and LIMA [66], are also provided in **giddy**. These indicators assess the degree to which changes in the positions in an (income) distribution over two time periods displays a global or local spatial pattern.

(iii) **inequality**

Indices for measuring inequality over space and time are included in the **inequality** package. These comprise classic measures such as the Theil T information index and the Gini index in mean deviation form; but also spatially-explicit measures that incorporate the location and spatial configuration of observations in the calculation of inequality measures. For example, the Theil inequality index can be decomposed into *between* and *within* inequality contributions, or the so-called inter- and intra-regional inequality [63]. Complementing this partition-based approach, the package also provides a Spatial Gini decomposition [72] that can be used to test if inequality is distinct between observations that are spatial neighbors and those that are not. Complementing the implementation of measures of inequality, several statistics also include inference methods that use a variety of permutation-based and analytical approaches.

(iv) **pointpats**

The statistical analysis of point data is supported by the **pointpats** package [61]. This package provides methods to characterise the spatial structure of an observed point pattern: a collection of locations where some phenomena of interest have been recorded. Measures of centrography provide overall geometric summaries of the point pattern, including central tendency, dispersion, intensity, and extent. In addition, **pointpats** supports a flexible window, or geometric frame, that is used in the calculation of these descriptive measures and in visualizations. This window is also used to implement formal tests for clustering or co-location, including quadrat-based methods and distance-based methods [81].

(v) **segregation**

The **segregation** package calculates over forty different segregation indices and provides a suite of additional features for measurement, visualization, and hypothesis testing that together represent the state-of-the-art in quantitative segregation analysis [14]. These methods are exposed to the user through a streamlined interface that allows the calculation of common and advanced measures of segregation, including aspatial, spatial, two-group, multi-group, and localized indices. In addition, the spatial structure of a dataset can be represented using spatial weights from the **lib** domain, or street network distances that can depict a more detailed picture of urban accessibility. Users of **segregation** can also perform simulation-based hypothesis testing for single values (e.g. when testing for the presence or absence of segregation) or value pairs (e.g. when testing whether a given city is more segregated than another), as well as decompose comparisons into spatial and demographic structures.

(vi) **spaghetti**

Many spatial processes are constrained to networks, and hence, studying them in a euclidean-based framework may lead to results that are less representative of reality [7,16]. Therefore, **Spa**tial **G**raph**s**: N**et**works, **T**opology, & **I**nference (**spaghetti**) was developed to provide data structures and analytical methods to study networks and statistical processes on networks [28]. For instance, the Network *K* Function allows for the statistical testing of clusters on networks [53, Ch. 6]. In order to make these kinds of statistics efficient, **spaghetti** provides a robust all-to-all Dijkstra shortest path algorithm with multiprocessing functionality. Other current functionality includes high-performance geometric and spatial computations using **geopandas** that are necessary for high-resolution interpolation along networks, and the ability to connect near-network observations onto the network [27].

(c) Explicitly-Spatial Statistical Modelling: **model**

In contrast to **explore**, the **model** layer focuses on confirmatory analysis. In particular, its packages focus on the estimation of spatial relationships in data with a variety of linear, generalized-linear, generalized-additive, nonlinear, multi-level, and local regression models.

(i) **mgwr**

Geographically-weighted regression (GWR) is a central tool in geographical analysis [24]. At its core, geographically-weighted regression models are a *local regression* technique [13] that borrows data from nearby locations to estimate place-specific coefficients. The method recognizes that parameters may vary across the spatial domain when the same stimulus elicits a different response depending upon geographical context. Recent innovations in the GWR methodology remove the limitation that only one scale is considered; typically a single "bandwidth" controls how far sites are allowed to borrow data for all of relationships in the model. Multiscale GWR is a new approach based on generalized additive models [88] that allows for bandwidths that vary uniquely for each predictor [26]. This means that data borrowing might be more local for

some covariates than others, suggesting more nuanced patterns in the relationships between a set of covariates and a response. Altogether, the **mgwr** package provides scalable algorithms for estimation, inference, and prediction using single- and multi-scale geographically-weighted regression models in a variety of generalized linear model frameworks, as well model diagnostics tools [54].

(ii) spglm

In order to solve geographical modelling problems efficiently, it is useful to employ sparse matrix operations where possible [9]. Existing generalized linear modelling frameworks in Python, such as **statsmodels** [76], did not fully incorporate sparse methods in its generalized linear modelling frameworks. To address this gap, **spglm** implements a set of generalized linear regression techniques, including Gaussian, Poisson, and Logistic regression, that allow for sparse matrix operations in their computation and estimation to lower memory overhead and decreased computation time.

(iii) spint

Spatial interaction models are a class of geographical models for studying the interaction between places [8,25,75]. **spint** seeks to provide a collection of tools to study spatial interaction processes and analyze spatial interaction data [55]. A primary functionality of **spint** is to facilitate the calibration and interpretation of a family of gravity-type spatial interaction models, including those with *production* constraints (where total outgoing flows predicted by the model must be unbiased), *attraction* constraints (where total incoming flows predicted by the model must be unbiased), or a combination of the two constraints [85]. Given the unique structure of calibrating models with constraints, **spint** provides scalable algorithms by leveraging sparse matrix operations in **spglm**.

(iv) spreg

The package **spreg** supports the estimation of classic and spatial econometric models. Currently it contains methods for estimating standard Ordinary Least Squares (OLS), Two Stage Least Squares (2SLS) and Seemingly Unrelated Regressions (SUR), in addition to various tests of homokestadicity, normality, spatial randomness, and different types of spatial autocorrelation. There is also a suite of tests for spatial dependence in models with binary dependent variables [2]. The package enables the incorporation of both spatial dependence and spatial heterogeneity into traditional econometric models. To deal with spatial dependence, the package contains methods for estimating spatial lag and/or error models. Different flavours of these methods are available according with the characteristics of the specification: with/without heteroskedasticity or with/without endogenous predictors. Most of these models can then be fit via Generalised Method of Moments—GMM—or Maximum Likelihood—ML. To incorporate spatial heterogeneity, **spreg** allows the specification of spatial regimes in all of its methods, and provides tests for coefficient stability. For spatial panel estimations, **spreg** contains classic Spatial SUR, Spatial Three Stage Least Squares, Lag SUR and Error SUR, in addition to Likelihood Ratio, Lagrange Multipliers and Chow tests to assess model specification or evaluate parameters. Additional details on these methods, as well as its implementation in the package, can be found in Anselin and Rey [4].

(v) spvcm

Variance components models are a kind of multilevel model used extensively in social science [31, 33]. They are most useful in situations where the differences between groups are of interest, but groups are of varying sizes or have differing levels of variation. Variance components methods partition variation into "within" group and "between" group variation, allowing for separate group-level and individual-level error terms. These models can be estimated using a variety of

Bayesian and Maximum Likelihood methods [11]. In **spvcm**, a general framework for estimating spatially-correlated variance components models is provided. This class of models allows for spatial dependence in the variance components, so that nearby groups may affect one another [45]. The **spvcm** package also provides a general-purpose framework for estimating models using Gibbs sampling in Python, accelerated by the **numba** package [46].

(d) Visualisation Layer: **viz**

The **viz** layer provides functionality to support the creation of geovisualisations and visual representations of outputs from a variety of spatial analyses. Visualization plays a central role in modern spatial/geographic data science. Current packages provide classification methods for choropleth mapping and a common API for linking PySAL outputs to visualization tool-kits in the Python ecosystem.

(i) **mapclassify**

Choropleth maps are thematic maps that rely on shading, color, or patterning to represent the measurement of a statistical attribute across polygonal areas. The effective design of a choropleth map requires careful consideration of the symbolization as well as the choice of classification scheme that assigns observations to different map classes. The **mapclassify** package in PySAL addresses the second design imperative. Currently, fifteen different classification schemes are available in **mapclassify**, including a highly-optimized implementation of Fisher-Jenks optimal classification [73]. Each scheme inherits a common structure that ensures computations are scalable and supports applications in streaming contexts. The popular geoprocessing and visualization packages **geopandas** and **geoplot** use **mapclassify**.

(ii) **splot**

The **splot** package provides statistical visualizations for spatial analysis. The package offers, i.e. methods for visualizing global and local spatial autocorrelation (through Moran scatterplots and cluster maps), temporal analysis of cluster dynamics (through heatmaps and rose diagrams), and multivariate choropleth mapping (through value-by-alpha maps; 74). A high level API supports the creation of publication-ready visualizations. Functionality that provides multiple views (i.e. scatterplots combined with cluster maps) and small multiples (i.e. facet plots) help to guide users in their visual analytics workflow and parameter choices through a "grammar of graphics." **splot**'s functionality is implemented across different graphical engines available in Python (including matplotlib and bokeh) to allow for static and interactive visualizations.

4. Pedagogy and Community

Since its inception in [68], **PySAL** has been a pedagogical project as much as a software library. This ambition has materialised over time in two distinctive goals: one, to serve as a platform that makes cutting-edge spatial analytic techniques available and accessible to a wide range of users; and two, a pedagogical one, to employ computer programming as a medium to communicate advanced statistical concepts. At the same time, the project has been built following standard approaches in the world of open-source development that have reached beyond pure software development and into community building, which is structured through a transparent governance model. Over the years, the role of these two aspects -pedagogy and community- has grown in both relevance and the amount of effort devoted. This section unpacks some of the approaches adopted and provides further detail on the processes established.

The pedagogical ethos of the project comes across in two broad areas: documentation and additional materials. First, an exhaustive, clear, and updated documentation is complemented through direct access to the source code. From the very beginning, a compulsory requirement for any functionality added to **PySAL** has been to include a "docstring" together with new code.

These are human-language explanations of what the method, class, or package does, along with a list of what is required to pass as input, and what the user can expect to receive as output, as well as a small example demonstrating its use. This close integration between computer code and human explanation, although by no means new or unique to **PySAL** [42], has been a distinctive feature of the library enhancing the understanding of functionality with wide coverage and consistency. The rationale behind this approach to developing community code is the belief that, by making the code easily accessible and complementing it with concise explanations, the user is more likely to use "code as text," as [64] argues. This supports and facilitates the transition from users of the package into developers and computational scholars. Well-documented code is easier to inspect and understand, so these users can get involved in the library's inner workings, and obtain a deeper insight into the computation and methodological details. This approach supports the library, in that it trains new developers and contributors, but it also supports the broader academic discipline, because it makes the procedures involved in new science explicit.

Second, much of the effort of the team has been directed not only to detailed software documentation but also at creating pedagogical materials through the integration of the software with study resources. For example, [5] and [70] introduce students to the nascent field of Geographic Data Science. To do so, they feature **PySAL** extensively. This material serves the purpose of extended, narrative documentation for the software; at the same time, the pedagogical approach to theoretical concepts is enriched by being able to take an explicitly-computational perspective, illustrating statistics with code snippets. The value of these materials is augmented by an additional effort to promote **PySAL** in a wide range of workshops and short courses.[6]

In addition to pedagogy, **PySAL** has paid special attention to governance. Its first ten years of existence saw the project grow from a small team localised in the same department, to a larger collective distributed across the globe. To make this transition successful, several activities that used to take place in an informal setting in the early days were taken forward more proactively. First, collaboration around code was from the early days coordinated through an open, version control-based platform (Google Code first, Github currently). These platforms offer a detailed log of changes and, through "commit messages", "issues" and "pull requests", allow to reconstruct the evolution of the project as well as the technical discussions that surrounded it. Given the geographical distribution of developers, the team uses a monthly call to cover aspects of the development for which written discussion was not practical. Topics such as the transition to Python 3 or the reorganisation of the library in subpackages were fleshed out in these calls, but also coordination around conference attendance or workshop proposals. Even though development is technically possible with the practices just described, the team has been purposeful about maintaining a regular schedule of face-to-face meetings. Usually held in the form of "code sprints" alongside academic conferences (such as the American Association of Geographers, the North American Regional Science, or the Scientific Python Conferences), these events serve a double purpose: first, they focus attention to particular areas of the project (maintenance, documentation, code development) that the group has identified as a priority; second, they act as a "social glue", keeping team members involved and engaged.

As the project has grown, it has become important to formalise how to integrate and foster external contributions. We have developed a code of conduct[7] that provides guidelines for interaction and collaboration around **PySAL** to any individual interested in contributing. As described above, part of the rationale behind moving to a federated model is to foster external contributions and to have a more flexible framework to incorporate cognate packages. To make this process easier, **PySAL** also has a package template[8] that details expected requirements from any package that wishes to join the federation.

[6]For an an illustration of materials developed with this outlets in mind, the reader is referred to: http://pysal.org/notebooks

[7]A copy is available at: https://github.com/pysal/governance/blob/master/conduct/code_of_conduct.rst

[8]A copy is available at: https://github.com/pysal/submodule_template

Besides collaboration within the **PySAL** framework, the team has embraced interaction with the larger Python community for data science. Rather than "reinventing the wheel", our goal is to provide the spatial analytic layer that makes cutting edge geospatial techniques available and integrates seamlessly within the larger ecosystem of Python packages and tools. An example of this strategy is the integration of choropleth classification schemes from **mapclassify** into the **geopandas** plotting API, or the interoperability between most of **PySAL** and **GeoDataFrames**, the foundational tabular data structure for geospatial data in **Python**. These technical integrations have been possible thanks to (but have also contributed to) closer collaboration with the development teams of other components of the ecosystem such as **geopandas** or **matplotlib**. Thanks to architecture and governance changes, the package is now much more embedded into the ecosystem at large.

Finally, a note on funding. Although much of the work devoted to **PySAL** has come out of traditional "research time", the team has begun to explore alternative and complementary funding models to support development. Many of the features currently available were developed as part of larger research projects and grants that required a computational implementation of a method that was not available. For example, the **giddy** package emerged out of substantive research on income inequality dynamics carried out by members of the team (e.g. 40,41,66,71). A more recent funding stream has been in the form of the Google Summer of Code[9], a program run by Google that funds students to work on implementing new features on open-source projects. **PySAL** has used this model to rewrite internal core data structures, to add new functionality to already existing packages, or to develop brand new packages such as **spint** or **splot**. Additionally, in 2019 PySAL joined NumFocus[10] and is now eligible for small grants to support the development of open source scientific software.

5. Future Plans and Next Steps

The first ten years of PySAL have seen the project evolve from a small, single package into a synchronised federation of packages that collectively enhance spatial analytics in **Python**. In this process, the technical and human infrastructure that support it has experienced profound changes, evolving to meet the demands of the given time. With all this ground covered, the logical question is: *What's next?* In this concluding section, we explore what lies ahead; what we consider as the main opportunities for the project to continue growing, but also the main challenges.

(a) Specific plans

A keen interest of several contributors to PySAL has been to build a first-class module for spatial optimisation and regionalization. Compared to other functionality in the project, optimisation has a more complicated set of dependencies. Spatial optimization algorithms heavily rely on general purpose optimisation or linear programming libraries once the spatial information has been expressed as a standard optimisation problem. These general-purpose optimization libraries must operate at peak performance given the difficulty of solving many spatial optimization problems. Since early releases, PySAL included a **region** module with a few algorithms implemented separately. However, it soon became clear that a more unified approach that offloads heavy computations to a general linear-programming library would be more efficient. This led to a Google Summer of Code to re-write **region** with a unified approach to its API. Recently however, as the ambitions of other packages such as **spaghetti** have expanded into domains that also require optimisation routines, the team has decided to move development to a new **spopt** package that unifies the approach, and provides underlying spatial optimisation routines in a more flexible and general way. In this context, there is an ample agenda to implement core algorithms, expose them through general interfaces, and then use them to build applications related to regionalization problems (e.g. spectral clustering, the SKATER or REDCAP algorithms),

[9] https://summerofcode.withgoogle.com
[10] https://numfocus.org/

spatial optimisation along networks (e.g. optimal facility location modeling), or other domains where it might be useful.

A second area of interest aims to provide better integration with related libraries from the **Python** data science ecosystem. As mentioned above, the first efforts in PySAL had to be spent in building a set of utilities that, even though were not planned as a core part of the library, allowed the user to interface with spatial data (e.g. shapefile readers and writers). As the Python community evolved, these tasks were taken up by more comprehensive projects and the main priority of PySAL in this respect became to appropriately interface with these projects. A good example of this is **geopandas**, a package that extends functionality of **pandas** datatypes to spatial data. Once the project matured, it allowed PySAL to drop support for file I/O and focus on analytics. But **geopandas** also became a direct user of the PySAL ecosystem by using choropleth classification algorithms from **mapclassify** instead of re-implementing them. As the ecosystem matures and foundational libraries become more established and stable, a priority of PySAL is to further integrate with this functionality, making it not only possible but pleasant to write code that seamlessly knits different projects into a unified workflow that favors developer productivity and computational performance.

Finally, we increasingly see Python as one of many environments with which scholars and industry researchers conduct their work. So-called "polyglot" environments that seamlessly allow scholars to use packages from different computer languages in a single analysis are becoming increasingly common. This suggests that users of the library and developers building on top of the library may actually be coming from entirely different computing platforms. Further, documenting the interactions between software ecosystems becomes important when considering actual analytical workflows, where it may be easier to conduct some parts of an analysis in some environments and not others [6]. Thus, it becomes important for the library to document and build upon its integrations with other packages, including desktop GIS software (QGIS, ESRI ArcGIS), and other computing languages (such as Julia or R), in order to ensure that PySAL is usable in whatever environment the user actually resides.

(b) General reflections

Beyond specific ideas, a series of guiding principles and ambitions are likely to be at the heart of the next "big decisions." The first one is the sense that the ground work required to build a platform of spatial analytics, and its place in the broader data science Python ecosystem, is largely completed. Maintenance (not a light task) aside, this makes it possible to focus entirely on ensuring the cutting edge methods are implemented shortly after they are invented. Our plan is that each federated package stays at the frontier of the domain whose functionality it represents. A key ingredient of this idea is to reach out to scientists beyond the core development team and work with them to integrate their methods in PySAL code. Much of this process is enhanced with the move to a federated model discussed in the second section and, to some extent, it is already at work. For example, the authors of the "S-MAUP" statistic proposed in [17] contributed their code to begin its implementation as part of **esda**.[11] Given recent changes in the library, we can effectively integrate contributions directly from the original authors rather than having to shoulder the burden of re-implementing cutting edge algorithms ourselves. Going forward, we will continue to integrate cutting-edge spatial science into PySAL given its new governance and technical structures.

To end this paper, we would also like to reflect on what we believe has been the most successful lesson learned over this period: the ability to maintain a flexible approach to adapt as the environment changes. It is important to be willing to change your own mindset to accommodate paradigm shifts in order to remain useful. This flexibility may slow achievement of short-term goals, but is the only way we have found to stay relevant. Our original intention was not to write file readers and writers, but there was no other way to make functionality in PySAL available to

[11] The original pull request, with discussion and progress made for the contribution, is available at: https://github.com/pysal/esda/pull/58

a wider audience. Neither were we enthusiastic about the work required to become compatible with Python 3. But, the rest of the ecosystem was moving in that direction, and ignoring it would have relegated the project to obsolescence; even the move to a federated model required a lot of additional developer time that could have been spent implementing new features. Flexibility can be expensive to attain, but it is a valuable investment for the future. We do not know what the scientific computing world will look like in ten years. But, as long as Python is playing a key role, we would like PySAL to continue contributing the spatial analytic layer to its larger ecosystem. We are sure that ensuring this contribution continues will take time, effort, and adaptation.

Authors' Contributions. SJR directs and is co-founder of the PySAL project, and leads **pysal** and co-maintains **libpysal**, **esda**, **inequality**, **mapclassify**, **giddy**, and **pointpats**. LA is co-founder of PySAL and leads **spreg**. PA maintains **spreg**. DAB leads **notebooks** and co-maintains **libpysal** and **splot**. RXC co-maintains **segregation**. JDG leads **spaghetti**. WK leads **giddy** and co-maintains **pointpats**. EK co-maintains **pysal** and **segregation**. ZL co-maintains **mgwr**. SL leads **splot**. TMO leads **spint** and **splgm**. HS co-maintains **pointpats**. LJW leads **spvcm**, and co-maintains **pysal**, **libpysal**, **mapclassify**, and **esda**.

References

1. Allen, C. and Mehler, D. M. A. (2019). Open science challenges, benefits and tips in early career and beyond. *PLOS Biology*, 17(5):e3000246.
2. Amaral, P. V., Anselin, L., and Arribas-Bel, D. (2013). Testing for spatial error dependence in probit models. *Letters in Spatial and Resource Sciences*, 6(2):91–101.
3. Anselin, L. (1995). Local indicators of spatial association-LISA. *Geographical Analysis*, 27(2):93–115.
4. Anselin, L. and Rey, S. (2014). *Modern Spatial Econometrics in Practice: A Guide to GeoDa, GeoDaSpace and PySAL*. GeoDa Press, Chicago.
5. Arribas-Bel, D. (2019). A course on geographic data science. *The Journal of Open Source Education*, 2(14).
6. Arribas-Bel, D., de Graaff, T., and Rey, S. J. (2017). *Looking at John Snow's Cholera Map from the Twenty First Century: A Practical Primer on Reproducibility and Open Science*, pages 283–306. Springer International Publishing, Cham.
7. Barthélemy, M. (2011). Spatial networks. *Physics Reports*, 499(1-3):1–101.
8. Batty, M. (2013). *The New Science of Cities*. The MIT Press.
9. Bivand, R. and Piras, G. (2015). Comparing Implementations of Estimation Methods for Spatial Econometrics. *Journal of Statistical Software*, 63(18):1–36.
10. Bivand, R. S., Pebesma, E., and Gomez-Rubio, V. (2013). *Applied spatial data analysis with R, Second edition*. Springer, NY.
11. Browne, W. and Draper, D. (2006). A comparison of Bayesian and likelihood-based methods for fitting multilevel models. *Bayesian Analysis*, 1(3):473–514.
12. Burt, J. B., Demirtaş, M., Eckner, W. J., Navejar, N. M., Ji, J. L., Martin, W. J., Bernacchia, A., Anticevic, A., and Murray, J. D. (2018). Hierarchy of transcriptomic specialization across human cortex captured by structural neuroimaging topography. *Nature neuroscience*, 21(9):1251.
13. Cleveland, W. S. and Devlin, S. J. (1988). Locally weighted regression: An approach to regression analysis by local fitting. *J. Am. Stat. Assoc.*, 83(403):596–610.
14. Cortes, R. X., Rey, S., Knaap, E., and Wolf, L. J. (2019). An open-source framework for non-spatial and spatial segregation measures: the PySAL segregation module. *Journal of Computational Social Science*, pages 1–32.
15. Cottam, J. A. and Lumsdaine, A. (2012). Spatial autocorrelation-based information visualization evaluation. In *Proceedings of the 2012 BELIV Workshop: Beyond Time and Errors - Novel Evaluation Methods for Visualization*, BELIV '12, pages 8:1–8:8, New York, NY, USA. ACM.
16. Ducruet, C. and Beauguitte, L. (2014). Spatial Science and Network Science: Review and Outcomes of a Complex Relationship. *Networks and Spatial Economics*, 14(3-4):297–316.
17. Duque, J. C., Laniado, H., and Polo, A. (2018). S-maup: Statistical test to measure the sensitivity to the modifiable areal unit problem. *PLOS ONE*, 13(11):e0207377.
18. Edelsbrunner, H. and Mücke, E. P. (1994). Three-dimensional Alpha Shapes. *ACM Trans. Graph.*, 13(1):43–72.
19. Efron, B. and Hastie, T. (2016). *Computer Age Statistical Inference*, volume 5. Cambridge University Press.
20. Fan, Y., Zhu, X., She, B., Guo, W., and Guo, T. (2018). Network-constrained spatio-temporal clustering analysis of traffic collisions in jianghan district of wuhan, china. *PLoS one*, 13(4):e0195093.
21. Felkner, J. S. and Townsend, R. M. (2011). The Geographic Concentration of Enterprise in Developing Countries. *The Quarterly Journal of Economics*, 126(4):2005–2061.
22. Ferguson, T. W. and Tamburello, J. A. (2015). The natural environment as a spiritual resource: A theory of regional variation in religious adherence. *Sociology of Religion*, 76(3):295–314.
23. FOSTER (2014). Open Science Taxonomy. Technical report, FOSTER Open Science.
24. Fotheringham, A. S., Brunsdon, C., and Charlton, M. (2002). *Geographically Weighted Regression: The Analysis of Spatially Varying Relationships*. Wiley.
25. Fotheringham, A. S. and O'Kelly, M. E. (1989). *Spatial Interaction Models:Formulations and Applications*. Kluwer Academic Publishers.

26. Fotheringham, A. S., Yang, W., and Kang, W. (2017). Multiscale Geographically Weighted Regression (MGWR). *Annals of the American Association of Geographers*, (6):1247–1265.
27. Gaboardi, J. D., Folch, D. C., and Horner, M. W. (2019). Connecting Points to Spatial Networks: Effects on Discrete Optimization Models. *Geographical Analysis*, 0:1–24.
28. Gaboardi, J. D., Laura, J., Rey, S., Wolf, L. J., Folch, D. C., Kang, W., Stephens, P., and Schmidt, C. (2018). pysal/spaghetti.
29. Gahegan, M. (1999). Guest Editorial: What is Geocomputation? *Transactions in GIS*, 3:203–206.
30. Gahegan, M. (2018). Our GIS is too small. *The Canadian Geographer / Le Géographe canadien*, 62(1):15–26.
31. Gelman, A. and Hill, J. (2006). *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press.
32. Heilmayr, R. and Lambin, E. F. (2016). Impacts of nonstate, market-driven governance on chilean forests. *Proceedings of the National Academy of Sciences*.
33. Hox, J. J., Moerbeek, M., and van de Schoot, R. (2010). *Multilevel Analysis: Techniques and Applications, Second Edition*. Routledge, New York, 2 edition edition.
34. Hughes, C., Naik, V. S., Sengupta, R., and Saxena, D. (2014). Geovisualization for cluster detection of hepatitis a & e outbreaks in ahmedabad, gujarat, india. In *Proceedings of the Third ACM SIGSPATIAL International Workshop on the Use of GIS in Public Health*, pages 39–44. ACM.
35. Ingram, M. C. and Harbers, I. (2019). Spatial tools for case selection: Using lisa statistics to design mixed-methods research. *Political Science Research and Methods*, pages 1–17.
36. Jakubska-Busse, A., Janowicz, M., Ochnio, L., and Ashbourn, J. (2018). Pickover biomorphs and non-standard complex numbers. *Chaos, Solitons & Fractals*, 113:46–52.
37. Jendryke, M. and McClure, S. C. (2019). Mapping crime - hate crimes and hate groups in the usa: A spatial analysis with gridded data. *Applied Geography*, 111:102072.
38. Joo, Y. (2017). Spatiotemporal study of elderly suicide in Korea by age cohort. *Public Health*, 142:144 – 151.
39. Kang, W., Rey, S., Stephens, P., Malizia, N., Wolf, L. J., Lumnitz, S., Gaboardi, J. D., Laura, J., Schmidt, C., Knaap, E., and Eschbacher, A. (2019). pysal/giddy: giddy 2.2.2.
40. Kang, W. and Rey, S. J. (2019a). Inference for income mobility measures in the presence of spatial dependence. *International Regional Science Review*, pages 1–30.
41. Kang, W. and Rey, S. J. (2019b). Smoothed estimators for markov chains with sparse spatial observations. *Geographical Analysis*.
42. Knuth, D. E. (1984). Literate programming. *The Computer Journal*, 27(2):97–111.
43. Kruchten, P., Nord, R. L., and Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *Ieee software*, 29(6):18–21.
44. Kwakkel, J. H., Carley, S., Chase, J., and Cunningham, S. W. (2014). Visualizing geo-spatial data in science, technology and innovation. *Technological Forecasting and Social Change*, 81:67 – 81.
45. Lacombe, D. J. and McIntyre, S. G. (2016). Local and global spatial effects in hierarchical models. *Applied Economics Letters*, 23(16):1168–1172.
46. Lam, S. K., Pitrou, A., and Seibert, S. (2015). Numba: A LLVM-based Python JIT Compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, pages 7:1–7:6, New York, NY, USA. ACM.
47. Lumnitz, S., Arribas-Bel, D., Cortes, R. X., Gaboardi, J. D., Greiss, V., Oshan, T. M., Wolf, L., and Rey, S. (2019). *The Journal of Open Source Software*.
48. Manduca, R. and Sampson, R. J. (2019). Punishing and toxic neighborhood environments independently predict the intergenerational social mobility of black and white children. *Proceedings of the National Academy of Sciences*, 116(16):7772–7777.
49. Merton, R. K. (1968). The matthew effect in science: The reward and communication systems of science are considered. *Science*, 159(3810):56–63.
50. Nielsen, L. H. (2019). Software citations now available in zenodo.
51. Noorbakhsh, J., Farahmand, S., Soltanieh-ha, M., Namburi, S., Zarringhalam, K., and Chuang, J. (2019). Pan-cancer classifications of tumor histological images using deep learning. *bioRxiv*.

52. Nourian, P., Ohori, K. A., and Martinez-Ortiz, C. (2018). Essential means for urban computing: Specification of web-based computing platforms for urban planning, a hitchhiker's guide. *Urban Planning*, 3(1):47–57.
53. Okabe, A. and Sugihara, K. (2012). *Spatial Analysis along Networks*. John Wiley & Sons, Ltd.
54. Oshan, T., Li, Z., Kang, W., Wolf, L., and Fotheringham, A. (2019). mgwr: A python implementation of multiscale geographically weighted regression for investigating process spatial heterogeneity and scale. 8(6):269.
55. Oshan, T. M. (2016). A primer for working with the Spatial Interaction modeling (SpInt) module in the python spatial analysis library (PySAL). *REGION*, 3(2):R11–R23.
56. Ozturk, D., Chaudhary, A., Votava, P., and Kotfila, C. (2016). Geonotebook: Browser based interactive analysis and visualization workflow for very large climate and geospatial datasets. In *AGU Fall Meeting Abstracts*.
57. Perrow, C. (2011). *Normal accidents: Living with high risk technologies-Updated edition*. Princeton university press.
58. Peters, T. (2010). The zen of python. In *Pro Python*, pages 301–302. Springer.
59. Piwowar, H. A., Day, R. S., and Fridsma, D. B. (2007). Sharing Detailed Research Data Is Associated with Increased Citation Rate. *PLOS ONE*, 2(3):e308.
60. Piwowar, H. A. and Vision, T. J. (2013). Data reuse and the open data citation advantage. *PeerJ*, 1:e175.
61. Rey, S., Kang, W., Shao, H., Wolf, L. J., Seth, M., Gaboardi, J. D., and Arribas-Bel, D. (2019). pysal/pointpats: pointpats 2.1.0.
62. Rey, S. J. (2001). Spatial empirics for economic growth and convergence. *Geographical Analysis*, 33(3):195–214.
63. Rey, S. J. (2004). Spatial analysis of regional income inequality. In Goodchild, M. and Janelle, D., editors, *Spatially Integrated Social Science: Examples in Best Practice*, pages 280–299. Oxford University Press, Oxford.
64. Rey, S. J. (2009). Show me the code: spatial analysis and open source. *Journal of Geographical Systems*, 11(2):191–207.
65. Rey, S. J. (2014). Rank-based Markov chains for regional income distribution dynamics. *Journal of Geographical Systems*, 16(2):115–137.
66. Rey, S. J. (2016). Space–time patterns of rank concordance: Local indicators of mobility association with application to spatial income inequality dynamics. *Annals of the American Association of Geographers*, 106(4):788–803.
67. Rey, S. J. (2019). Pysal: the first 10 years. *Spatial Economic Analysis*, 0(0):1–10.
68. Rey, S. J. and Anselin, L. (2007). PySAL: A Python library of spatial analytical methods. *The Review of Regional Studies*, 37(1):5–27.
69. Rey, S. J., Anselin, L., Li, X., Pahle, R., Laura, J., Li, W., and Koschinsky, J. (2015). Open Geospatial Analytics with PySAL. *ISPRS International Journal of Geo-Information*, 4(2):815–836.
70. Rey, S. J., Arribas-Bel, D., and Wolf, L. J. (2021, *under contract*). *Geographic Data Science with Python and the PyData Stack*. CRC Press, Boca Raton, FL.
71. Rey, S. J. and Montouri, B. D. (1999). US regional income convergence: a spatial econometric perspective. *Regional studies*, 33(2):143–156.
72. Rey, S. J. and Smith, R. J. (2013). A spatial decomposition of the Gini coefficient. *Letters in Spatial and Resource Sciences*, 6:55–70.
73. Rey, S. J., Stephens, P., and Laura, J. (2017). An evaluation of sampling and full enumeration strategies for fisher jenks classification in big data settings. *Transactions in GIS*, 21(4):796–810.
74. Roth, R. E., Woodruff, A. W., and Johnson, Z. F. (2010). Value-by-alpha Maps: An Alternative Technique to the Cartogram. *The Cartographic Journal*, 47(2):130–140.
75. Roy, J. R. and Thill, J.-C. (2003). Spatial interaction modelling. 83(1):339–361.
76. Seabold, S. and Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*.
77. Singleton, A. D., Spielman, S., and Brunsdon, C. (2016). Establishing a framework for open geographic information science. *International Journal of Geographical Information Science*, 30(8):1507–1521.

78. Spiridon, L. and Minh, D. D. (2017). Hamiltonian monte carlo with constrained molecular dynamics as gibbs sampling. *Journal of Chemical Theory and Computation*, 13(10):4649–4659.
79. Theodoridis, S., Nogués-Bravo, D., and Conti, E. (2019). The role of cryptic diversity and its environmental correlates in global conservation status assessments: Insights from the threatened bird's-eye primrose (primula farinosa l.). *Diversity and Distributions*, 25(9):1457–1471.
80. van der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30.
81. van Lieshout, M. N. M. and Baddeley, A. J. (1996). A nonparametric measure of spatial interaction in point patterns. *Statistica Neerlandica*, 50(3):344–361.
82. van Rossum, G. (1989). Glue it all together with Python. In *OMG-DARPA-MCC Workshop on Compositional Software Architecture*. CNRI.
83. Vaz, E., Miki, J., de Noronha, T., and Cusimano, M. (2017). New methods for resilient societies: The geographical analysis of injury data. *Journal of Spatial and Organizational Dynamics*, 5(1):12–26.
84. Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Jarrod Millman, K., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C., Polat, I., Feng, Y., Moore, E. W., Vanderplas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and Contributors, S. . . (2019). SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python. *arXiv e-prints*, page arXiv:1907.10121.
85. Wilson, A. G. (1971). A family of spatial interaction models, and associated developments. 3:1–32.
86. Wolf, L. J., Knaap, E., and Rey, S. (2019a). Geosilhouettes: Geographical measures of cluster fit. *Environment and Planning B: Urban Analytics and City Science*, page 2399808319875752.
87. Wolf, L. J., Rey, S. J., and Oshan, T. M. (2019b). Open code is not enough: towards a replicable future for geographic data science. Position paper for the Third Geospatial Software Institute Workshop on Strategic Planning and Governance.
88. Wood, S. N. (2006). *Generalized Additive Models: An Introduction with R*. CRC Press, Boca Raton.