[CP-05-027] GIS&T and Computational Notebooks

Abstract

Researchers and practitioners across many disciplines have recently adopted computational notebooks to develop, document, and share their scientific workflows—and the GIS community is no exception. This chapter introduces computational notebooks in the geographical context. It begins by explaining the computational paradigm and philosophy that underlie notebooks. Next it unpacks their architecture to illustrate a notebook user's typical workflow. Then it discusses the main benefits notebooks offer GIS researchers and practitioners, including better integration with modern software, more natural access to new forms of data, and better alignment with the principles and benefits of open science. In this context, it identifies notebooks as the "glue" that binds together a broader ecosystem of open source packages and transferable platforms for computational geography. The chapter concludes with a brief illustration of using notebooks for a set of basic GIS operations. Compared to traditional desktop GIS, notebooks can make spatial analysis more nimble, extensible, and reproducible and have thus evolved into an important component of the geospatial science toolkit.

Keywords: computational notebook, computing infrastructures, Geospatial Computing, open, open science, replicability, reproducibility

Author & citation

Boeing, G., and Arribas-Bel, D. (2020). GIS and Computational Notebooks. The Geographic Information Science & Technology Body of Knowledge (1st Quarter 2021 Edition), John P. Wilson (Ed.). DOI: 10.22224/gistbok/2021.1.2.

Explanation

1. Definitions
2. Introduction
3. How Computational Notebooks Work
4. Open-Notebook GIS
5. Notebooks in Action
6. Conclusion

1. Definitions

Computational Notebook: a computer file containing code, output, images, and narrative text woven together.

Interactive Scientific Computing: a method of executing code nonlinearly with interactive user input to develop and run scientific workflows.

Jupyter Notebook: an interactive computing environment comprising a web browser, a notebook server, a notebook file, and a kernel.

JupyterLab: the notebook user interface in the Jupyter ecosystem, rendered by a web browser.

Open Science: a movement to make the workflows and findings of scientific research more accessible, transparent, and reproducible.

Open Source Software: software released under a license that allows users to freely examine, modify, and redistribute its source code.

Transferable Platform: a complete set of underlying software infrastructure to allow code to run consistently and reproducibly across different hardware and users.

2. Introduction

A computational notebook is a computer file that contains code, output, images, and narrative text woven together. Notebooks allow users to consolidate their analytics workflows, blending code, documentation, and results into a single reproducible and distributable file. They also enable interactive computing in the literate programming paradigm. By demonstrating the relevance of these concepts—and this approach—to both research and teaching, this chapter positions computational notebooks as a key emerging tool in the GIS landscape and discusses their advantages for geospatial analysts.

Today many geospatial scholars and practitioners use notebooks to load, clean, filter, analyze, visualize, and model spatial data, as well as to share their work flows and findings with peers and broader audiences. Recent years have witnessed rapid adoption of computational notebooks among data scientists and instructors across disciplines like biology, astronomy, economics, and geography (Perkel, 2018). These tools have also grown ubiquitous in industry as companies like Google and Airbnb have adopted them for analytics work. To understand this methodological shift, we must consider notebooks' capabilities and the value they create for analysts, researchers, teachers, and students.

3. How Computational Notebooks Work

3.1 The Paradigm

Scientific researchers and analysts historically used lab notebooks to record their workflow's questions, hypotheses, data, models, results, and all the various analytical decisions made along the way. This was critical for organizing research activities and documenting all the "whats," "whys," and "hows" of the scientific process for recollection, replication, and re-analysis. But as graphical user interface (GUI) based analytics software became more common in the late 20th century, many of these details were lost to point-and-click software and the rationales underpinning analytical decisions became obfuscated.

To digitally emulate the benefits of traditional lab notebooks, Mathematica first developed the computational notebook interface in the 1980s as a closed source commercial tool for scientists (Somers, 2018). During the 2010s, open source notebook development, spearheaded by the Jupyter project, expanded throughout research and practice by

supporting popular languages in the open source and open science communities such as Python, R, and Julia. Today, computational notebooks mimic traditional lab notebooks digitally and enhance them through two computational paradigms: literate programming and interactive computing.

To explain these two paradigms, we can contrast them with a traditional computer program which consists of lines of code and optional inline comments and is executed linearly from beginning to end. In literate programming, a computer program instead consists of both code and natural language narratives woven together to explain and document the logic of the program (Knuth, 1992). In interactive computing, a computer program interacts in real-time with its user and these interactions shape its execution flow (Pérez and Granger, 2007). Thus, while a traditional program runs through its code in order, line-by-line, a computational notebook can be executed nonlinearly and can include natural language narratives documenting and explaining each “chunk” of code alongside its results and output.

Many different kinds of computational notebooks exist today for many different programming languages. It is important to note that there are both general characteristics of notebooks and, separately, individual implementations’ own sets of features and elements. Some notebook implementations emphasize code with annotations, as discussed above, while others are more plain-text oriented—but nearly all of these various implementations share a set of common features. As the Jupyter notebook has become the most prominent, we will focus on it as an illustrative example.

3.2 Notebook Architecture

A Jupyter notebook’s architecture comprises four components: a web browser, a notebook server, a notebook file, and a kernel (Kluyver et al., 2016). As illustrated in Figure 1, the user relies on a web browser to interact with the notebook server, which reads the notebook file and returns it to the browser to render as a web page. This user interface rendered in the web page includes individual cells—sections of either code or formatted text—in which the user can interactively type. A code cell can be executed by the user: the browser sends the code in a cell to the server, which routes it to the kernel, which runs the code and returns the result to the browser via the notebook server. The browser renders this result inline in the notebook beneath the code cell. Thus, computational output such as individual calculations, tables, or figures appears beside the code that generated it.

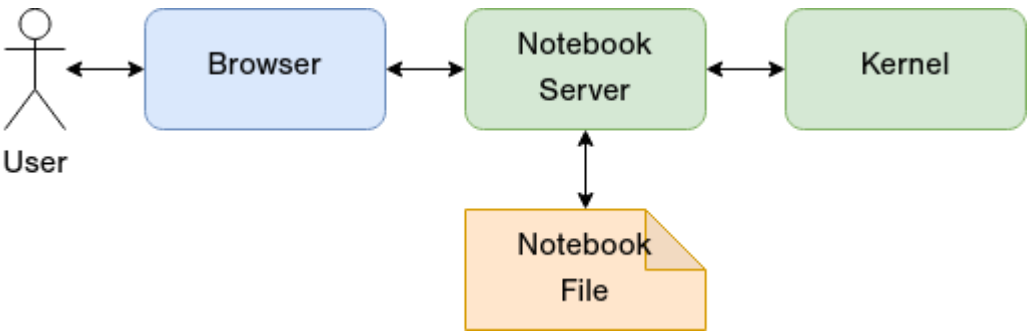


Figure 1. Architecture of the Jupyter notebook computing environment. Source: authors.

Computational notebooks save, store, and present the resulting output of their code within the notebook itself. By integrating these elements, they blend what the code is (code cells) with what the code does (narrative) and with what the code produces (output). The notebook becomes a “one-stop shop” that allows the user to focus what the code is doing and why and how it is doing it (à la literate programming), as well as to explore and tinker with it while immediately seeing how those changes translate into different results (à la interactive computing).

Another important feature is notebooks’ distributed architecture. The notebook server could be running on your own computer, in a server room down the hall, or even on the other side of the world—yet you have easy access to it through any web browser. The server itself is language-agnostic, but notebook kernels are language-specific. Hundreds of Jupyter kernels exist for dozens of different programming languages. The most popular languages for data science and spatial analysis are all supported, including Python, R, and Julia.

3.3 Notebook Usage

In the Jupyter ecosystem, the user interface in the web browser is called JupyterLab. JupyterLab shares many common elements with other computational notebooks. It primarily consists of a main work area and a sidebar to browse files and running kernels, as illustrated in Figure 2. The main work area displays the currently open notebooks. Here, notebooks can be created, edited, and executed. Users can add or remove notebook cells, move cells around to reorder their execution, type code or markdown text into cells, or execute one or more cells.

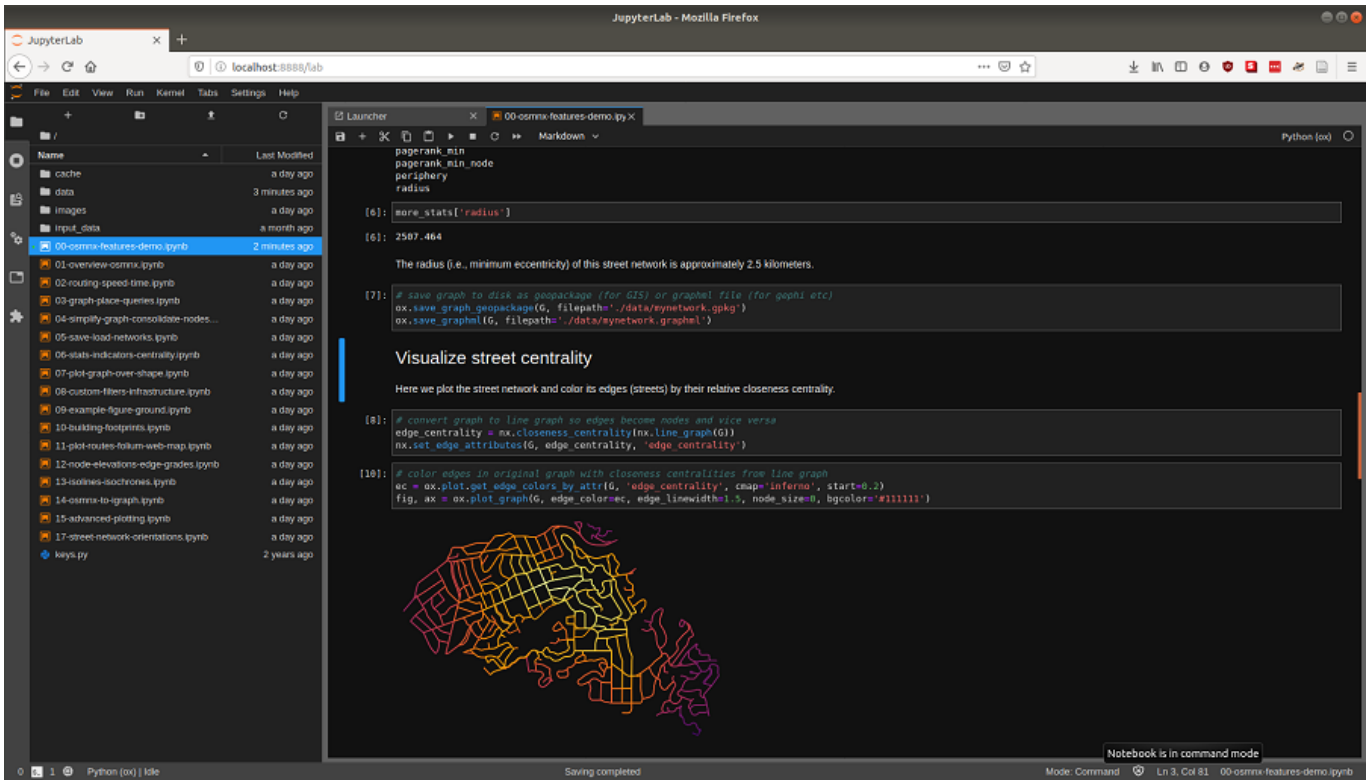


Figure 2. The JupyterLab notebook interface. Source: authors.

Computational notebooks are increasingly used today in pedagogy, research, and practice. Instructors use them to introduce students to coding and data science because they can show the results of each computation, step-by-step, and explain each new language detail along the way (Reades, 2020). Researchers use them to document, explain, and visualize their questions, hypotheses, data, experiments, and results alongside their analytics code (Perkel, 2018). Software developers use them to provide visual, narrative usage examples and demonstrations of their software packages for newcomers to learn how they work (Boeing, 2020b).

All of a computational notebook's code, text, and multimedia content is stored in a single notebook file, so they can easily be shared and distributed, and they work well in collaborative version control systems. Because the specification used to store notebooks is open source and text-based, a wide ecosystem of tools exists to convert between the notebook file format and other formats used in scientific work (e.g., PDF, markdown, LaTeX, Microsoft Word).

4. Open-Notebook GIS

Computational notebooks are not specific to GIS. However, the nature of GIS work makes notebooks an essential building block for modern geospatial workflows. Although their domain of use is much wider and spans almost every branch of (interactive) scientific computing, several of their features fit particularly well with existing, well-documented needs in the GIS discipline. If computational notebooks did not exist, the GIS community would have to invent them. Their necessity derives from three core aspects of modern GIS that notebooks address natively: 1) the distributed nature of modern GIS software, 2) the increasingly messy nature of spatial data, and 3) emerging trends in open science.

4.1 Software Ecosystem

The GIS software ecosystem has changed dramatically in recent years (Arribas-Bel and Reades, 2018). Until the early 2000s, the desktop had remained the standard platform (Gahegan, 2018). GIS's desktop-centric paradigm favored all-encompassing software covering as much functionality as possible and exposed it in a simple manner for non-technical audiences. Several GIS industry subdomains still operate in this paradigm today (e.g., local government, military).

However, most of the research community that uses and expands the GIS domain, as well as a nascent industry community around geographic data science, have transitioned to a different model. This new framework employs a distributed and decentralized approach to software production and usage, centered on coding and open source packages. In this modern context, notebooks provide an ideal tool to transparently tie the ecosystem together, because they are built around coding rather than point-and-click desktop GUIs and provide explicit mechanisms to detail the "whats," "whys," and "hows" of analytics.

4.2 Data Ecosystem

Since the early 2000s, the geospatial data ecosystem has also been radically redefined. Traditional data sources available to geospatial analysts (e.g., decadal censuses, official surveys, limited remote sensing) have been augmented by entirely new kinds of data from sensors such as smartphones, video camera feeds, drones, and nanosatellites. These data are not just “more” or “bigger”—they differ fundamentally from traditional data (Kitchin andMcArdle, 2016). Furthermore, unlike many traditional data sources, they were not explicitly designed for research and rarely come packaged as tidy, organized datasets. Transforming them requires particular effort and attention to render them usable for analytics (Harris et al., 2017; Singleton and Arribas-Bel, 2019). In this context, notebooks integrate the data science tools required to wrangle and transform these data, empowering analysts to document all the decisions and steps in the process of turning raw inputs into an analysis-ready dataset.

4.3 GIS and Open Science

The GIS world has recently engaged in broader discussions concerning open science and reproducible research (Brunsdon, 2016; Kedron et al., 2019, 2020). Recent scandals regarding a lack of transparency and reproducibility have generated attention across the scientific world. In response, reform efforts have coalesced around the notion of open science—which promotes transparent, well-documented, and publicly disseminated research—so third parties can fully understand and replicate the workflows (Boeing, 2020a; Koster and Rowe, 2020; Wilson et al., 2020; Poorthuis and Zook, 2019; Rey, 2009).

In GIS, reproducibility should also consider the (too often ignored) underlying technology supporting the research. Although early GIS work was mostly mathematical—and thus easily fully documented in journal articles—the evolution of GUI-based desktop GIS made it difficult to maintain a close correspondence between the numerous operations carried out on the researcher’s computer and the final results presented in an article. Computational notebooks present an alternative where transparency and reproducibility are built-in, nudging researchers toward open science.

“Open-notebook GIS” ties together modern geospatial software and data ecosystems for open science. Notebooks help integrate the modern geospatial software landscape although they are not geospatial software themselves. They help leverage new forms of data whose availability and accessibility relies on modern technologies, such as application programming interfaces (APIs) or distributed databases. In sum, they open up new research pathways while encouraging analytical transparency, documentation, and reproducibility.

4.4 Notebooks are Not Enough

As important as computational notebooks are becoming to modern GIS, they alone are necessary but insufficient. Notebooks can rather be seen as the “glue” that ties together the various distributed components that make up the modern geospatial scientific stack. This broader landscape of modern GIS—of which computational notebooks are a core component— has two more key building blocks: open source packages and transferable platforms.

First, open source packages are the main vehicle through which scientific software in general (and GIS in particular) is currently distributed. Packages are compilations of code that allow users to reuse and reapply their functionality in a variety of contexts. Open source refers to the license and set of rights that are granted to the user, which typically

include examining, modifying, and redistributing the code that makes up the package. Open source packages offer more modular and flexible ways of distributing software than traditional desktop GIS—albeit with an expectation that the user is comfortable writing at least some code. The open source model is more efficient and agile at incorporating new technologies and supporting a wider variety of use cases. In this model, computational notebooks integrate different packages and provide a natural interface that replaces the GUI of traditional desktop GIS.

Second, transferable platforms refer to the broader lower-level set of software and infrastructure required to execute code in a way that is easy to transfer across hardware and users. Modern scientific software stacks are complex and delicate. They rely on many interconnected components that must be installed in a specific way (i.e., version, compiler, etc.) to maintain compatibility with every other piece of the whole. Building and replicating these environments is non-trivial and can hinder reproducibility. The motivation behind transferable platforms is to develop tools and practices that make it easier to distribute these complex stacks in reliable ways. No single standard method for transferable platforms currently exists, but a set of connected components are emerging around package managers like conda and containerization technologies like Docker.

Package managers take care of solving complex package inter-dependencies to ensure the entire stack can function without one package breaking the functionality of another due to version incompatibilities. This situation may seem unlikely or irrelevant, but is in fact a widespread challenge that consumes significant time for data scientists who must ensure packages work well with each other before they can start analyzing data.

Containerization allows users to take a complete software environment that works in one particular context (e.g., a data scientist’s laptop), copy it, and then run it elsewhere isolated in a “container.” Containers ensure all package versions and configurations are the same irrespective of hardware (e.g., laptop, server, or the cloud) and user. In this context, computational notebooks provide the main user interface to a pre-built, fully compartmentalized platform. Containerization makes it easier to distribute transferable platforms in more accessible and user-friendly ways.

5. Notebooks in Action

Today, the geographic data science ecosystem is most robust in R (particularly the r-spatial community) and Python, where many packages exist to support spatial analysis and modeling such as geopandas (geospatial data handling), PySAL (advanced spatial analytics and modeling), matplotlib (data visualization), OSMnx (street network modeling and analytics), cenpy (working with US Census Bureau data), contextily (basemap tiles), folium (web mapping), and many more: see the Additional Resources section for links. These tools allow geospatial data scientists to fully replace traditional desktop GIS software with reproducible, universal analytics workflows in computational notebooks.

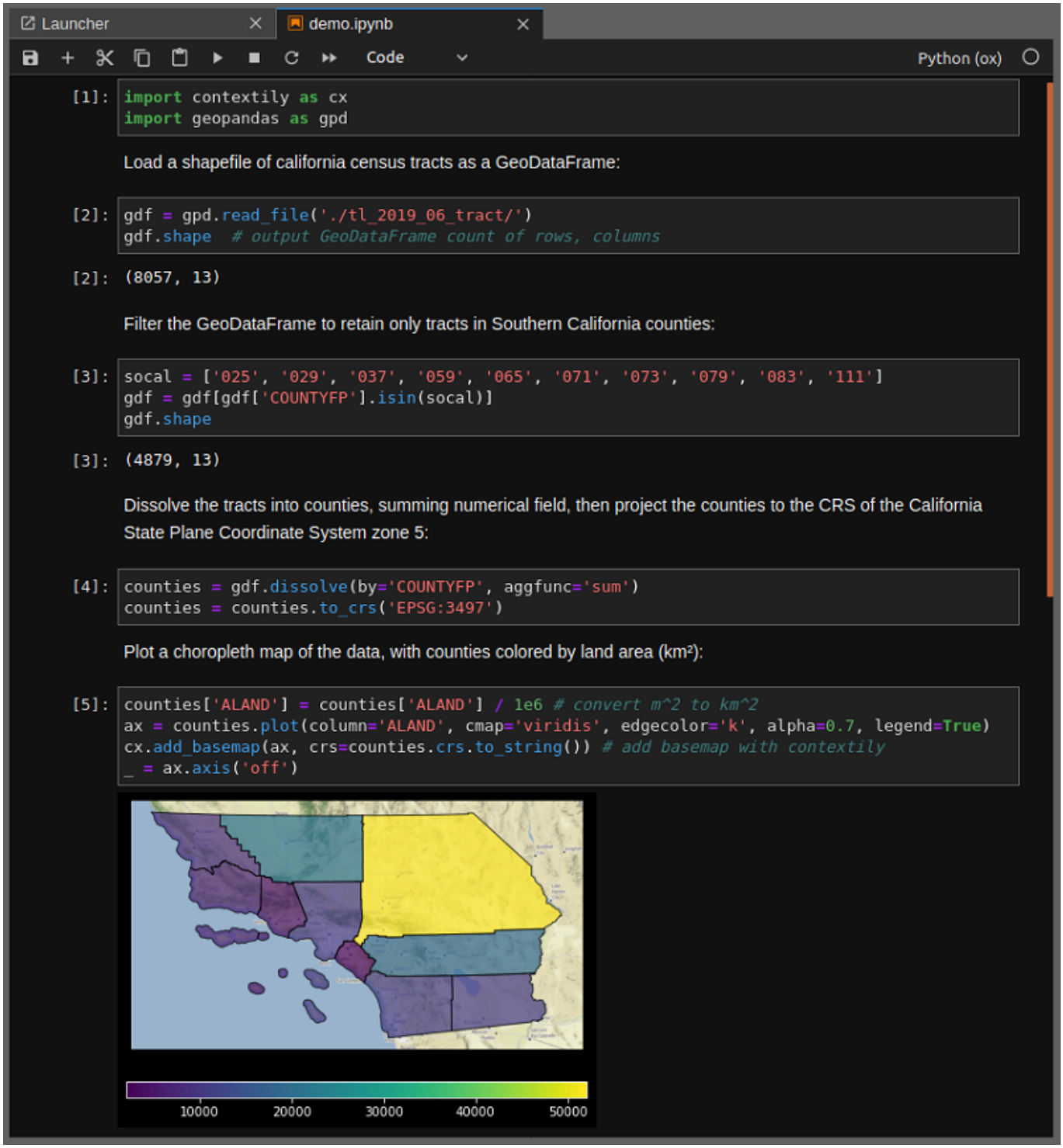


Figure 3. Simple real-world usage example that demonstrates basic GIS in a computational notebook, using JupyterLab and Python. Source: authors.

Figure 3 presents a real-world usage example using JupyterLab and Python to perform a set of simple GIS operations in a computational notebook. Computational notebooks are not tied to Python—we could perform this example in any programming language that includes such capabilities. Rather, it is the notebook architecture and interactive analytics that matter here.

In cell 1, we import two packages for geospatial analysis and mapping. In cell 2, we read an Esri shapefile containing all the census tracts in California, turn it into a GeoDataFrame, and output the shape of this resulting GeoDataFrame. We have loaded 8,057 rows (i.e., census tracts) and 13 columns (i.e., variables). Between these two code cells, we see a rendered markdown cell containing a short comment to elucidate the workflow. Markdown provides a simple mechanism for including formatted text such as headings, equations, tables, and hyperlinks. The data in this example are read from a local file, but the code would look similar were we instead to read it directly from a remote location (e.g., a file server such as Amazon Web Services' Landsat 8 satellite imagery catalogue) or to query a web service such as the US Census Bureau API (e.g., with cenpy).

In cell 3, we create a list of all the county IDs in Southern California, then filter the GeoDataFrame to only retain tracts whose county IDs appear in that list. Outputting the shape of the resulting GeoDataFrame reveals that we have retained just 4,879 of the original 8,057 tracts. In cell 4, we dissolve and project the GeoDataFrame. First we aggregate the tracts up to the county level with a standard spatial dissolve operation and sum their numerical attributes to get new county totals. Then we project the GeoDataFrame from its original coordinate reference system (as defined by the shapefile we loaded) to a new one representing the meter-based California State Plane Coordinate System zone 5. Finally, in cell 5, we convert the land area column from square meters to square kilometers, plot a basic choropleth map of the counties colored by land area, and add a basemap.

This notebook can be executed linearly like a script by running it from the top-down, or it can be executed nonlinearly by a user choosing individual cells to run one or multiple times in an arbitrary order. Given the interactive paradigm, a cell or cells can be run once or many times repeatedly, and cells may be run in any order the user desires. Due to this possibly nonlinear flow of execution, it is important to periodically restart the kernel and run all cells to ensure that objects are defined and used in the expected order of operations.

6. Conclusion

The open science movement promotes transparency and reproducibility in the workflows underlying data analyses. Traditional desktop GIS's reliance on GUIs makes the software easy to learn and use, but difficult to fully document, understand, and replicate complex spatial analysis projects. Computational notebooks offer an alternative paradigm for scientists, educators, and analysts across disciplines—including quantitative geography. On one hand, notebooks' reliance on code creates a steeper learning curve. On the other hand, they lend themselves much more readily to transparency, reproducibility, documentation, dissemination, and modern data science workflows.

The interested reader is encouraged to explore the Additional Resources section below for "next steps" getting started with geospatial analytics in computational notebooks.

References

Arribas-Bel, D., and Reades, J. (2018). Geography and computers: Past, present, and future. Geography Compass.

Boeing, G. (2020a). The Right Tools for the Job: The Case for Spatial Science Tool-Building. Transactions in GIS, 24(5), 1299-1314.

Boeing, G. (2020b). Urban Street Network Analysis in a Computational Notebook. Region, 6(3):39–51.

Brunsdon, C. (2016). Quantitative Methods I: Reproducible Research and Quantitative Geography. Progress in Human Geography, 40(5):687–696.

Gahegan, M. (2018). Our GIS Is Too Small. The Canadian Geographer, 62(1):15–26.

Harris, R., O'Sullivan, D., Gahegan, M., Charlton, M., Comber, L., Longley, P., Brunsdon, C.,Malleson, N., Heppenstall, A., Singleton, A., Arribas-Bel, D., and Evans, A. (2017). More Bark than Bytes? Reflections on 21+ Years of Geocomputation. Environment and Planning B: Urban Analytics and City Science, 44(4):598–617.

Kitchin, R. andMcArdle, G. (2016). What makes Big Data, Big Data? Exploring the ontological characteristics of 26 datasets. Big Data & Society, 3(1).

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K.,Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Safia, A., Willing, C., and Jupyter Development Team. (2016). Jupyter Notebooks: A Publishing Format for Reproducible ComputationalWorkflows. In Loizides, F. and Schmidt, B., editors, Positioning and Power in Academic Publishing: Players, Agents and Agendas, pages 87–90. IOS Press, Amsterdam, Netherlands.

Knuth, D. E. (1992). Literate Programming. Center for the Study of Language and Information, Stanford, CA.

Koster, S. and Rowe, F. (2020). Fueling Research Transparency: Computational Notebooks and the Discussion Section. Region, 6(3):E1–E2.

Pérez, F. and Granger, B. E. (2007). IPython: A System for Interactive Scientific Computing. Computing in Science & Engineering, 9(3):21–29.

Perkel, J. M. (2018). Why Jupyter Is Data Scientists' Computational Notebook of Choice. Nature, 563(7729):145–146.

Poorthuis, A. and Zook, M. (2019). Being Smarter about Space: Drawing Lessons from Spatial Science. Annals of the American Association of Geographers, 110(2): 349-359.

Reades, J. (2020). Teaching on Jupyter: Using Notebooks to Accelerate Learning and Curriculum Development. Region, 7(1):21–34.

Rey, S. J. (2009). Show Me the Code: Spatial Analysis and Open Source. Journal of Geographical Systems, 11(2):191–207.

Singleton, A. and Arribas-Bel, D. (2019). Geographic Data Science. Geographical Analysis,

53(1): 61-75.

Somers, J. (2018). The Scientific Paper Is Obsolete. The Atlantic.

Wilson, J. P., Butler, K., Gao, S., Hu, Y., Li, W., and Wright, D. J. (2021). A Five-Star Guide for Achieving Replicability and Reproducibility When Working with GIS Software and Algorithms. Annals of the American Association of Geographers, 111(5): 1311-1317.