Chapter 3: Analytical environments
Roger Bivand

This is a draft chapter/article. The final version is available in Handbook of Spatial Analysis in the Social Sciences edited by Sergio J. Rey and Rachel Franklin, published in 2022, Edward Elgar Publishing Ltd, https://doi.org/10.4337/9781789903942.00010.

Source materials are available from https://github.com/rsbivand/sass22.

Analytical Environments

Roger Bivand

Abstract

Analytical environments are layered abstractions, and may be understood differently both by those belonging or not belonging to them, and by varieties of communities taken as belonging to a given analytical environment. It is helpful to see the role of evolving communities and often symbiotic relationships between participants as a lens that will help us grasp the "life" of analytical environments. This life is also implicit in the experiences of participants, be they users, developers, recipients of research output, students, or providers of key underlying software libraries.

In this chapter, a partly autobiographical approach has been adopted in order to position this reading of how analytical environments may be understood as outcomes of willed actions. These willed actions, to provide software among other things for teaching and research, do not limit themselves to achieving pre-defined goals. In the cases used, Python and R, the reach and influence of the language platforms has evolved greatly over the twenty years work has been progressing. In addition, the communities of users and developers have matured and grown, although the pressing need for bringing in younger and more diverse contributors is recognised.

We also know too little about the impact of these analytical environments, as citation practice with regard to software tools such as Python or R packages has not been encouraging in the social sciences. Going forward, and as part of the movement towards reproducible research, it would be interesting to encourage journal editors to require article authors to document their work by citing the software used. Unfortunately, as of now it is not possible to give a systematic overview of who uses which analytical environment for what, so this chapter will follow the chosen partly autobiographical approach, rendering the claims advanced largely subjective.

Introduction

In conducting applied research, social scientists use a range of toolboxes containing methods. The methods and their allocation to toolboxes evolve over time, building on contributions from social science disciplines and from disciplines beyond the social sciences. An analytical environment in such a setting is associated with one or more toolboxes and may relate to one or more sub-discipline or disciplines. Use of toolboxes may engender collaboration or communication between users and developers of methods, with overlaps between the roles of applied researchers viewing themselves as just users, users expressing needs to developers, and developers focussed chiefly on innovation in methods, among others.

One could also view an analytical environment as a layered community, in which users and developers interact. Interaction may be asymmetrical, with developers (or software vendors) conditioning the choices proposed to users. In scientific research, these choices may be further ordered by reviewers of work for publication, and by thesis committees. It is not at all impossible for domain communities to adopt differing specifications of the conceptualizations

they are using, leading to the choice of different analytical environments, based on the relative distances between nodes in research communities.

This chapter will consider the background to selected analytical environments for spatial analysis in the social sciences, choosing to use the R and Python languages as examples. Attention will be concentrated on empirical quantitative analysis rather than, for example, deterministic and simulation-based modelling, or qualitative methods. The ability to map data and the results of fitting models to data will be used to draw together similar kinds of analytical environments. Stress will be placed on the roles played by the shared open source infrastructure underpinnings in the R and Python communities. This extends to the ways in which communities associated with these analytical environments view the mutual relationships between software components.

Antecedents

Duncan et al. (1961) discussed in some detail the challenges facing social scientists analyzing areal data. The increasing use of areal data by social scientists from the 1950s, such as census and population data, raised numerous questions. These included some topics that are still central, and others which are now more neglected. They also highlighted issues arising in the development of appropriate methods of analysis for areal data, but apologize for their concern with methodology. They acknowledged that most scholars occupy themselves with substantive research questions, conceptual and applied, but few develop and implement appropriate methods. In this context, they pointed to the potential benefits of learning from other disciplines with disparate subject matter. The dust jacket of the book starts with the depressing assertion:

> "Specialists adopting techniques derived for the solution of other sorts of problems than those specifically concerning them share an unhappy lot. Such are those faced with the analysis of the quantitative data of the social sciences having areal significance."

While spatial data encompass more than areal data, such as aggregate census counts within arbitrarily given boundaries, it remains important to develop appropriate methods, and to propose and describe how they should be used in research practice. In the 1950s, some tabulation could be done by machine, but most calculation in the social sciences was done manually. During the 1960s, psychometricians and some social scientists, as well as their statistician colleagues and co-authors, began to be able to access computers, but most methods needed to be implemented as software using available languages such as Fortran and Algol using batch processing, or BASIC, which provided a degree of interactivity when using a teletype terminal.

Once SAS (1972) and SPSS (1968) became available for batch processing on mainframe computers, many university quantitative social scientists were relieved of the burden of building software from scratch, and analytical environments developed around the proprietary systems available. If the required method was not available, researchers still needed to be able to compile from source, but as compilers were always provided with mainframe computers, this was only an additional step, provided source code was available. Note that it was not uncommon at this time for authors to offer access to listings of source code in papers and books (Roger S. Bivand, 2009; Cliff & Ord, 1969).

The entry of minicomputers and microcomputers during the 1970s presented challenges to the batch processing mainframe model of computing in the social sciences. It took time for licensing and software publication business models to catch up with new hardware opportunities. However, libraries of Fortran subroutines for linear algebra (LINPACK, EISPACK) were released to the public domain, initially on magnetic tape, later from FTP archives. Researchers still needed to compile from source, linking to these libraries. A profusion of development environments arose, like MATLAB, which was written to give interactive access to linear algebra functionalities (MATrix LABoratory).

The birth of spreadsheets on the earliest microcomputers before the IBM PC offered an important and direct way for social scientists to handle tabular data. Its importance lay in the immediacy of interaction between the analyst and the data, subsequently mirrored in graphical interface design for many statistical software systems. This perception of the user interface has now been influential for almost forty years despite its weaknesses. A major weakness is that, unless backed by a transactional database or other system, there is no record of changes made to data or instructions using the GUI. Consequently, and unlike the batch processing of the previous two decades, it can be very difficult to reconstruct steps taken in the course of analysis.

In the decades between 1990 and 2010 with the widespread use of personal computers and licensed commercial software, a division of labour was strengthened separating developers and users. In the case of economics and econometrics, Renfro (2009) writes:

> One of the consequences of this specialization has been to introduce an element of user dependence, now grown to a sufficiently great degree that for many economists whichever set of econometric operations can be performed by their choice of program or programs has for them in effect become the universal set (p. 24).

This situation arose over time, and certainly did not characterise the research context when quantitative social science came into being. At that time, graduate students simply regarded learning to program, often in Fortran, as an essential part of their preparation as researchers. This meant that researchers were much "closer" to their tools, and could adapt them to suit their research needs.

Until recently, fewer appear to have felt confident as programmers despite the fact that modern high-level languages such as Matlab, Python, or R are easy to learn and very flexible, and many statistical software applications offer scripting languages (such as SAS, Stata, SPSS). In addition, Python and R can be used without licence costs, and installed cross-platform. Happily, this is changing, including collaboration between R and Python communities (Turner, 2020) to increase diversity and to ensure that all are welcome in rapidly evolving communities.

In this article, a number of pictures will be drawn, hoping to indicate in a non-prescriptive manner the motivations behind and the functioning of the chosen environments for handling and analysing spatial data. First we will consider the handling of spatial data as a prerequisite for analysis, before progressing to environments for analysing spatial data.

Environments for handling spatial data

Bending towards the consideration of autobiography in Holt-Jensen (2019), I have chosen to discuss environments that have informed and shaped my own work (see also Johnston, 2019;

Meeteren, 2019). I learned to program as a PhD student at the London School of Economics in the early 1970s, because I needed to handle and analyse data for my thesis in ways that were not otherwise possible using software available at the University of London Computing Centre.

London, early 1970s

Few social science departments had facilities for technical drawing when books and articles were prepared using typewriters. Human geographers often did have access to map libraries, from which the bases for thematic cartography could be borrowed for tracing. Many articles and books were richly furnished with figures constructed by hand, sometimes by researchers themselves, but often by skilled technical staff working from sketches provided by researchers. While tabular computations could be conducted and line-printer output could be entered into typewritten manuscripts, figures were a much greater problem.

Figure 1 (left panel) shows a typical example of a hand-drafted map showing county boundaries, county letter codes and contiguous neighbour counts used extensively in Cliff & Ord (1973) for testing for spatial autocorrelation. When the figure was prepared, very few researchers irrespective of discipline had access to hardware permitting the input of boundary positions as data, or to hardware for outputting hard copy maps.

I was fortunate to participate in the doctoral programme at the Department of Geography, London School of Economics 1972-1975. We had access to a Wang minicomputer (probably a 2200 model with CRT, using Basic for programming), connected to a large digitizer. This let those of us willing to use considerable time to construct boundaries by recording chosen boundary points. The points were then listed in line sets, and line sets as closed polygon boundary sets; other examples of work using a Wang minicomputer with a digitizer may be found (Jeremíasson, 1976).

Figure 1. Left panel: Counties of the Irish Republic, (Cliff and Ord, 1973, page 54; Right panel: Population density in London, 1971, (Shepherd et al., 1974, page 15.)

Figure 1 (right panel) is typical of the output of choropleth maps with boundary input digitized using the Wang minicomputer (Shepherd et al., 1974). Margaret Jeffery and Hazel O'Hare wrote software called "Chormap" to utilize the digital polygon boundaries with matching census data, to be run in batches on the University of London Computing Centre CDC7600, and output on a CalComp 1670 microfilm recorder on 35mm monochrome film. The association of input and output hardware, together with uniquely skilled technical assistance made it possible for the very few researchers with access to these resources to plot publishable choropleth maps much more efficiently than hand drafting had permitted. These polygon boundaries also permitted the extraction of lists of neighbouring Norwegian census tracts, used in my thesis to calculate Moran's I statistics using self-written software (Roger S. Bivand, 1975).

This analytical environment was people-based, a community of graduate students, technical staff and junior lecturers, who needed to create choropleth maps from census data, and carry out aspatial analyses, for example using SPSS. Any non-standard analyses meant creating software oneself in Basic or Fortran; some shared code was available, and advice from colleagues very helpful.

Geographical information systems and handling spatial data

The subsequent emergence of geographical information systems (GIS) software, and of the adoption of standards for file formats for transferring spatial data, made it easier for social scientists to work where no digitizers were available. The US Census introduced the Topologically Integrated Geographic Encoding and Referencing (TIGER) format (Marx, 1986), and others adopted other ad-hoc standards. The earlier GIS ran only on larger, multi-user computer systems with command line interfaces. During the 1990s, ESRI introduced ArcView, and MapInfo Corporation MapInfo; these were programs with graphical user interfaces for personal computers. They used the ESRI Shapefile and Mapinfo TAB file formats respectively, and both formats used multiple files for a single data collection, and dBase DBF files to hold attribute data.

Typically, digitizer output was specified in planar units, often in the native units of the digitizer itself, or in decimal inches or millimetres. At this stage, coordinate reference systems (CRS) were not seen as essential. This treatment of CRS as unimportant extended to the specification of the ESRI Shapefile (ESRI, 1998), which was only subsequently optionally supplemented with a `*.prj` file containing an ESRI-specific Well Known Text representation.

University geography departments were offered academic licenses for teaching and research; use of GIS was tied to PC labs, and often to dedicated staff who could keep the software operating. Certainly, in the 1990s and 2000s the dominant analytical environment was GIS-based, but while take-up in the environmental sciences was strong, this was seldom the case in the social sciences.

The adoption of GIS in human geography among the social sciences led to accusations that GIS was positivist in its essence and expression (Schuurman, 2000). Consequently, while other empirical and quantitative social sciences found that the ability to handle spatial data in the new common file formats enhanced the range of tasks they could accomplish, human geographers were challenged. The "science wars" begun thirty years ago continue (Thatcher et al., 2016), at least in human geography. This has meant that few human geographers receive any

training in handling spatial data as graduate students or earlier, leaving them disadvantaged when the tools might prove useful.

Moving beyond GIS

It is not however the case that being able to handle spatial data, especially aggregate data, has passed by without application in the social sciences, despite the "science wars." One example is the use of R as an environment for handling and visualizing data, some of which is spatial (Cheshire & Uberti, 2014). One of the authors related in a blog (https://jcheshire.com/visualisation/r-visualisations-design/) that:

> The majority of graphics we produced for London: The Information Capital required R code in some shape or form. This was used to do anything from simplifying millions of GPS tracks, to creating bubble charts or simply drawing a load of straight lines. We had to produce a graphic every three days to hit the publication deadline, so without the efficiencies of copying and pasting old R code, or the flexibility to do almost any kind of plot, the book would not have been possible.

Another pair of examples are two atlases mostly using cartograms to visualize many aggregate variables avoiding the excessive figure area occupied by large sparsely populated areal units (Ballas et al., 2014, 2017). In these cases, the authors do not state which software environment has been used for handling the spatial data beyond the method (Gastner & Newman, 2004), but they provide guidance in an earlier article (Dorling & Ballas, 2011).

These examples demonstrate very active use of spatial data in the social sciences, using scripts and other software components, but not as such directed towards the fostering of analytical environments. They do, however, show that GIS had become too limiting for the creative needs of these projects, leading to movement beyond expecting software to be provided ready-for-use by GIS vendors.

Contemporary scripting environments for handling spatial data

Scripting is not just interactive computing, entering successive commands at a command prompt. It presumes the active use of the language "behind" the prompt, and that the command sequences can be submitted from a file, a script, by analogy with performance. The script contains the sequence of steps needed to reach the intended conclusion.

Scripting and the use of "little languages" had been seen as an alternative to the graphical user interfaces that increasingly dominated GIS from the early 1990s (Roger S. Bivand, 1996, 1997, 1998). Scripts would also play an important role in advancing reproducibility in analysing spatial data (Brunsdon & Comber, 2020). Although neither R nor Python are "little languages" any more, their on-ramps are less forbidding and much better supported by communities and training than previously. Both are also largely open-source, and both support the integration of external open-source geospatial software into scripting environments.

R (R Core Team, 2021) and Python (Van Rossum & Drake, 2009) permit scripts to be written using basic language functionality. This functionality, however, does not extend to the handling of spatial data. The R analytical environment includes the Comprehensive R Archive Network (CRAN), which serves user contributed packages of software with documentation. These have

included packages for spatial data handling and analysis since the early 2000s (Roger S. Bivand, 2020b), including packages providing scripting access to key open-source geospatial software libraries. These libraries, and some of the interface and other code in R packages, are written in languages requiring compilation, which may complicate distribution to users. CRAN provides users with "binary" contributed packages for Windows and MacOS to permit users to install such packages and their dependencies. R contributed packages may be installed and updated in a running R session, but should not be updated if already used in a session. Because CRAN contributed package binaries are available for Windows and MacOS, and contain their required external library components by static linking, the computing environment is closely controlled and almost certainly coherent (because served packages are tested against each other on multiple platforms continuously).

Python packages are provided in a very similar way using the Python Package Index. Unlike R, Python packages should not be installed in a running Python session, but prior to its commencement. Typically, `pip` has been used to manage handling contributed packages, and more recently complete `conda` computing environments are also available. Specified computing environments permit users to carry out chosen tasks without the need to install chains of individual software components, in much the same way as the decision by CRAN to link Windows and MacOS packages accessing external libraries statically. External libraries do evolve, and if a new version changes an interface component for whatever reason, then software such as Python or R packages using that library but linking dynamically would need re-compilation. Further opportunities for customizing and controlling computing environments are offered by containers such as Docker, or by using Binder to create computing environments for remote users.

While computing environments are a strict subset of analytical environments, they do matter a great deal. Community activity on mailing lists and question-and-answer fora often provide example scripts resolving problems. These solutions are however dependent on the versions of software components being used, both the packages themselves, other packages they use, and the external software libraries that they interface. Open source packages for analysing spatial data share many components in their use of open source external libraries, and this leaves most of them vulnerable to changes (Roger S. Bivand, 2014). Such changes do occur frequently, recently in PROJ (Evenden et al., 2022), GEOS and GDAL (Rouault et al., 2022). These changes lead to churn in releases of R and Python packages as small numbers of maintainers struggle to adapt to upstream changes while assuring the stability of downstream packages and scripted workflows.

Examples of upstream changes in PROJ and GDAL feeding through into R and Python packages concern the ways in which coordinate reference systems (CRS) are represented. In the 1990s, the specification of CRS of spatial data was downplayed; for example `.prj` files containing a text version of the CRS were optional. Correct CRS specification impacted visualization in web mapping applications from the mid-2000s, and was always essential for the integration of data across multiple data sources. In such cases of upstream changes, a change in the computing environment may engender changes in the wider analytical environment, where users are obliged to absorb technical details which might have seemed unimportant.

R and contributed packages

```
Sys.setenv(PROJ_NETWORK="ON")
library(sf)

## Linking to GEOS 3.10.2, GDAL 3.4.1, PROJ 8.2.1; sf_use_s2() is TRUE

packageVersion("sf")

## [1] '1.0.5'
```

The **sf** package (Pebesma, 2018, 2020), introduced in late 2016, is a modern replacement for the **sp** package (Pebesma & Bivand, 2005, 2021); see Roger S. Bivand (2020b) for a fuller account. In Roger S. Bivand et al. (2013) and other books presenting spatial data handling workflows, **sp** is used for both vector and raster data, supplemented by **raster** (Hijmans, 2022). The **rgdal** package (Roger S. Bivand et al., 2021) then provided functionalities offered by the external PROJ and GDAL libraries for reading and writing spatial data files; the **rgeos** package (Roger S. Bivand & Rundel, 2021) interfaced GEOS predicates and topological operations. The **sf** package was first written as a replacement for vector data handling in **sp**, **rgdal** and **rgeos**, but also now supports raster data handling by interfacing GDAL for the **stars** package (Pebesma, 2021). Changing from **sp** based workflows to **sf** based workflows is proceeding, but necessarily impacts the broader analytical environment. It has been important to ensure that **sp** based workflows continue to function as far as possible. The **rgdal** and **rgeos** will be retired by or before the end of 2023 to simplify maintenance of the R-spatial package ecosystem.

We will use a dataset from the British Census 2011 for Output Areas in the Borough of Camden in London used in Roger S. Bivand & Wong (2018), and described in detail there. It is stored in a Geopackage format file, which overcomes many of the shortcomings of the legacy ESRI Shapefile format. It is read into an `"sf"` object using the GDAL GPKG driver:

```
camden <- st_read("oa_census.gpkg")

## Reading layer `oa_census' from data source
##   `/home/rsb/papers/sass19/oa_census.gpkg' using driver `GPKG'
## Simple feature collection with 749 features and 5 fields
## Geometry type: POLYGON
## Dimension:     XY
## Bounding box:  xmin: 523954.5 ymin: 180959.8 xmax: 531554.9 ymax: 187603.6
## Projected CRS: OSGB36 / British National Grid
```

This dataset includes only a very few variables chosen from the 2011 Census, including counts of total population, unemployed and potentially economically active. The proportion of the population who were potentially economically active varies a good deal across the output areas, from less than half of the population to almost everyone:

```
summary(with(camden, all_categories_economic_activity/Population))

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.4636  0.7360  0.7870  0.7886  0.8399  0.9912
```

The density of the potentially economically active also varies greatly by output area; there are some parks in Camden which yield low densities. The **units** package (Pebesma et al., 2021) is used to present the output of `st_area()` in hectares rather than square metres:

```
ha <- st_area(camden)
library(units)
```

```
units(ha) <- as_units("ha")
dens <- camden[["all_categories_economic_activity"]]/ha
print(quantile(dens, seq(0, 1, 0.25)), digits=4)

## Units: [1/ha]
##       0%      25%      50%      75%     100%
##    2.327   85.288  125.428  177.413 3364.327
```

The cartogram map shown in the right panel of Figure 3 iterates over the boundaries of the output areas to adjust them to better proportionality with the number of potentially economically active residents (Dougenik et al., 1985; Jeworutzki, 2020).

```
library(cartogram)
sf_cont <- cartogram_cont(camden, "all_categories_economic_activity", 7)
```

The **classInt** package (Roger S. Bivand, 2020a) is used internally by a number of map plotting functions in R packages to calculate class intervals. Here we show its output breaks, calculated using the Fisher (1958) style:

```
(cI <- classInt::classIntervals(camden[["Unemployment"]], n=6, style="fisher"))

## style: fisher
##         [0,2.02774)  [2.02774,3.668135) [3.668135,5.542755) [5.542755,7.745236)
##                 129                 196                 189                 144
## [7.745236,11.19164) [11.19164,18.62348]
##                  80                  11
```

The colour palette is chosen from the sequential palettes provided by the **rcartocolor** package (Nowosad, 2019).

```
pal <- rcartocolor::carto_pal((length(cI$brks)-1L), "TealGrn")
```

The left panel of Figure 3 shows a standard choropleth map of unemployment as a percentage of the number of economically active residents, using the **tmap** package (Tennekes, 2018, 2021); the right panel shows the cartogram. Using the same intervals and colours for the output areas map and the cartogram is made easy by passing the class intervals and colour palette just defined.

```
library(tmap)
a <- tm_shape(camden) + tm_fill("Unemployment", breaks=cI$brks, palette=pal) +
  tm_layout(main.title="Output Areas")
b <- tm_shape(sf_cont) + tm_fill("Unemployment", breaks=cI$brks, palette=pal) +
  tm_layout(main.title="Cartogram", legend.show=FALSE)
tmap_arrange(a, b)
```

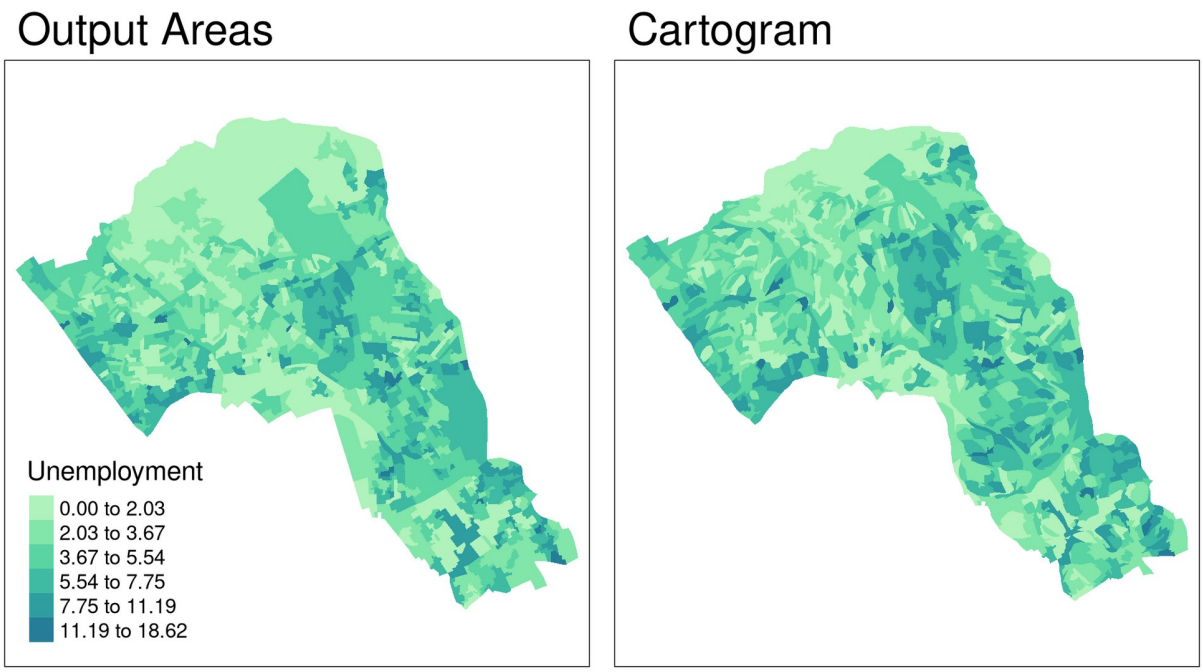

Figure 2: left panel: Percentage unemployment by output area in Camden, 2011 Census; right panel: cartogram of percentage unemployment with output areas adjusted to the size of the economically active population

We can capture an interactive visualization on contextual information from Open Street Map using the **mapview** package (Appelhans et al., 2021), showing the census output area boundaries on the contextual background of the street network in Camden and its surroundings (Figure 4).

```
library(mapview)
if (sf:::CPL_gdal_version() >= "3.1.0") mapviewOptions(fgb = FALSE)
camden_wmap <- mapview(camden, zcol="Unemployment", col.regions=pal, at=cI$brks)
mapshot(camden_wmap, file=file.path(getwd(), "camden_wmap.png"))
```

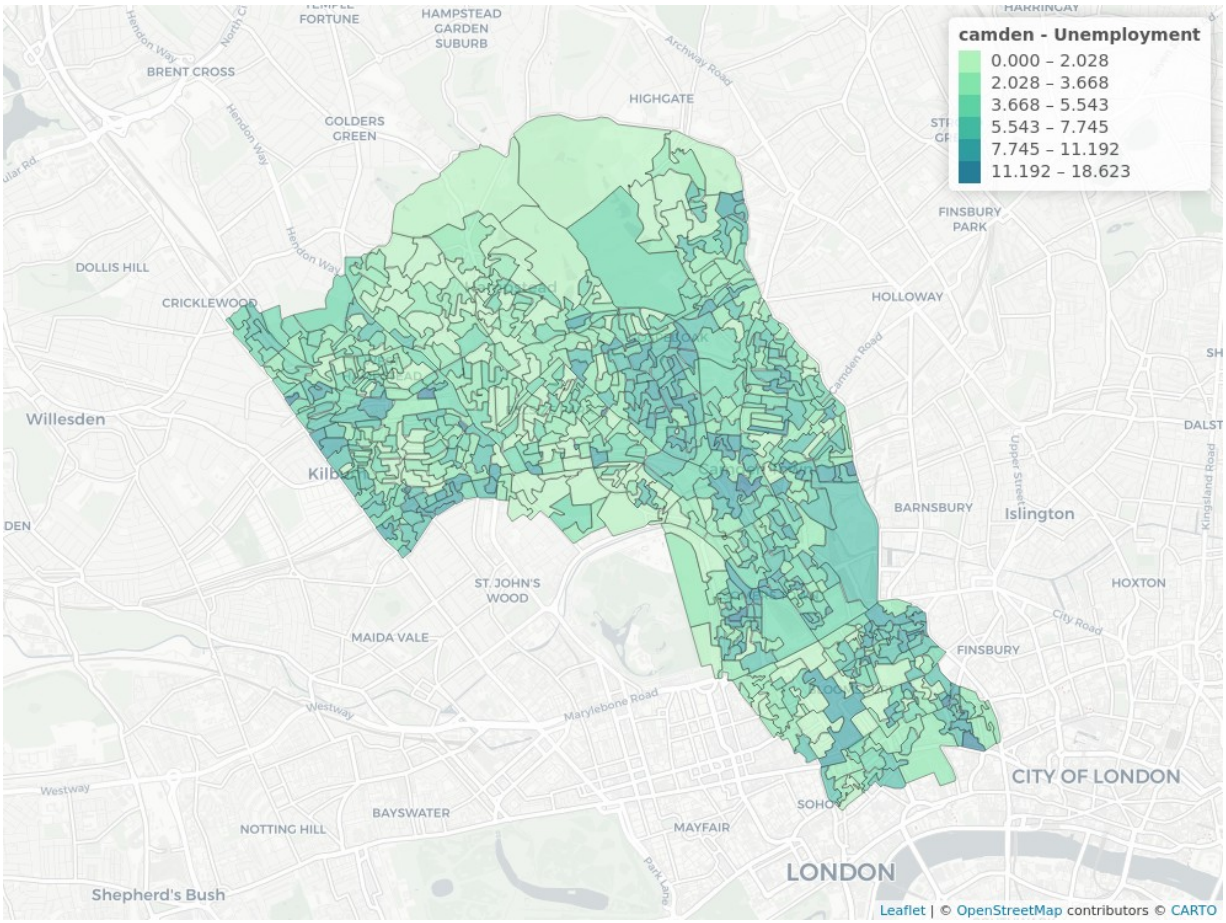

Figure 3. Interactive choropleth map of percentage unemployment shown on an Open Street Map layer

In order to create an interactive web map, the `camden` object has to be transformed from the "OSGB 1936 / British National Grid" projected CRS (EPSG:27700) as read in from file to geographical coordinates in the `WGS84` datum (OGC:CRS84), then projected internally to Web Mercator (EPSG:3857) to match the background. The second step is not hard, but the first step depends on the spatial object having a well-defined CRS, and that descriptions of the steps to get from that CRS to the geographical CRS in `WGS84` are known. They may be multiple descriptions, with differing degrees of accuracy.

Before PROJ 6 and GDAL 3, released in March 2019, projected CRS definitions were typically stored in text files distributed with the software. The definitions, known as Proj4 strings, contained key-value pairs of definition components, and often provided hints about how to transform to `WGS84`. To transform from one non-`WGS84` datum to another non-`WGS84` datum, maps were transformed first from source CRS to `WGS84`, then on to the target CRS, incurring two sets of transformation errors. Transformation means converting from one datum, a model of deviations from a specified ellipsoid, to another datum (Iliffe & Lott, 2008). Errors from ignoring datum transformation, termed *ballpark* accuracy, are typically greater than 100m, or 3 to 4 Landsat 7 satellite image pixels.

PROJ 6 (2019) introduced an SQLite database, also used by GDAL 3 (also 2019), to replace the text files distributed with the system. This modernization changed the approach used for datum transformation from partly ad-hoc to authority-based, and happened at the same time as the introduction of the `WKT2:2019` international standard for referencing coordinates (ISO, 2019). All geospatial software is adapting to these ongoing changes, and for open source software, the changes are driven by decisions made by PROJ and GDAL developers.

Before PROJ 6, Proj4 strings could contain `+datum=` keys taking values such as `OSGB36`. A very few values are valid in PROJ 6 and later, but `OSGB36` is not one of these. If we feed the valid PROJ 5 string through the **sf** `st_crs()` method to create a `"crs"` object from a Proj4 string, degradation occurs when **sf** uses PROJ version 6 or later, with the `+datum=OSGB36` key value pair replaced by `+ellps=airy`; this ellipsoid definition is correct, but the key-value pair is insufficiently clear about which datum definition to choose, so the PROJ library falls back to the ellipsoid:

```
(degraded_proj4 <- st_crs(paste("+proj=tmerc +lat_0=49 +lon_0=-2 +k=0.9996012717",
"+x_0=400000 +y_0=-100000 +datum=OSGB36 +units=m +no_defs"))$proj4string)

## [1] "+proj=tmerc +lat_0=49 +lon_0=-2 +k=0.9996012717 +x_0=400000 +y_0=-100000
+ellps=airy +units=m +no_defs"
```

The fully qualified `WKT2:2019` representation as read from the Geopackage file (CRS name shown as User input) appears more verbose, specifying exactly which European Petroleum Survey Group (EPSG) code is used in each definition:

```
st_crs(camden)
```

The step being taken within **mapview** is to transform from the source CRS to the target `OGC:CRS84` CRS; here we will choose the centroid of the first census output area as an example (for comparison with Python below):

```
sfc <- st_centroid(camden[1,])
```

From PROJ 7, a cloud resource permits transformation grids to be accessed from a content download network (CDN, https://cdn.proj.org/) on-demand as an Amazon Public Dataset and used if accuracy would be improved. The network may be enabled by setting the environment variable `PROJ_NETWORK` to `ON` before loading the **sf** package or any other package needing access to the CDN.

```
Sys.getenv("PROJ_NETWORK")

## [1] "ON"
```

This means that if:

```
sf_proj_network()

## [1] TRUE
```

then `st_transform()` will proceed by first looking up candidate transformation pipelines in the PROJ database, and choose the most accurate, including pipelines using transformation grids. If the required grids are not already available on the system running the command, they will be downloaded and cached before being used; the output coordinates are given with extreme detail to demonstrate below that they agree with the Python output:

```
(sfc_ll_best <- st_transform(sfc, "OGC:CRS84")) |> st_coordinates() |> print(digits=17)

##                      X                  Y
## 1 -0.17053693927522964 51.546909001711455
```

Since `st_transform()` does not report which pipeline was chosen, and if we would like to know the choice, we may use `sf_proj_pipelines()` to list the candidate transformation pipelines. Here we see that the tenth accuracy value (given in metres) is the smallest, followed by the second; the last (eleventh) is known as *ballpark* accuracy as its accuracy cannot be determined from the PROJ database:

```
trans_pipes <- sf_proj_pipelines(st_crs(sfc), "OGC:CRS84")
trans_pipes$accuracy

##  [1] 21  2 21 10 21 18 35  3  5  1 NA
```

Transformation pipelines are divided into steps, where here for the best choice the first step reverse projects from OSGB36 to geographical coordinates, next a horizontal grid shift is applied between that datum and the current WGS84 datum (actually the European Terrestrial Reference System 1989 (ETRS89), EPSG:4258, for this grid), finally converting back from radians to degrees:

```
gsub("\\+step", "\n+step", trans_pipes$definition[10]) |> cat("\n")

## +proj=pipeline
## +step +inv +proj=tmerc +lat_0=49 +lon_0=-2 +k=0.9996012717 +x_0=400000 +y_0=-100000
+ellps=airy
## +step +proj=hgridshift +grids=uk_os_OSTN15_NTv2_OSGBtoETRS.tif
## +step +proj=unitconvert +xy_in=rad +xy_out=deg
```

We can confirm that this pipeline was in fact used by re-running the transformation and specifying the pipeline to be used; the output point is the same as that returned when `st_transfrom()` chose the most accurate available pipeline itself:

```
st_transform(sfc, "OGC:CRS84", pipeline=trans_pipes$definition[10]) |>
st_distance(sfc_ll_best) |> c()

## 0 [m]
```

Had we persisted in using the legacy Proj.4 string representation described above, no transformation step is present and we simply apply inverse projection:

```
gsub("\\+step", "\n+step", trans_pipes$definition[11]) |> cat("\n")

## +proj=pipeline
## +step +inv +proj=tmerc +lat_0=49 +lon_0=-2 +k=0.9996012717 +x_0=400000 +y_0=-100000
+ellps=airy
## +step +proj=unitconvert +xy_in=rad +xy_out=deg
```

In this degenerate case, the output point is about 123m from the most accurate transformation:

```
st_transform(sfc, "OGC:CRS84", pipeline=trans_pipes$definition[11]) |>
st_distance(sfc_ll_best) |> c()

## 122.837 [m]
```

In the absence of transformation grids, the next-best transformation pipeline with 2m accuracy is offered by a Helmert seven-parameter transformation, with parameters retrieved from the PROJ database:

```
gsub("\\+step", "\n+step", trans_pipes$definition[2]) |> cat("\n")

## +proj=pipeline
## +step +inv +proj=tmerc +lat_0=49 +lon_0=-2 +k=0.9996012717 +x_0=400000 +y_0=-100000
+ellps=airy
## +step +proj=push +v_3
## +step +proj=cart +ellps=airy
## +step +proj=helmert +x=446.448 +y=-125.157 +z=542.06 +rx=0.15 +ry=0.247 +rz=0.842
+s=-20.489 +convention=position_vector
## +step +inv +proj=cart +ellps=WGS84
## +step +proj=pop +v_3
## +step +proj=unitconvert +xy_in=rad +xy_out=deg
```

This pipeline would be the one chosen if the transformation grid was not available and automatic downloading from the PROJ content download network was turned off; the point is almost 2m from the most accurate transformation known to the PROJ database:

```
st_transform(sfc, "OGC:CRS84", pipeline=trans_pipes$definition[2]) |>
st_distance(sfc_ll_best) |> c()

## 1.764523 [m]
```

In other settings, the PROJ database offers a starker choice between enabling the CDN and only *ballpark* accuracy if grid transformation is not available, so use of `sf_proj_pipelines()` to check which choices are available is advised.

Handling spatial data, then, requires that the geometries of the spatial entities be recorded and input, be associated with the best available specification of the CRS, be correctly associated with the values of variables of interest, be subsetted and transformed, and usually be visualized in the form of thematic cartography. It is only very recently that it has been desirable to pay sustained attention to coordinate reference systems and their transformation. To put this brief presentation of some simple aspects of the R environment for spatial data handling, we'll now draw on Python equivalents. These environments are by no means just the language platform and contributed packages, but include upstream software libraries, software developers and maintainers and their interactions with users, and the varied ways in which these participants conceive of their needs and the goals of their activities.

Python and contributed packages

The whole Python environment has shifted to Python 3, causing some churn among packages. For geospatial data, the **geopandas** package (Bossche et al., 2021) uses a similar approach to the R **sf** package, representing data as a data frame with a geometry column. Unlike **sf**, **geopandas** does not itself link to PROJ, GDAL or GEOS, relying on the **pyproj** package for PROJ (Snow et al., 2021), the **Fiona** package for GDAL (Gillies, Buffat, et al., 2021), and the **Shapely** package for GEOS (Gillies, Taves, et al., 2021):

```
import geopandas
geopandas.show_versions()

##
## SYSTEM INFO
```

```
## -----------
## python     : 3.10.2 (main, Jan 17 2022, 00:00:00) [GCC 11.2.1 20211203 (Red Hat
11.2.1-7)]
## executable : /usr/bin/python
## machine    : Linux-5.15.16-200.fc35.x86_64-x86_64-with-glibc2.34
##
## GEOS, GDAL, PROJ INFO
## ---------------------
## GEOS       : 3.10.2
## GEOS lib   : /usr/local/lib64/libgeos_c.so
## GDAL       : 3.4.1
## GDAL data dir: None
## PROJ       : 8.2.0
## PROJ data dir: /home/rsb/.local/lib/python3.10/site-packages/pyproj/proj_dir/share/
proj
##
## PYTHON DEPENDENCIES
## -------------------
## geopandas  : 0.10.2
## pandas     : 1.3.4
## fiona      : 2.0dev
## numpy      : 1.21.5
## shapely    : 1.8.0
## rtree      : None
## pyproj     : 3.3.0
## matplotlib : 3.5.1
## mapclassify: 2.4.3
## geopy      : None
## psycopg2   : None
## geoalchemy2: None
## pyarrow    : None
## pygeos     : None
```

Consequently, **geopandas** bundles other packages and their interfaces to external software libraries, modularizing perhaps more effectively than **sf**. The same Geopackage file may be read into a `"GeoDataFrame"` using many of the same underlying parts of the open source geospatial software stack as in the **sf** example above:

```
geodf = geopandas.read_file('oa_census.gpkg')
```

In R, we took the centroid of the first output area, indexing by 1 because R indices are 1-based; Python indices are 0-based, so we get the same output by using a zero index; the centroid is found using GEOS through **shapely**:

```
cent1 = geodf.centroid[0]
cx = cent1.x
cy = cent1.y
```

The CRS of the census output areas data set is identical with the **sf** output above:

```
geodf.crs
```

```
## <Derived Projected CRS: EPSG:27700>
## Name: OSGB36 / British National Grid
## Axis Info [cartesian]:
## - E[east]: Easting (metre)
## - N[north]: Northing (metre)
## Area of Use:
## - name: United Kingdom (UK) - offshore to boundary of UKCS within 49°45'N to 61°N
and 9°W to 2°E; onshore Great Britain (England, Wales and Scotland). Isle of Man
onshore.
```

```
## - bounds: (-9.0, 49.75, 2.01, 61.01)
## Coordinate Operation:
## - name: British National Grid
## - method: Transverse Mercator
## Datum: Ordnance Survey of Great Britain 1936
## - Ellipsoid: Airy 1830
## - Prime Meridian: Greenwich
```

The adoption by **geopandas** of the **pyproj** CRS interface described in a helpful blog (https://jorisvandenbossche.github.io/blog/2020/02/11/geopandas-pyproj-crs/) mirrors the changes made in the R spatial package ecosystem. Indeed, the **pyproj** package was an early mover in this respect, and coding in **rgdal** was in part based on the then development version of **pyproj** (Snow et al., 2021). Anecdotally, the use of Cython to interface the compiled code of PROJ in **pyproj** permitted the prototyping of this functionality during a 7 hour train trip in late October 2019, leading to workable listing of transformations from a source CRS to a target CRS; `TransformerGroup()` is very similar to `sf::sf_proj_pipelines()`. The benefits of open source software also lie in seeing how other communities have approached shared puzzles.

It is then not surprising that **sf** retrieves the same transformation pipelines from PROJ as **pyproj** does. Here we'll retrieve all the feasible pipelines included in a `TransformerGroup` object. We turn on the CDN first to include any relevant transformation grid (here downloaded and cached during the first such run in R above):

```
import pyproj
from pyproj.transformer import TransformerGroup
pyproj.network.set_network_enabled(True)
tg = TransformerGroup(geodf.crs, 'OGC:CRS84', always_xy=True)
itg = TransformerGroup('OGC:CRS84', geodf.crs, always_xy=True)
tg

## <TransformerGroup: best_available=True>
## - transformers: 11
## - unavailable_operations: 0
```

The best available pipeline is returned in the 0-index position; this is the grid-based transformation:

```
import re
print(re.sub("step", "\nstep", tg.transformers[0].definition))

## proj=pipeline
## step inv proj=tmerc lat_0=49 lon_0=-2 k=0.9996012717 x_0=400000 y_0=-100000
ellps=airy
## step proj=hgridshift grids=uk_os_OSTN15_NTv2_OSGBtoETRS.tif
## step proj=unitconvert xy_in=rad xy_out=deg
```

GEOS is here returning the same projected coordinates for the centroid of the first census output area in projected coordinates, and PROJ is returning the same best available transformation pipeline and using it to yield the same output coordinates:

```
ll0 = tg.transformers[0].transform(cx, cy)
ll0

## (-0.17053693927522964, 51.546909001711455)
```

The second pipeline is the same Helmert transformation-based one as above:

```
print(re.sub("step", "\nstep", tg.transformers[2].definition))

## proj=pipeline
## step inv proj=tmerc lat_0=49 lon_0=-2 k=0.9996012717 x_0=400000 y_0=-100000
ellps=airy
## step proj=push v_3
## step proj=cart ellps=airy
## step proj=helmert x=446.448 y=-125.157 z=542.06 rx=0.15 ry=0.247 rz=0.842 s=-20.489
convention=position_vector
## step inv proj=cart ellps=WGS84
## step proj=pop v_3
## step proj=unitconvert xy_in=rad xy_out=deg
```

and yields a similar distance between coordinates. There does not seem to be functionality in **geopandas** or packages it uses to calculate distances on the sphere, so we transform back using the grid-based transformation and measure the distance on the plane; **sf** in R uses the **s2** package interfacing the `s2geometry` library, so the measurement is not identical:

```
ll2 = tg.transformers[2].transform(cx, cy)
from shapely.geometry import Point
point_df = geopandas.GeoDataFrame({'geometry':
[Point(itg.transformers[0].transform(ll2[0], ll2[1]))]}, crs='EPSG:27700')
point_df.distance(cent1)

## 0    1.769709
## dtype: float64
```

Finally, the *ballpark* accuracy transformation pipeline is the same:

```
print(re.sub("step", "\nstep", tg.transformers[10].definition))

## proj=pipeline
## step inv proj=tmerc lat_0=49 lon_0=-2 k=0.9996012717 x_0=400000 y_0=-100000
ellps=airy
## step proj=unitconvert xy_in=rad xy_out=deg
```

with almost the same distance between points:

```
ll10 = tg.transformers[10].transform(cx, cy)
point_df = geopandas.GeoDataFrame({'geometry':
[Point(itg.transformers[0].transform(ll10[0], ll10[1]))]}, crs='EPSG:27700')
point_df.distance(cent1)

## 0    123.137447
## dtype: float64
```

So for important components of spatial data handling, the Python and R environments perform in very similar ways, not least because both build on the strengths of the same underlying open source geospatial software libraries. Density (count per unit area) measures build on the same library, GEOS, to calculate polygon areas, but then need to use other software implementations for creating a numeric sequence and quantiles, R using `seq()` in base-R and `quantile()` in the base-R **stats** package, but in Python two functions in **numpy**:

```
import numpy as np
dens = geodf['all_categories_economic_activity'] / (geodf.area/10000)
arr = np.quantile(dens, np.linspace(0, 1, 5))
np.set_printoptions(precision=3, suppress=True)
print(arr)

## [   2.327   85.288  125.428  177.413 3364.327]
```

The output is the same, but this might not have occurred had the default interpolation methods for the two implementations not given the same outcome for this data set (`quantile()` in R has 9 types, in **numpy** there are 5).

Creating cartograms is a much narrower topic, so it is perhaps not surprising that output of the **cartogram_geopandas** package (Viry, 2015) is rather different from that of the R **cartogram** package (Jeworutzki, 2020) used above, even though both claim to implement the same Dougenik et al. (1985) algorithm.

```
from cartogram_geopandas import make_cartogram
transformed_geodf = make_cartogram(geodf, 'all_categories_economic_activity', 7,
inplace=False)
```

The Python PySAL (Python Spatial Analysis Library) has developed, like the R-spatial package ecosystem, over a number of years (Rey et al., 2015; Rey & Anselin, 2007, 2010). It includes the **mapclassify** package (Rey & Wolf, 2021), which is very similar to the R **classInt** package, and the `FisherJenks()` classifier is like the **classInt** `"fisher"` style shown above:

```
import mapclassify as mc
fj6 = mc.FisherJenks(geodf["Unemployment"], k=6)
fj6

## FisherJenks
##
##    Interval      Count
## ----------------------
## [ 0.00,  2.02] |   129
## ( 2.02,  3.65] |   196
## ( 3.65,  5.53] |   189
## ( 5.53,  7.69] |   144
## ( 7.69, 10.94] |    80
## (10.94, 18.62] |    11
```

The breaks are systematically off by 0.01, so we can insert the values from R to achieve better comparability in plotting; as yet the reasons for the differences have not been explored:

```
cIfj = mc.UserDefined(geodf["Unemployment"], bins=[2.027740,  3.668135,  5.542755,
7.745236, 11.191642, 18.623482])
cIfj

## UserDefined
##
##    Interval      Count
## ----------------------
## [ 0.00,  2.03] |   129
## ( 2.03,  3.67] |   196
## ( 3.67,  5.54] |   189
## ( 5.54,  7.75] |   144
## ( 7.75, 11.19] |    80
## (11.19, 18.62] |    11
```

We can create a figure similar to Figure 3 using **matplotlib** and inserting the `TealGrn` colour palette used above:

```
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
mycmap = ListedColormap(["#b0f2bc", "#82e6aa", "#5bd4a4", "#3fbba3", "#2e9ea1",
"#257d98"])
plt.figure(figsize=(13,7))
```

```
ax1 = plt.subplot(121)
cIfj.plot(geodf, cmap=mycmap, ax=ax1, legend=True, legend_kwds={'loc':'lower left'});
plt.title('Output Areas')
plt.axis('off');
ax2 = plt.subplot(122)
cIfj.plot(transformed_geodf, cmap=mycmap, ax=ax2);
plt.title('Cartogram')
plt.axis('off');
plt.show()
```

Figure 4: left panel: Percentage unemployment by output area in Camden, 2011 Census; right panel: cartogram of percentage unemployment with output areas adjusted to the size of the economically active population

As one might expect, these examples of handling spatial data in two environments show some differences, but also a considerable convergence, especially where both environments build on the same open geospatial software libraries. Consequently, one would not expect to see differences in analytical results because of differences in the reading and cleaning of spatial data.

Perhaps it would be helpful to have added more than citations of the packages, because the authorships of packages in each environment do overlap to a considerable extent. This overlapping generated within communities of developers and users leads to understandings of the needs of packages where they form dependency networks. In addition, there is some contact between developers of spatial packages in Python and R; for example the author of **cartogram_geopandas** is a member of the Github organization to which the R **cartogram** package belongs.

Environments for analysing spatial data

Some spatial analysis in the social sciences has used distance between point-support observations as such, in point pattern analysis and using spatial interpolation in a geostatistical way. Point-support observations, such as house prices or reported crimes, are however often aggregated either by the analyst, or by public agencies prior to publication, for example census data. Election data are only reported for areal aggregates. Much quantitative analysis takes place on aggregated, tabular data without acknowledging the underlying challenge of spatial autocorrelation (Cliff & Ord, 1969; Duncan et al., 1961). In addition, the aggregation units themselves may not match the spatial footprint of the underlying data generation process, leading to the intertwined issues of the ecological fallacy (Robinson, 1950), and the modifiable areal unit problem (MAUP) (Openshaw & Taylor, 1979).

While the ecological fallacy and MAUP remain open problems, spatial autocorrelation can be approached in a number of ways, including testing for its presence in variables and regression residuals, and adding spatial autoregressive terms in regression models. These steps involve the preparation of spatial weights representing the sets of neighbouring observations for a chosen definition of neighbour. A typical definition is that areal units share a boundary point (Queen neighbours as on a chessboard), or a boundary line segment (Rook neighbours). It is possible to read shared boundaries from a map, such as Figure 1, and tabulated in Appendix 2 of the same book (Cliff & Ord, 1973). The provision of software for automating this process is central in the creation of analytical environments for areal data.

Having had access to the digitizer at the LSE in the early 1970s, it was not hard to see that finding lists of neighbours of polygons for analysis could be automated. The choropleth map program needed point coordinates, ordered vectors of point IDs belonging to lines, and ordered vectors of line IDs forming closed polygons. It was then uncomplicated to extract pairs of polygon neighbours, where the same line IDs occurred in polygon definitions (Roger S. Bivand, 1975). Sadly, the Fortran subroutines I wrote have not survived many moves, but the underlying specification of spatial neighbour objects is effectively unchanged. Most of those implementing spatial neighbour and spatial weights objects from the 1970s to today have either used lists of neighbour IDs, or dense or sparse matrices.

An example may be the `poly2nb()` function in the R **spdep** package, returning a list of neighbours object specified as neighbours if polygons share at least one boundary point (Queen criterion):

```
library(spdep)
(nb_q <- poly2nb(camden, queen=TRUE))

## Neighbour list object:
## Number of regions: 749
## Number of nonzero links: 4342
## Percentage nonzero weights: 0.7739737
## Average number of links: 5.797063

table(card(nb_q))

##
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  17
##   1  19  61 104 168 157 112  66  33  14   6   2   4   1   1
```

The print method for the neighbour object reports the number of regions and of non-zero links; these are counted treating the graph as possibly directed, so symmetric links are counted twice. Some implementations assume that the neighbours are symmetric, so count each link once only.

```
lw <- nb2listw(nb_q, style="W")
```

Once we have the neighbour sets, we can attach weights to the links, for example row-standardising the weights for each observation so that they sum to unity for each observation, using `style="W"`. W can compare the sizes of the objects: the spatial weights list `lw`, the same object converted into a dense matrix, and converted into a sparse matrix (from **spdep** version 1.2-1, modelling functionality is to be found in **spatialreg**), here given in Kb:

```
sz1 <- object.size(lw)
sz2 <- object.size(listw2mat(lw))
requireNamespace("spatialreg", quietly=TRUE)
sz3 <- object.size(as(lw, "CsparseMatrix"))
sz <- c("listw"=sz1, "dense"=sz2, "sparse"=sz3)
round(sz/1024, 1)

##  listw  dense sparse
##  246.1 4430.1  149.0
```

In the Python environment, the **libpysal** package (Rey, Wolf, Gaboardi, et al., 2021; Rey, Anselin, et al., 2021) can be used to create a Queen-criterion neighbour object from the polygons in the **geopandas** object; the tabulation of counts of observations with the same numbers of neighbours is the same as for `poly2nb()`:

```
from libpysal import weights
w_cont = weights.Queen.from_dataframe(geodf)
w_cont.histogram

## [(1, 1), (2, 19), (3, 61), (4, 104), (5, 168), (6, 157), (7, 112), (8, 66), (9, 33),
(10, 14), (11, 6), (12, 2), (13, 4), (14, 1), (15, 0), (16, 0), (17, 1)]
```

From there, we can test for spatial autocorrelation in the unemployment variable using global Moran's I without further difficulty using the **spdep** package (Roger S. Bivand & Wong, 2018):

```
moran.test(camden[["Unemployment"]], lw)

##
##  Moran I test under randomisation
##
## data:  camden[["Unemployment"]]
## weights: lw
##
## Moran I statistic standard deviate = 12.254, p-value < 2.2e-16
## alternative hypothesis: greater
## sample estimates:
## Moran I statistic       Expectation          Variance
##       0.268652075      -0.001336898       0.000485435
```

The test output is returned as an `"htest"` object with a print method in the base-R **stats** package, here with the alternative hypothesis asserting positive spatial autocorrelation, and the variance computed under randomisation. A Python alternative can be found in the **esda** package (Rey, Wolf, Arribas-Bel, et al., 2021). We set the style - transformation - of the neighbours to weights as `"R"` for row-standardised, and print out two of the returned elements,

the value of I itself, and the standard deviate under randomisation, both of which agree with those from **spdep**:

```
import esda
w_cont.transform = "R"
mi = esda.Moran(geodf["Unemployment"], w_cont)
np.set_printoptions(precision=9, suppress=True)
print(np.array((mi.I, mi.z_rand)))

## [ 0.268652075 12.254073315]
```

Similarly, calculating the local Moran's I_i values only involves choosing the appropriate function, in R from **spdep**. Global Moran's I is the sum of the local I_i (in the first column of the returned object) divided by the sum of the spatial weights returned by utility function `Szero()`:

```
I_i <- localmoran(camden[["Unemployment"]], lw)
print(sum(I_i[,1])/Szero(lw), digits=9)

## [1] 0.268652075
```

Reproducing the same output in **esda** leads to a small puzzle, accounted for in Roger S. Bivand & Wong (2018) (pp. 724, 738); in the original definition of local Moran's I_i, the variance of the variable of interest was calculated dividing the sum of squared deviations from the mean by n not $(n-1)$:

```
lisa = esda.Moran_Local(geodf["Unemployment"], w_cont)
np.sum(lisa.Is)/w_cont.s0

## 0.2682933940977779
```

The **spdep** function includes an argument `mlvar=` which is `TRUE` by default to demonstrate the reason for the difference:

```
I_ia <- localmoran(camden[["Unemployment"]], lw, mlvar=FALSE)
print(sum(I_ia[,1])/Szero(lw), digits=16)

## [1] 0.2682933940977779
```

Figure 5 presents hotspot cluster core maps of local Moran's I_i calculated under randomisation on a choropleth map and a cartogram - even though the output areas are designed to be similar in counts, the visual impact of the two displays differs somewhat. The figure does not follow the proposal advanced in Anselin (2019), to firstly adjust the probability values by false discovery rate and secondly tighten the cutoff for *interesting* observations to 0.005, as this would have left no *interesting* observations for the variable in question.

While it might be tempting to see the `"Low-Low"` local spatial autocorrelation measure in one southern output area (here an oddly-shaped OA with 10% unemployment among about 300 economically active where only 30 inhabitants were not economically active), the OA contains Russell Square, the Senate House of the University of London, and most of the British Museum, so any interpretation might well be misleading. On the other hand, the `"High-High"` local autocorrelation seen around Rowley Way and Ainsworth Way on the edge of the borough in the west reflects neighbouring OAs with 6-9% unemployment. The cartogram helpfully adjusts the visual impact of the OA after its area has been adjusted by the number of economically active:

```
HS <- attr(I_ia, "quadr")$mean
is.na(HS) <- p.adjust(I_ia[, "Pr(z != E(Ii))"], "none") >= 0.05
camden$HS <- HS
sf_cont$HS <- HS
a <- tm_shape(camden) + tm_fill("HS", palette=pal, colorNA="grey95", textNA="Not
\"interesting\"") + tm_layout(main.title="Output Areas")
b <- tm_shape(sf_cont) + tm_fill("HS", palette=pal, colorNA="grey95", textNA="Not
\"interesting\"") + tm_layout(main.title="Cartogram", legend.show=FALSE)
tmap_arrange(a, b)
```

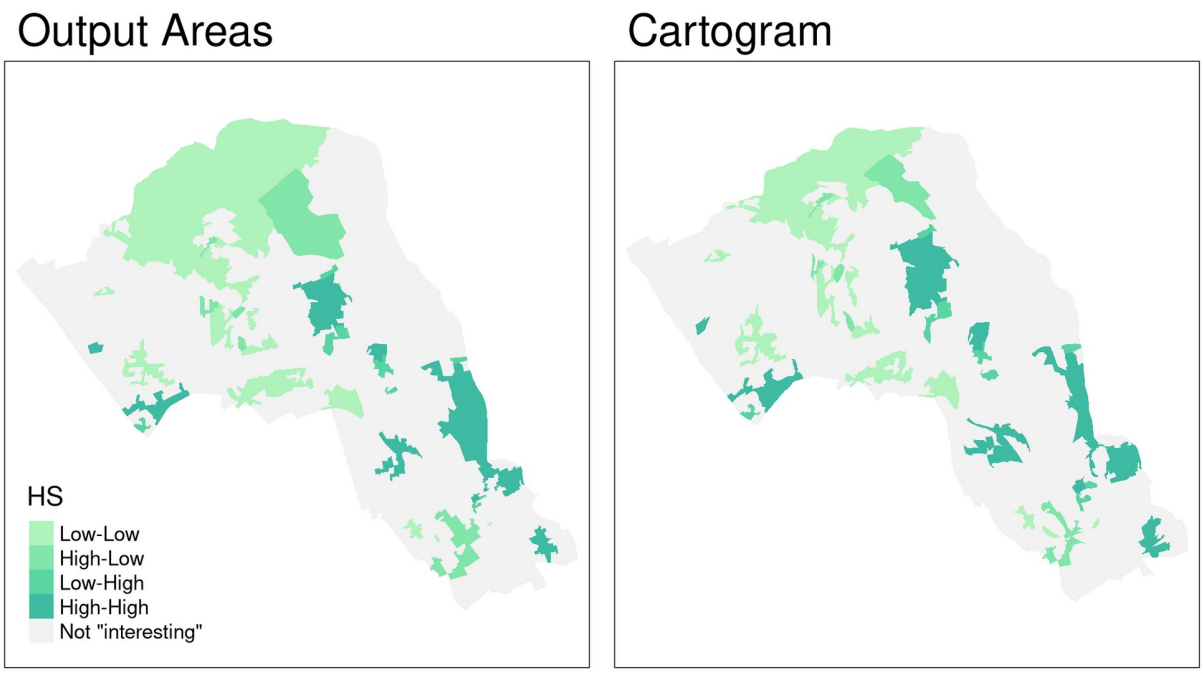

Figure 5: Local Moran's I hotspot cluster cores: left panel: choropleth map; right panel: cartogram

Here we have used R and Python to exemplify how flexible a command line interface may be. Both of these offer notebooks, permitting reports to be written embedding code, the output of executing the code, and text relating these to the questions being handled in a single document. This approach has been followed here as an Rmarkdown document rendered into Word format (source materials available from https://github.com/rsbivand/sass22). We could also have referred to ArcGIS (Scott & Janikas, 2010), the original SpaceStat system (Anselin, 1992), or indeed GeoDa (Anselin et al., 2006, 2021) and other analytical environments, but have chosen to present R and Python as examples of analytical environments within the broader open source geospatial communities. In conclusion, it is worth stressing that these analytical environments are not (just) computer scripts, packages, libraries or languages; they grow and thrive through self-constituted and self-renewing communities of users and developers. In particular, if users intervene actively to express needs for additional functionalities, it is likely that progress will be achieved, but being active is crucial, as there is no "sales department" searching for unexploited demand.

Prospects

As the ease of use of notebook approaches to authoring research reports increases, and as commitments to reproducible research strengthen (Brunsdon & Comber, 2020), the scripting language used may slip into the background. Many tools are already available, and applied researchers will be most concerned to use these tools in addressing relevant research questions.

It will remain relevant to compare implementations of research methods in software tools. With co-authors, I have been attempting to compare the implementations of tools in different analytical environments for analysing spatial data: fitting spatial regression models (R. S. Bivand et al., 2021; Roger S. Bivand & Piras, 2015), fitting spatial multilevel models (Roger S. Bivand et al., 2017), and testing for spatial autocorrelation (Roger S. Bivand & Wong, 2018). When we know more about reasons for differences in tools apparently implementing the same method, users will be able to be confident that at least the calculation of results is carried out in a documented way.

Without doubt there is more to do, and it remains important to maintain and enhance collaboration between those implementing similar functionalities in the various environments, as we undoubtedly have much to share and to learn from each other. In this spirit, building communities of developers and users emerges as being as important as maintaining software and carrying out research into methods in geoinformatics, spatial statistics, or visualization. The R community has been fortunate in seeing advances in this respect, with a recent book undergoing active revision as an example (Lovelace et al., 2019). The PySAL community is also working on a work-in-progress book (https://geographicdata.science/book/intro.html), as are the R-spatial community (https://www.r-spatial.org/book). All these three reviews are also available online, and all use the notebook paradigm embedding reproducible code examples in the text.

References

Anselin, L. (1992). *SpaceStat, a software program for analysis of spatial data*. National Center for Geographic Information and Analysis (NCGIA), University of California.

Anselin, L. (2019). A local indicator of multivariate spatial association: Extending Geary's c. *Geographical Analysis*, *51*(2), 133–150. https://doi.org/10.1111/gean.12164

Anselin, L., Li, X., & Koschinsky, J. (2021). GeoDa, from the desktop to an ecosystem for exploring spatial data. *Geographical Analysis*. https://doi.org/10.1111/gean.12311

Anselin, L., Syabri, I., & Kho, Y. (2006). GeoDa: An introduction to spatial data analysis. *Geographical Analysis*, *38*, 5–22.

Appelhans, T., Detsch, F., Reudenbach, C., & Woellauer, S. (2021). *Mapview: Interactive viewing of spatial data in R*. https://CRAN.R-project.org/package=mapview

Ballas, D., Dorling, D., & Hennig, B. (2014). *The social atlas of Europe*. Policy Press.

Ballas, D., Dorling, D., & Hennig, B. (2017). *The human atlas of Europe*. Policy Press.

Bivand, Roger S. (1975). *The economic geography of regional differentiation: Studies in Sogn og Fjordane, Norway* [PhD thesis, The London School of Economics; Political Science (LSE)]. http://etheses.lse.ac.uk/id/eprint/3210

Bivand, Roger S. (1996). Scripting and toolbox approaches to spatial analysis in a GIS context. In M. Fischer, Henk Scholten, & D. Unwin (Eds.), *Spatial analysis perspectives on GIS* (pp. 39–52). Taylor; Francis.

Bivand, Roger S. (1997). Scripting and tool integration in spatial analysis: Prototyping local indicators and distance statistics. In Z. Kemp (Ed.), *Innovations in GIS* (pp. 127–138). Taylor; Francis.

Bivand, Roger S. (1998). Software and software design issues in the exploration of local dependence. *The Statistician*, *47*, 499–508.

Bivand, Roger S. (2009). Applying measures of spatial autocorrelation: Computation and simulation. *Geographical Analysis*, *41*, 375–384.

Bivand, Roger S. (2014). Geocomputation and open source software: Components and software stacks. In R. J. Abrahart & L. M. See (Eds.), *Geocomputation* (pp. 329–355). CRC Press. http://hdl.handle.net/11250/163358

Bivand, Roger S. (2020a). *classInt: Choose univariate class intervals*. https://CRAN.R-project.org/package=classInt

Bivand, Roger S. (2020b). Progress in the R ecosystem for representing and handling spatial data. *Journal of Geographical Systems*. https://doi.org/10.1007/s10109-020-00336-0

Bivand, Roger S., Keitt, T., & Rowlingson, B. (2021). *Rgdal: Bindings for the 'geospatial' data abstraction library*. https://cran.r-project.org/package=rgdal

Bivand, R. S., Millo, G., & Piras, G. (2021). A review of software for spatial econometrics in R. *Mathematics*, *9*(11). https://doi.org/10.3390/math9111276

Bivand, Roger S., Pebesma, E., & Gomez-Rubio, V. (2013). *Applied spatial data analysis with R, second edition*. Springer, NY. https://asdar-book.org/

Bivand, Roger S., & Piras, G. (2015). Comparing implementations of estimation methods for spatial econometrics. *Journal of Statistical Software*, *63*(1), 1–36. https://doi.org/10.18637/jss.v063.i18

Bivand, Roger S., & Rundel, C. (2021). *Rgeos: Interface to geometry engine - open source ('GEOS')*. https://cran.r-project.org/package=rgeos

Bivand, Roger S., Sha, Z., Osland, L., & Thorsen, I. S. (2017). A comparison of estimation methods for multilevel models of spatially structured data. *Spatial Statistics*. https://doi.org/10.1016/j.spasta.2017.01.002

Bivand, Roger S., & Wong, D. W. S. (2018). Comparing implementations of global and local indicators of spatial association. *TEST*, *27*(3), 716–748. https://doi.org/10.1007/s11749-018-0599-x

Bossche, J. van den, Jordahl, K., & Fleischmann, M. (2021). *GeoPandas: Python tools for geographic data*. GeoPandas. https://github.com/geopandas/geopandas

Brunsdon, C., & Comber, A. (2020). Opening practice: Supporting reproducibility and critical spatial data science. *Journal of Geographical Systems*. https://doi.org/10.1007/s10109-020-00334-2

Cheshire, J., & Uberti, O. (2014). *London: The information capital*. Particular Books.

Cliff, A. D., & Ord, J. K. (1969). The problem of spatial autocorrelation. In A. J. Scott (Ed.), *London papers in regional science 1, studies in regional science* (pp. 25–55). Pion.

Cliff, A. D., & Ord, J. K. (1973). *Spatial autocorrelation*. Pion.

Dorling, D., & Ballas, D. (2011). Innovative ways of mapping data about places. In J. Mason & A. Dale (Eds.), *Understanding social research: Thinking creatively about method* (pp. 150–164). SAGE Publications Ltd. https://doi.org/10.4135/9781446287972.n10

Dougenik, J. A., Chrisman, N. R., & Niemeyer, D. R. (1985). An algorithm to construct continuous area cartograms. *The Professional Geographer*, *37*, 75–81.

Duncan, O. D., Cuzzort, R. P., & Duncan, B. (1961). *Statistical geography: Problems in analyzing areal data*. Free Press.

ESRI. (1998). *ESRI shapefile technical description*. Environmental Systems Research Institute, Inc., Redlands. https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

Evenden, G. I., Rouault, E., Warmerdam, F., Evers, K., Knudsen, T., & Butler, H. (2022). *PROJ*. https://doi.org/10.5281/zenodo.5884395

Fisher, W. D. (1958). On grouping for maximum homogeneity. *Journal of the American Statistical Association*, *53*(284), 789–798. https://doi.org/10.1080/01621459.1958.10501479

Gastner, M. T., & Newman, M. E. J. (2004). Diffusion-based method for producing density-equalizing maps. *Proceedings of the National Academy of Sciences*, *101*(20), 7499–7504. https://doi.org/10.1073/pnas.0400280101

Gillies, S., Buffat, R., & Arnott, J. (2021). *Fiona reads and writes geographic data files*. Toblerity. https://github.com/Toblerity/Fiona

Gillies, S., Taves, M., & Arnott, J. (2021). *Manipulation and analysis of geometric objects in the cartesian plane*. Toblerity. https://github.com/Toblerity/Shapely

Hijmans, R. J. (2022). *Raster: Geographic data analysis and modeling*. https://CRAN.R-project.org/package=raster

Holt-Jensen, A. (2019). Transformations in the discipline of geography experienced over 60 years by a norwegian geographer. *Norsk Geografisk Tidsskrift - Norwegian Journal of Geography*, *73*(4), 229–244. https://doi.org/10.1080/00291951.2019.1617346

Iliffe, J., & Lott, R. (2008). *Datums and map projections: For remote sensing, GIS and surveying*. CRC.

ISO. (2019). *ISO 19111:2019 geographic information – referencing by coordinates*. https://www.iso.org/standard/74039.html

Jeremíasson, K. (1976). Basic program for point-density measurements using a Wang 2200C minicomputer with digitizer. *Computers & Geosciences*, *2*(4), 507–508. https://doi.org/10.1016/0098-3004(76)90042-X

Jeworutzki, S. (2020). *Cartogram: Create cartograms with R*. https://CRAN.R-project.org/package=cartogram

Johnston, R. (2019). On (auto)biography and the history of geography. *Norsk Geografisk Tidsskrift - Norwegian Journal of Geography*, *73*(4), 245–250. https://doi.org/10.1080/00291951.2019.1696399

Lovelace, R., Nowosad, J., & Muenchow, J. (2019). *Geocomputation with R*. Chapman and Hall/CRC. https://geocompr.robinlovelace.net/

Marx, R. W. (1986). The TIGER system: Automating the geographic structure of the United States census. *Government Publications Review*, *13*(2), 181–201. https://doi.org/10.1016/0277-9390(86)90003-8

Meeteren, M. van. (2019). The pedagogy of autobiography in the history of geographic thought. *Norsk Geografisk Tidsskrift - Norwegian Journal of Geography*, *73*(4), 250–255. https://doi.org/10.1080/00291951.2019.1696397

Nowosad, J. (2019). *'CARTOColors' palettes*. https://CRAN.R-project.org/package=rcartocolor

Openshaw, S., & Taylor, P. (1979). A million or so correlation coefficients: Three experiments on the modifiable areal unit problem. In N. Wrigley (Ed.), *Statistical applications in the spatial sciences* (pp. 127–144). Pion.

Pebesma, E. (2018). Simple Features for R: Standardized Support for Spatial Vector Data. *The R Journal*, *10*(1), 439–446. https://doi.org/10.32614/RJ-2018-009

Pebesma, E. (2020). *Sf: Simple features for R*. https://cran.r-project.org/package=sf

Pebesma, E. (2021). *Stars: Spatiotemporal arrays, raster and vector data cubes*. https://CRAN.R-project.org/package=stars

Pebesma, E., & Bivand, R. S. (2005). Classes and methods for spatial data in R. *R News*, *5*(2), 9–13.

Pebesma, E., & Bivand, R. S. (2021). *Sp: Classes and methods for spatial data*. https://cran.r-project.org/package=sp

Pebesma, E., Mailund, T., & Kalinowski, T. (2021). *Units: Measurement units for r vectors*. https://CRAN.R-project.org/package=units

R Core Team. (2021). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.R-project.org/

Renfro, C. G. (2009). Econometric software. In D. A. Belsley & E. Kontoghiorghes (Eds.), *Handbook of computational econometrics* (pp. 1–53). John Wiley & Sons.

Rey, S., & Anselin, L. (2007). PySAL: A python library of spatial analytical methods. *The Review of Regional Studies*, *37*(1), 5–27. http://journal.srsa.org/ojs/index.php/RRS/article/view/134

Rey, S., & Anselin, L. (2010). PySAL: A python library of spatial analytical methods. In M. M. Fischer & A. Getis (Eds.), *Handbook of applied spatial analysis: Software tools, methods and applications* (pp. 175–193). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-03647-7_11

Rey, S., Anselin, L., Amaral, P., Arribas-Bel, D., Cortes, R. X., Gaboardi, J. D., Kang, W., Knaap, E., Li, Z., Lumnitz, S., Oshan, T. M., Shao, H., & Wolf, L. J. (2021). The PySAL ecosystem: Philosophy and implementation. *Geographical Analysis*. https://doi.org/10.1111/gean.12276

Rey, S., Anselin, L., Li, X., Pahle, R., Laura, J., Li, W., & Koschinsky, J. (2015). Open geospatial analytics with PySAL. *ISPRS International Journal of Geo-Information*, *4*(2), 815–836. https://doi.org/10.3390/ijgi4020815

Rey, S., & Wolf, L. J. (2021). *Classification schemes for choropleth mapping*. PySAL. https://github.com/pysal/mapclassify

Rey, S., Wolf, L. J., Arribas-Bel, D., & Kang, W. (2021). *Statistics and classes for exploratory spatial data analysis*. PySAL. https://github.com/pysal/esda

Rey, S., Wolf, L. J., Gaboardi, J., Oshan, T., & Arribas-Bel, D. (2021). *Core components of python spatial analysis library*. PySAL. https://github.com/pysal/libpysal

Robinson, W. S. (1950). Ecological correlations and the behavior of individuals. *American Sociological Review*, *15*, 351–357.

Rouault, E., Warmerdam, F., Schwehr, K., Kiselev, A., Butler, H., & Łoskot, M. (2022). *GDAL*. https://doi.org/10.5281/zenodo.5884352

Schuurman, N. (2000). Trouble in the heartland: GIS and its critics in the 1990s. *Progress in Human Geography*, *24*(4), 569–590. https://doi.org/10.1191/030913200100189111

Scott, L. M., & Janikas, M. V. (2010). Spatial statistics in ArcGIS. In M. M. Fischer & A. Getis (Eds.), *Handbook of applied spatial analysis: Software tools, methods and applications* (pp. 27–41). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-03647-7_2

Shepherd, J., Westaway, J., & Lee, T. (1974). *A social atlas of London*. Clarendon Press.

Snow, A. D., Whitaker, J., Cochran, M., & Bossche, J. van den. (2021). *Python interface to PROJ*. pyproj4. https://github.com/pyproj4/pyproj

Tennekes, M. (2018). tmap: Thematic maps in R. *Journal of Statistical Software*, *84*(6), 1–39. https://www.jstatsoft.org/v084/i06

Tennekes, M. (2021). *Tmap: Thematic maps*. https://CRAN.R-project.org/package=tmap

Thatcher, J., Bergmann, L., Ricker, B., Reuben Rose-Redwood, (Landscapes. of I. R. C., O'Sullivan, D., Barnes, T. J., Barnesmoore, L. R., Imaoka, L. B., Burns, R., Cinnamon, J., Dalton, C. M., Davis, C., Dunn, S., Harvey, F., Jung, J.-K., Kersten, E., Knigge, L., Lally, N., Lin, W., … Young, J. C. (2016). Revisiting critical GIS. *Environment and Planning A: Economy and Space*, *48*(5), 815–824. https://doi.org/10.1177/0308518X15622208

Turner, H. (2020). *Moving forwards greater equity and inclusion in the R community*. https://youtu.be/BbpkKzz71EY

Van Rossum, G., & Drake, F. L. (2009). *Python 3 reference manual*. CreateSpace.

Viry, M. (2015). *Easy construction of continuous cartogram with GeoPandas / GeoDataFrame*. https://github.com/mthh/cartogram_geopandas