

Chapter 17

Looking at John Snow's Cholera Map from the Twenty First Century: A Practical Primer on Reproducibility and Open Science

Daniel Arribas-Bel, Thomas de Graaff, and Sergio J. Rey

17.1 Introduction

In the fall of 2015 Ann Case and Economics Nobel Prize winner Agnus Deaton published a very influential paper in the Proceedings of the National Academy of Sciences (Case and Deaton 2015) concerning the increasing and alarmingly high mortality rates of white Americans aged 45–54. As possible reasons for this phenomenon, they suggested the devastating effects of suicide, alcohol and drug abuse. This article caused quite a great deal of upheaval, and political analysts and columnists even linked this with the electoral unrest amongst the white middle class. However, a comment of an anonymous blogger caused Andrew Gelman to rethink and recalculate the results of Case and Deaton. Namely, what if a shift *within* the age cohort of 45–54 would have happened now with more people being closer to 54 than to 45? Indeed, it turns out that, when correcting for age shifts within cohorts, the results of Case and Deaton are severely less pronounced (although the mortality rates of the white middle aged in the US still stand out compared to other countries).

The example above signifies that, even for Nobel Laureates, there is always a need to be able to reproduce and rethink scientific analyses, especially when the results are this influential. Mistakes can be made, and anecdotes like the above

D. Arribas-Bel (✉)

Department of Geography & Planning, University of Liverpool, Liverpool, UK
e-mail: D.Arribas-Bel@liverpool.ac.uk

T. de Graaff

Department of Spatial Economics, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
e-mail: t.de.graaff@vu.nl

S.J. Rey

School of Geographical Sciences and Urban Planning, Arizona State University,
Tempe, AZ, USA
e-mail: srey@asu.edu

are abundant across all sciences. The scientific process is traditionally designed to correct itself, although this adjustment can be quite sluggish. To facilitate this self-correcting process and to minimize the number of errors within the data preparation, data analysis and results presentation phase, we argue that a proper workflow is needed: namely, one that facilitates reproducibility and Open Science.

In general, the need for more emphasis on research reproducibility and Open Science is increasingly recognised by universities, government institutions and even the public at large. Strangely, however, virtually no training is provided on workflow design and choice of appropriate tools. Students and researchers receive no guidance as to why or how they should adopt habits that favor Open Science principles in their research activity.¹ This applies as well to regional science where, given the emphasis on spatial data, maps and quantitative approaches, the need for a reproducible workflow is probably even more challenging than in most other social sciences. This chapter, therefore, focuses on the concept of workflows, reproducibility and Open Science, and how to apply them in a very practical sense. Moreover, it illustrates these concepts by providing a completely reproducible environment and hands-on example.

The next section deals with the concept of workflow, reproducibility and Open Science, introduces some specific workflows and tackles the question of why these approaches are relevant. In the third section, we give an example of a completely open and reproducible analysis of John Snow's famous cholera map from the nineteenth century. Although a proper workflow does not revolve around one single tool, but instead consists of a *coherent* set of tools and methodologies, we have chosen to use for this purpose the programming languages R and Python in combination with the Jupyter Notebook environment, because of its relative ease of use, accessibility and flexibility. The chapter concludes with a discussion of the advantages and the (perceived) disadvantages of our approach.

17.2 Workflow, Reproducibility and Open Science in Regional Science

The Business Dictionary (BusinessDictionary 2016) states that a workflow is a

progression of steps (tasks, events, interactions) that comprise a work process, involve two or more persons, and create or add value to the organization's activities.

So a workflow in science is a set of steps (such as data gathering, data manipulation, analyses, presenting results), usually taken by multiple researchers, which leads to an outcome (the research findings). Reproducibility requires that the materials used to make a finding are available and that they are sufficient for

¹See for notable exceptions Healy (2011) in the social sciences, and Rey (2014) and Arribas-Bel and de Graaff (2015) in regional science.

an independent researcher (including the original researcher herself) to recreate the finding. Open Science requires that all researchers have free and easy access to all materials used to make such a finding. Unfortunately, making your research open and reproducible often requires additional effort, and one may wonder whether it is worth it. Indeed, adopting a workflow directed at reproducibility and openness can often be costly in terms of time. However, there are significant gains to be made.

First, and the most obvious of all, the research becomes reproducible. This brings great benefits to the scientific community at large. Sharing code for estimations, figures and data management leads to a faster dispersion of knowledge. Secondly, it leads to larger transparency, and thus a higher probability of early error detection. Thirdly, research becomes more modular and portable, so that it is easier to cooperate with colleagues at a distance and to continue with parts of the research where others have left it. Fourthly, one of the most salient advantages of a reproducible workflow is that, in the long term, it makes the scientist more efficient. However, this will show up at the end of the research cycle, when somebody—an editor, a supervisor, a referee, a colleague, your own future self—may ask to redo (parts of the) analysis with slight modifications. In this context, having an automated process that prepares your data, analyses them and presents the final results is of great help. An additional benefit of a reproducible workflow is self-sanity. The effort put to explain to others what steps were taken and how they were approached provides an additional degree of confidence over the traditional case-scenario where documentation is scarce and unclear. Finally, reproducibility and especially openness increases the visibility of the research. Most notably, when code for a complex estimation is available alongside a paper, others will not only be more convinced of the results, but they also will be more likely to use it and give it proper credit.

Often, complete reproducibility in regional science is hard to achieve. Proprietary data, qualitative methods such as interviews and case studies and sampling issues in surveys often prohibit others from perfectly mimicking a study's results. However, by choosing appropriate tools, one can strive to work as reproducibly as possible. Making available coding books for surveys and interviews, protocols for case studies and data management code for proprietary data often significantly helps others to understand how the results have been obtained.

Recent years have seen a remarkable increase in tools and attention to reproducibility and openness. Unfortunately, most of these tools come from the realm of computer science and have not yet permeated into other domains, such as regional science. In general, there is not a particular set of tools that we advocate. However, there are some *types* of tools that in general are unavoidable when striving for an open and reproducible workflow, including:

- Data analysis and programming applications. For quantitative data analyses, one needs tools for data management and statistical analysis, such as the two most popular data science tools at the moment, R and Python.
- Typesetting applications. These are used to convey the text and results, whether on paper (typically using the pdf format) or on screen (using the html

language). Typically, LaTeX is often used for scientific purposes, especially because it is scriptable and produces high quality results. Nowadays, however, Markdown seems to be growing in popularity, mostly because of its very accessible and easy to learn syntax.

- Reference managers. These typically are combined with typesetting applications and form a database of references and a system to handle citations and reference lists. BibTeX, Mendeley, and Endnote are popular applications.
- Version control systems. These enable the archiving of older file versions, while only one copy is ever in use (this avoids the usual awkward naming conventions for files, such as `FinalDraftVersion3.3.doc.final.really.docx`). In combination with central repositories, these version control systems act as well as backup applications. Dropbox is an example of a version control system, just as is the popular open source version control system Git.
- Literate programming environments. These are typically applications able to *weave* code, text and output together. At the moment, there are not many *general* literature programming environments. The most popular are probably the `knitr` package for R² and the `Jupyter` notebook for a multi-language environment. Moreover, these environments are able to write output to different formats (usually, html, Markdown, LaTeX/pdf, and the Open Office .odt format).

In general, tools for reproducible research need to be preferably open source and particularly *scriptable*. The lack of the latter makes it very difficult for other applications to communicate and “cooperate” with the tools used.

17.3 John Snow’s Cholera Map

To demonstrate some of the ideas discussed above, we use a classic dataset in the history of spatial analysis: the cholera map by Dr. John Snow. His story is well known and better described elsewhere (Hempel 2006). Thanks to his mapping exercise of the location of cholera deaths in nineteenth century London, he was able to prove that the disease is in fact transmitted through contaminated water (associated to a specific pump), as opposed to the conventional thinking of the day, which stated that transmission occurred through the air. In this section, we will support Snow’s view with the help of Exploratory Spatial Data Analysis (ESDA) tools. In the process, we will show how a reproducible and open workflow can in fact be applied by including the code required to produce the results presented.³ In fact, the entire content, as well as its revision history, have been tracked using the Git version control software and can be downloaded from

²See for further information how to use R to make your research as reproducible as possible Gandrud (2013) and Stodden et al. (2014).

³Part of this section is based upon Lab 6 of Arribas-Bel (2016), available at <http://darribas.org/gds15>.

https://bitbucket.org/darribas/reproducible_john_snow. Equally, the code required to carry out the analysis is closely integrated in the paper and will be shown inline. A reproducible notebook version of this document, available from the online resource, allows the reader to not only see the code but to interactively execute it without decoupling it from the rest of the content in this chapter.

17.3.1 *Point Pattern Exploration*

This analysis will be performed using a combination of both the Python and R programming languages. In addition to both being free and open-source, they have become essential components of the data scientist's toolkit and are also enjoying growing adoption within academia and scientific environments. Thanks to the Jupyter Notebook (Perez 2015), both can be included alongside each other and the best of both worlds can be leveraged. We start with a visual map exploration by using data stored in the R package `HistData`. We then use this data for an analysis in the Python language. To do this, we need to import the Python interface to R.

```
import rpy2.objects.conversion

import rpy2 as r
import rpy2.objects

import rpy2.interactive as r
import rpy2.interactive.packages
```

The data for the original John Snow analysis is available in R as part of the package `HistData`, which we need to import together with the `ggplot2` package to create figures and maps.

```
r.packages.importr('HistData')
r.packages.importr('ggplot2')
```

In order to have a more streamlined analysis, we define a basic `ggplot` map using the data from `HistData` that we will call on later:

```
%%R

Snow_plot <- ggplot(Snow.deaths, aes(x = x, y=y)) +
  geom_point(data=Snow.deaths, aes(x=x, y=y),
            col="red", pch=19, cex=1.5) +
  geom_point(data=Snow.pumps, aes(x=x, y=y),
            col="black", pch=17, cex=4) +
  geom_text(data=Snow.pumps,
            aes(label = label, x = x, y = y+0.5))+
  xlim(6, 19.5) + ylim(4, 18.5) +
  geom_path(data=Snow.streets,
```

```

aes(x=x,y=y,group=street), col="gray40") +
ggtitle("Pumps and cholera deaths\n
        in 19th century London")+
theme(panel.background = element_rect(fill = "gray85"),
plot.background = element_rect(fill = "gray85"),
panel.grid.major = element_blank(),
panel.grid.minor = element_blank(),
axis.line=element_blank(),
axis.text.x=element_blank(),
axis.text.y=element_blank(),
axis.ticks=element_blank(),
axis.title.x=element_blank(),
axis.title.y=element_blank(),
plot.title = element_text(size = rel(2), face="bold"))

```

At this point, we can access the data:

```
%R head(Snow.deaths$x)
```

which produces the following results:

```
array([13.58801, 9.878124, 14.65398, 15.22057, 13.16265,
13.80617])
```

And move the coordinates from R to Python:

```
X = %R Snow.deaths$x
Y = %R Y=Snow.deaths$y
```

A first visual approximation to the distribution of cholera deaths can then be easily produced:

```
%R plot(X,Y)
```

which gives Fig. 17.1.

A more detailed map can also be produced by calling on the map we defined earlier:

```
%%R Snow_plot
```

which gives the spatial context of the coordinates as in Fig. 17.2.

We can start moving beyond simple visualization and into a more in-depth analysis by adding a kernel density estimate as follows:

```
%%R
## overlay bivariate kernel density contours of deaths
Snow_plot + geom_density_2d()
```

and overlaying it on top of our death locations map as in Fig. 17.3.

This already allows us to get a better insight into Snow's hypothesis of a contaminated pump (the one in Broad Street in particular). To further support this view, we will use some of the most common components of the ESDA toolbox.

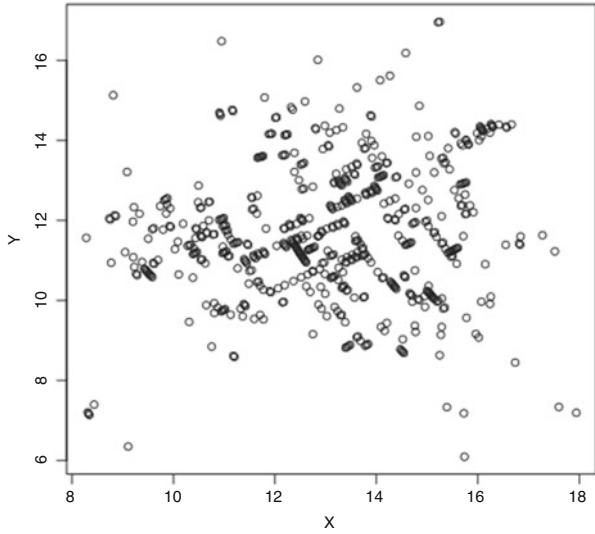


Fig. 17.1 X and Y coordinates of cholera deaths

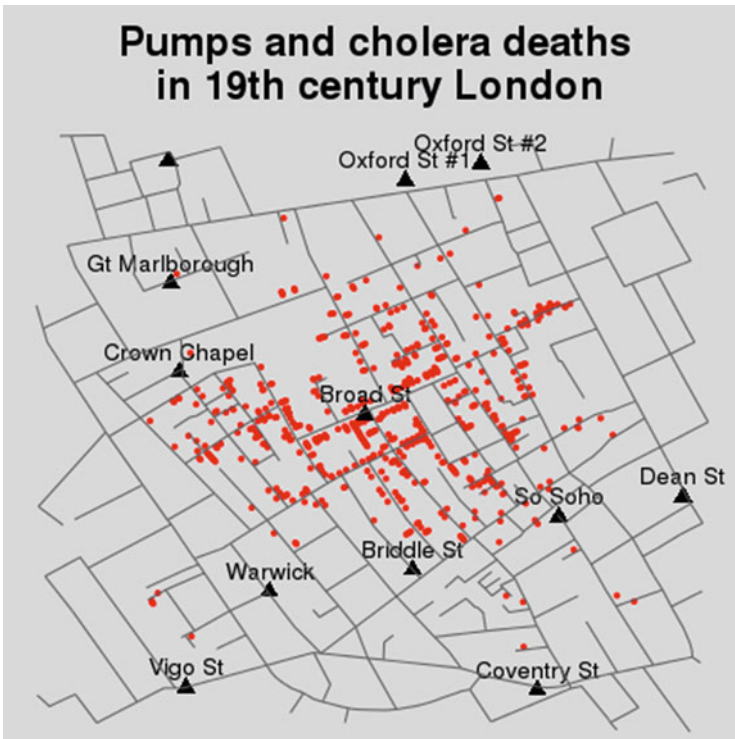


Fig. 17.2 Spatial point of map of cholera deaths

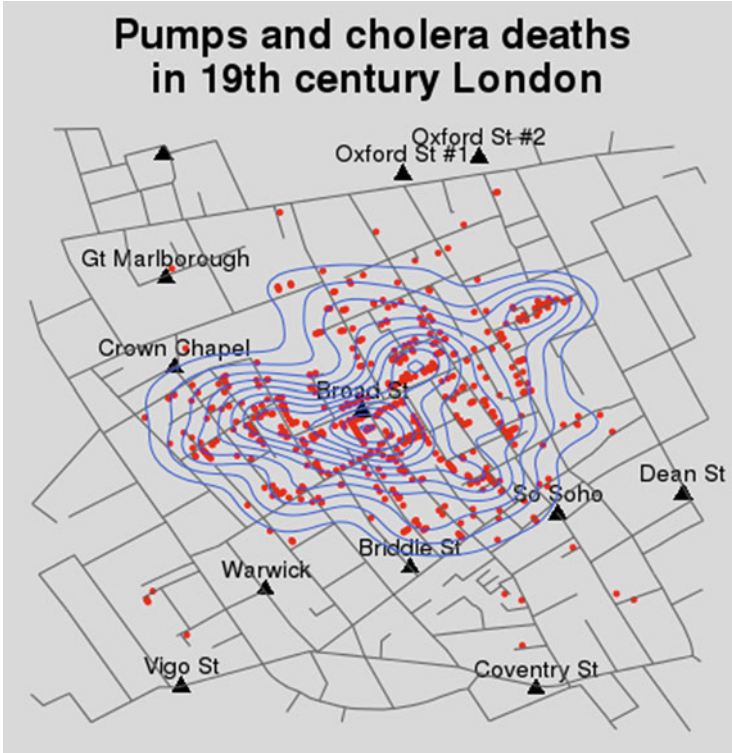


Fig. 17.3 Kernel estimation of cholera deaths

17.3.2 ESDA

Although the original data were locations of deaths at the point level, for this section we will access an aggregated version that reports cholera death counts at the street level. Street segments (lines, topologically) are the spatial unit that probably best characterizes the process we looking at; since we do not have individual data on house units, but only the location of those who passed away, aggregating at a unit like the street segment provides a good approximation of the scale at which the disease was occurring and spreading.

In addition, since the original data are raw counts, we should include a measure of the underlying population. If all maps are the events of interest, unless the population is evenly distributed, the analysis will be biased because high counts could just be a reflection of a large underlying population (everything else being equal, a street with more people will be more likely to have more cholera deaths). In the case of this example, the ideal variable would be to have a count of the inhabitants of each street. Unfortunately, these data are not available, so we need to find an approximation. This will inevitably imply making assumptions and

potentially introducing a certain degree of measurement error. For the sake of this example, we will assume that, within the area of central London covered by the data, population was evenly spread across the street network. This means that the underlying population of one of our street segments is proportional to its length. Following this assumption, if we want to control for the underlying population of a street segment, a good approach could be to consider the number of cholera deaths per (100) metre(s)—a measure of density—rather than the raw count. The polygon file includes building blocks from the Ordnance Survey (OS data l' Crown copyright and database right, 2015).

This part of the analysis will be performed in Python, for which we need to import the libraries required:

```
%matplotlib inline
import seaborn as sns
import pandas as pd
import pysal as ps
import geopandas as gpd
import numpy as np
import matplotlib.pyplot as plt
```

17.3.2.1 Loading and Exploring the Data

Data in this case come from Robin Wilson.⁴

```
# Load point data
pumps = gpd.read_file('data/Pumps.shp')
# Load building blocks
blocks = gpd.read_file('data/polys.shp')
# Load street network
js = gpd.read_file('data/streets_js.shp')
```

To inspect the data and find out the structure as well as the variables included, we can use the head function:

```
print js.head().to_string()
```

with the following output

	Deaths	Deaths_dens	geometry	segIdStr	seg_len
0	0	0.000000	LINSTRING (529521 180866, 529516 180862)	s0-1	6.403124
1	1	1.077897	LINSTRING (529521 180866, 529593 180925)	s0-2	92.773279
2	0	0.000000	LINSTRING (529521 180866, 529545 180836)	s0-3	38.418745
3	0	0.000000	LINSTRING (529516 180862, 529487 180835)	s1-25	39.623226
4	26	18.079549	LINSTRING (529516 180862, 529431 180978)	s1-27	143.808901

Before we move on to the analytical part, we can also create choropleth maps for line data. In the following code snippet, we build a choropleth using the Fisher-Jenks

⁴See: <http://blog.rtwilson.com/john-snows-cholera-data-in-more-formats/>.

classification for the density of cholera deaths in each street segment, and style it by adding a background color, building blocks and the location of the water pumps:

```
# Set up figure and axis
f, ax = plt.subplots(1, figsize=(9, 9))
# Plot building blocks
for poly in blocks['geometry']:
    gpd.plotting.plot_multipolygon(ax, poly, facecolor='0.9')
# Quantile choropleth of deaths at the street level
js.plot(column='Deaths_dens', scheme='fisher_jenks',
        axes=ax, colormap='YlGn')
# Plot pumps
xys = np.array([(pt.x, pt.y) for pt in pumps.geometry])
ax.scatter(xys[:, 0], xys[:, 1], marker='^', color='k', s=50)
# Remove axis frame
ax.set_axis_off()
# Change background color of the figure
f.set_facecolor('0.75')
# Keep axes proportionate
plt.axis('equal')
# Title
f.suptitle('Cholera Deaths per 100m.', size=30)
# Draw
plt.show()
```

which produces Fig. 17.4.

17.3.2.2 Spatial Weights Matrix

A spatial weights matrix is the way geographical space is formally encoded into a numerical form so it is easy for a computer (or a statistical method) to understand. These matrices can be created based on several criteria: contiguity, distance, blocks, etc. Although usually spatial weights matrices are used with polygons or points, these ideas can also be applied with spatial networks made of line segments.

For this example, we will show how to build a simple contiguity matrix, which considers two observations as neighbors if they share one edge. For a street network as in our example, two street segments will be connected if they “touch” each other. Since lines only have one dimension, there is no room for the discussion between “queen” and “rook” criteria, but only one type of contiguity.

Building a contiguity matrix from a spatial network like the streets of London’s Soho can be done with PySAL, but creating it is slightly different, technically. For this task, instead of the `ps.queen_from_shapefile`, we will use the `network` module of the library, which reads a line shapefile and creates a network representation of it. Once loaded, a contiguity matrix can be easily created using the `contiguity weights` attribute. To keep things aligned, we rename the IDs of the matrix to match those



Fig. 17.4 Choropleth map of cholera deaths

in the table and, finally, we row-standardize the matrix, which is a standard `ps.W` object, like those we have been working with for the polygon and point cases:

```
# Load the network
ntw = ps.Network('data/streets_js.shp')
# Create the spatial weights matrix
w = ntw.contiguityweights(graph=False)
# Rename IDs to match those in the 'segIdStr' column
w.remap_ids(js['segIdStr'])
# Row standardize the matrix
w.transform = 'R'
```

Now, the `w` object we have just created comes from a line shapefile, but it is of the same type as if it came from a polygon or point topology. As such, we can inspect it in the same way. For example, we can check who is a neighbor of observation `s0-1`:

```
w['s0-1']
{u's0-2': 0.25, u's0-3': 0.25,
 u's1-25': 0.25, u's1-27': 0.25}
```

Note how, because we have row-standardized them, the weight given to each of the four neighbors is 0.25, which, all together, sum up to one.

17.3.2.3 Spatial Lag

Once we have the data and the spatial weights matrix ready, we can start by computing the spatial lag of the death density. Remember, the spatial lag is the product of the spatial weights matrix and a given variable and that, if W is row-standardized, the result amounts to the average value of the variable in the neighborhood of each observation. We can calculate the spatial lag for the variable `Deaths_dens` and store it directly in the main table with the following line of code:

```
js['w_Deaths_dens'] = ps.lag_spatial(w, js['Deaths_dens'])
```

Let us have a quick look at the resulting variable, as compared to the original one:

```
toprint = js[['segIdStr', 'Deaths_dens', 'w_Deaths_dens']].head()
# Note: next line is for printed version only. On interactive mode,
# you can simply execute 'toprint'
print toprint.to_string()
```

which yields:

	segIdStr	Deaths_dens	w_Deaths_dens
0	s0-1	0.000000	4.789361
1	s0-2	1.077897	0.000000
2	s0-3	0.000000	0.538948
3	s1-25	0.000000	6.026516
4	s1-27	18.079549	0.000000

The way to interpret the spatial lag (`w_Deaths_dens`) for the first observation is as follows: the street segment `s0-2`, which has a density of zero cholera deaths per 100 m, is surrounded by other streets which, on average, have 4.79 deaths per 100 m. For the purpose of illustration, we can check whether this is correct by querying the spatial weights matrix to find out the neighbors of `s0-2`:

```
w.neighbors['s0-1']
[u's0-2', u's0-3', u's1-25', u's1-27']
```

And then checking their values:

```
# Note that we first index the table on the index variable
neigh = js.set_index('segIdStr').loc[w.neighbors['s0-1'],
                                     'Deaths_dens']
neigh

segIdStr
s0-2      1.077897
s0-3      0.000000
s1-25     0.000000
s1-27     18.079549
Name: Deaths_dens, dtype: float64
```

And the average value, which we saw in the spatial lag is 4.79, can be calculated as follows:

```
neigh.mean()
4.7893612696592509
```

For some of the techniques we will be seeing below, it makes more sense to operate with the standardized version of a variable, rather than with the raw one. Standardizing means to subtract the average value and divide by the standard deviation each observation of the column. This can be done easily with a bit of basic algebra in Python:

```
js['Deaths_dens_std'] = (js['Deaths_dens'] -
                        js['Deaths_dens'].mean()) / js['Deaths_dens'].std()
```

Finally, to be able to explore the spatial patterns of the standardized values, sometimes called *z* values, we need to create its spatial lag:

```
js['w_Deaths_dens_std'] =
    ps.lag_spatial(w, js['Deaths_dens_std'])
```

17.3.2.4 Global Spatial Autocorrelation

Global spatial autocorrelation relates to the overall geographical pattern present in the data. Statistics designed to measure this trend thus characterize a map in terms of its degree of clustering and summarize it. This summary can be visual or numerical. In this section, we will walk through an example of each of them: the Moran Plot, and Moran's *I* statistic of spatial autocorrelation.

The Moran plot is a way of visualizing a spatial dataset to explore the nature and strength of spatial autocorrelation. It is essentially a traditional scatter plot in which the variable of interest is displayed against its spatial lag. To be able to interpret values as above or below the mean and their quantities in terms of standard deviations, the variable of interest is usually standardized by subtracting its mean and dividing it by its standard deviation.

Technically speaking, creating a Moran Plot is very similar to creating any other scatter plot in Python, provided we have standardized the variable and calculated its spatial lag beforehand:

```
# Setup the figure and axis
f, ax = plt.subplots(1, figsize=(9, 9))
# Plot values
sns.regplot(x='Deaths_dens_std', y='w_Deaths_dens_std',
            data=js)
# Add vertical and horizontal lines
plt.axvline(0, c='k', alpha=0.5)
plt.axhline(0, c='k', alpha=0.5)
# Display
plt.show()
```

which produces Fig. 17.5.

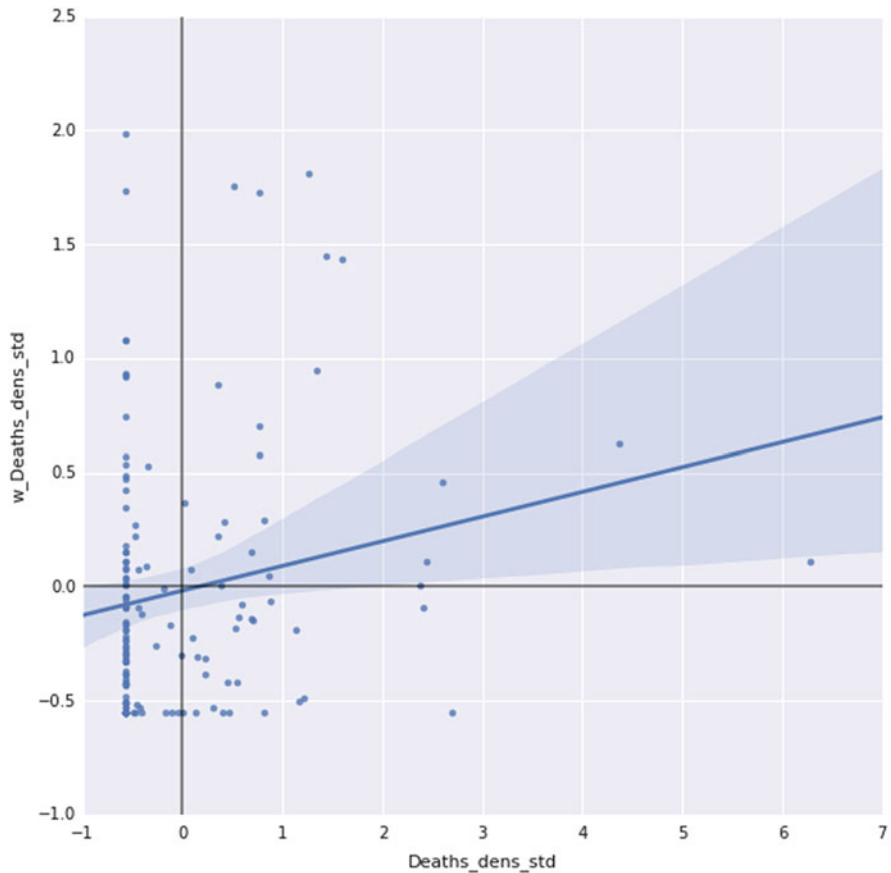


Fig. 17.5 Moran plot of cholera deaths

Figure 17.5 displays the relationship between `Deaths_dens_std` and its spatial lag which, because the W that was used is row-standardized, can be interpreted as the average standardized density of cholera deaths in the neighborhood of each observation. In order to guide the interpretation of the plot, a linear fit is also included in the plot, together with confidence intervals. This line represents the best linear fit to the scatter plot or, in other words, what is the best way to represent the relationship between the two variables as a straight line. Because the line comes from a regression, we can also include a measure of the uncertainty about the fit in the form of confidence intervals (the shaded blue area around the line).

The plot displays a positive relationship between both variables. This is associated with the presence of positive spatial autocorrelation: similar values tend to be located close to each other. This means that the overall trend is for high values to be close to other high values, and for low values to be surrounded by other low values. This, however, does not mean that this is the only pattern in the dataset: there can of course be particular cases where high values are surrounded by low ones, and vice versa. But it means that, if we had to summarize the main pattern of the data in terms of how clustered similar values are, the best way would be to say they are positively correlated and, hence, clustered over space.

In the context of the example, the street segments in the dataset show positive spatial autocorrelation in the density of cholera deaths. This means that street segments with a high level of incidents per 100 m tend to be located adjacent to other street segments also with high number of deaths, and vice versa.

The Moran Plot is an excellent tool to explore the data and get a good sense of how many values are clustered over space. However, because it is a graphical device, it is sometimes hard to condense its insights into a more concise way. For these cases, a good approach is to come up with a statistical measure that summarizes the figure. This is exactly what Moran's I is meant to do.

Very much in the same way the mean summarizes a crucial element of the distribution of values in a non-spatial setting, so does Moran's I for a spatial dataset. Continuing the comparison, we can think of the mean as a single numerical value summarizing a histogram or a kernel density plot. Similarly, Moran's I captures much of the essence of the Moran Plot. In fact, there is an even closer connection between the two: the value of Moran's I corresponds with the slope of the linear fit overlaid on top of the Moran Plot.

In order to calculate Moran's I in our dataset, we can call a specific function in PySAL directly:

```
mi = ps.Moran(js['Deaths_dens'], w)
```

Note how we do not need to use the standardized version in this context as we will not represent it visually.

The method `ps.Moran` creates an object that contains much more information than the actual statistic. If we want to retrieve the value of the statistic, we can do it this way:

```
mi.I
0.10902663995497329
```

The other bit of information we will extract from Moran's I relates to statistical inference: how likely is it that the pattern we observe in the map and Moran's I is not generated by an entirely random process? If we considered the same variable but shuffled its locations randomly, would we obtain a map with similar characteristics?

The specific details of the mechanism to calculate this are beyond the scope of this paper, but it is important to know that a small enough p -value associated with the Moran's I of a map allows rejection of the hypothesis that the map is random. In other words, we can conclude that the map displays more spatial pattern than we would expect if the values had been randomly allocated to a particular location.

The most reliable p -value for Moran's I can be found in the attribute `p_sim`:

```
mi.p_sim
0.045999999999999999
```

That is just below 5% and, by standard terms, it would be considered statistically significant. Again, a full statistical explanation of what that really means and what its implications are is beyond the discussion in this context. But we can quickly elaborate on its intuition. What that 0.046 (or 4.6%) means is that, if we generated a large number of maps with the same values but randomly allocated over space, and calculated the Moran's I statistic for each of those maps, only 4.6% of them would display a larger (absolute) value than the one we obtain from the real data, and the other 95.4% of the random maps would receive a smaller (absolute) value of Moran's I . If we remember again that the value of Moran's I can also be interpreted as the slope of the Moran plot, what we have in this case is that the particular spatial arrangement of values over space we observe for the density of cholera deaths is more concentrated than if we were to randomly shuffle the death densities among the Soho streets, hence the statistical significance.

As a first step, the global autocorrelation analysis can teach us that observations do seem to be positively correlated over space. In terms of our initial goal to find evidence for John Snow's hypothesis that cholera was caused by water in a single contaminated pump, this view seems to align: if cholera was contaminated through the air, it should show a pattern over space—arguably a random one, since air is evenly spread over space—that is much less concentrated than if this was caused by an agent (water pump) that is located at a particular point in space.

17.3.2.5 Local Spatial Autocorrelation

Moran's I is a good tool to summarize a dataset into a single value that informs about its degree of clustering. However, it is not an appropriate measure to identify

areas within the map where specific values are located. In other words, Moran's I can tell us whether values are clustered overall or not, but it will not inform us about where the clusters are. For that purpose, we need to use a local measure of spatial autocorrelation. Local measures consider each single observation in a dataset and operate on them, as opposed to on the overall data, as global measures do. Because of that, they are not good at summarizing a map, but they do provide further insight.

In this section, we will consider Local Indicators of Spatial Association (LISAs), a local counter part of global measures like Moran's I . At the core of these methods is a classification of the observations in a dataset into four groups derived from the Moran Plot: high values surrounded by high values (HH), low values nearby other low values (LL), high values among low values (HL), and vice versa (LH). Each of these groups are typically called "quadrants". An illustration of where each of these groups fall into the Moran Plot can be seen below:

```
# Setup the figure and axis
f, ax = plt.subplots(1, figsize=(9, 9))
# Plot values
sns.regplot(x='Deaths_dens_std', y='w_Deaths_dens_std', data=js)
# Add vertical and horizontal lines
plt.axvline(0, c='k', alpha=0.5)
plt.axhline(0, c='k', alpha=0.5)
ax.set_xlim(-2, 7)
ax.set_ylim(-2.5, 2.5)
plt.text(3, 1.5, "HH", fontsize=25)
plt.text(3, -1.5, "HL", fontsize=25)
plt.text(-1, 1.5, "LH", fontsize=25)
plt.text(-1, -1.5, "LL", fontsize=25)
# Display
plt.show()
```

which gives Fig. 17.6.

So far we have classified each observation in the dataset depending on its value and that of its neighbors. This is only halfway into identifying areas of unusual concentration of values. To know whether each of the locations is a statistically significant cluster of a given kind, we again need to compare it with what we would expect if the data were allocated in a completely random way. After all, by definition every observation will be of one kind or another based on the comparison above. However, what we are interested in is whether the strength with which the values are concentrated is unusually high.

This is exactly what LISAs are designed to do. As before, a more detailed description of their statistical underpinnings is beyond the scope in this context, but we will try to shed some light into the intuition of how they go about it. The core idea is to identify cases in which the comparison between the value of an observation and the average of its neighbors is either more similar (HH, LL) or dissimilar (HL, LH) than we would expect from pure chance. The mechanism to do this is similar to the one in the global Moran's I , but applied in this case to each observation, results in as many statistics as the original observations.

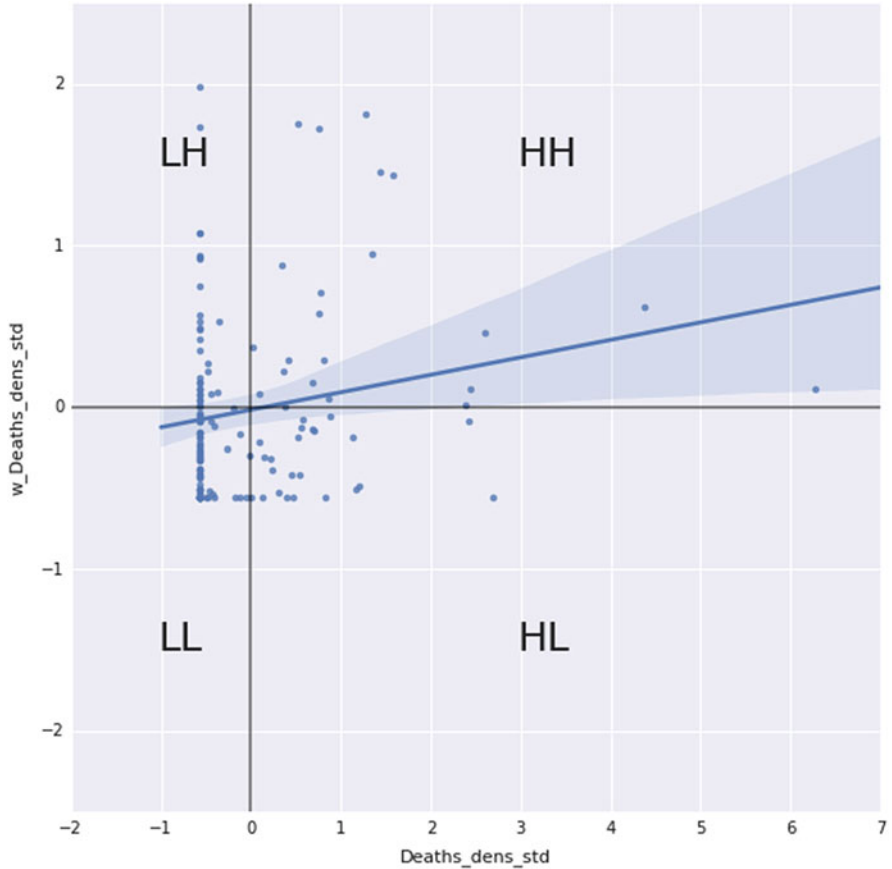


Fig. 17.6 Moran plot of cholera deaths with “quadrants”

LISAs are widely used in many fields to identify clusters of values in space. They are a very useful tool that can quickly return areas in which values are concentrated and provide suggestive evidence about the processes that might be at work. For that, they have a prime place in the exploratory toolbox. Examples of contexts where LISAs can be useful include: identification of spatial clusters of poverty in regions, detection of ethnic enclaves, delineation of areas of particularly high/low activity of any phenomenon, etc.

In Python, we can calculate LISAs in a very streamlined way thanks to PySAL:

```
lisa = ps.Moran_Local(js['Deaths_dens'].values, w)
```

All we need to pass is the variable of interest—density of deaths in this context—and the spatial weights that describes the neighborhood relations between the different observation that make up the dataset.

Because of their very nature, looking at the numerical result of LISAs is not always the most useful way to exploit all the information they can provide. Remember that we are calculating a statistic for every single observation in the data so, if we have many of them, it will be difficult to extract any meaningful pattern. Instead, what is typically done is to create a map, a cluster map as it is usually called, that extracts the significant observations (those that are highly unlikely to have come from pure chance) and plots them with a specific color depending on their quadrant category.

All of the needed pieces are contained inside the LISA object we have created above. But, to make the map making more straightforward, it is convenient to pull them out and insert them in the main data table, `js`:

```
# Break observations into significant or not
js['significant'] = lisa.p_sim < 0.05
# Store the quadrant they belong to
js['quadrant'] = lisa.q
```

Let us stop for second on these two steps. First, look at the significant column. Similarly as with global Moran's I , PySAL is automatically computing a p -value for each LISA. Because not every observation represents a statistically significant one, we want to identify those with a p -value small enough that to rule out the possibility of obtaining a similar situation from pure chance. Following a similar reasoning as with global Moran's I , we select 5% as the threshold for statistical significance. To identify these values, we create a variable, `significant`, that contains True if the p -value of the observation has satisfied the condition, and False otherwise. We can check this is the case:

```
js['significant'].head()
0    False
1    False
2    False
3    False
4     True
Name: significant, dtype: bool
```

And the first five p -values can be checked by:

```
lisa.p_sim[:5]
array([ 0.418,  0.085,  0.301,  0.467,  0.001])
```

Note how only the last one is smaller than 0.05, as the variable `significant` correctly identified.

The second column denotes the quadrant each observation belongs to. This one is easier as it comes built into the LISA object directly:

```
js['quadrant'].head()
0    3
1    3
```

```
2    3
3    3
4    4
```

Name: quadrant, dtype: int64

The correspondence between the numbers in the variable and the actual quadrants is as follows:

- 1: HH
- 2: LH
- 3: LL
- 4: HL

With these two elements, significant and quadrant, we can build a typical LISA cluster map.

```
# Setup the figure and axis
f, ax = plt.subplots(1, figsize=(9, 9))
# Plot building blocks
for poly in blocks['geometry']:
    gpd.plotting.plot_multipolygon(ax, poly, facecolor='0.9')
# Plot baseline street network
for line in js['geometry']:
    gpd.plotting.plot_multilinestring(ax, line, color='k')
# Plot HH clusters
hh = js.loc[(js['quadrant']==1) & (js['significant']==True),
            'geometry']
for line in hh:
    gpd.plotting.plot_multilinestring(ax, line, color='red')
# Plot LL clusters
ll = js.loc[(js['quadrant']==3) & (js['significant']==True),
            'geometry']
for line in ll:
    gpd.plotting.plot_multilinestring(ax, line, color='blue')
# Plot LH clusters
lh = js.loc[(js['quadrant']==2) & (js['significant']==True),
            'geometry']
for line in lh:
    gpd.plotting.plot_multilinestring(ax, line, color='#83cef4')
# Plot HL clusters
hl = js.loc[(js['quadrant']==4) & (js['significant']==True),
            'geometry']
for line in hl:
    gpd.plotting.plot_multilinestring(ax, line, color='#e59696')
#gpd.plotting.plot_multilinestring(ax, line, color='#e59696',
    linewidth=5)
# Plot pumps
xys = np.array([(pt.x, pt.y) for pt in pumps.geometry])
ax.scatter(xys[:, 0], xys[:, 1], marker='^', color='k', s=50)
# Style and draw
```

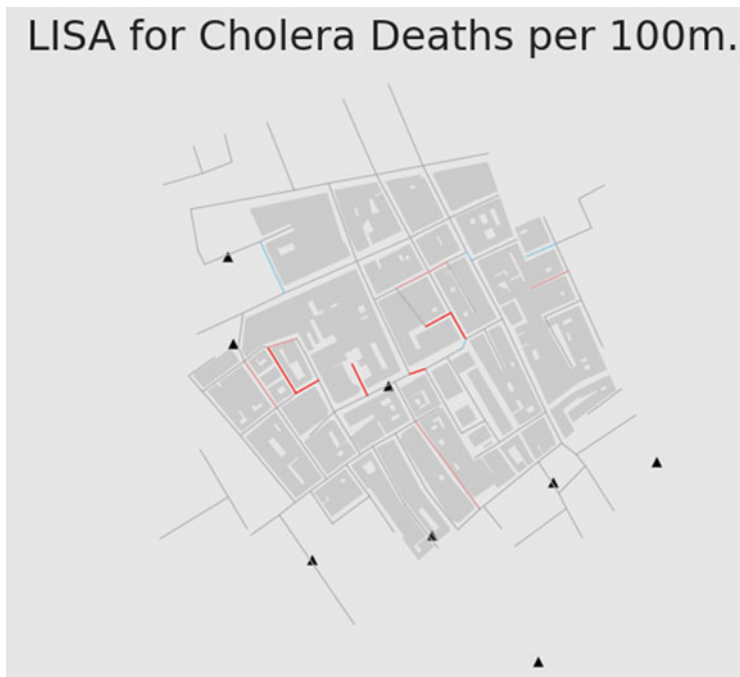


Fig. 17.7 LISA cluster map cholera deaths

```
f.suptitle('LISA for Cholera Deaths per 100m.', size=30)
f.set_facecolor('0.75')
ax.set_axis_off()
plt.axis('equal')
plt.show()
```

which yields Fig. 17.7.

Figure 17.7 displays the streets of the John Snow map of cholera and overlays on top of it the observations that have been identified by the LISA as clusters or spatial outliers. In bright red we find those street segments with an unusual concentration of high death density surrounded also by high death density. This corresponds with segments that are close to the contaminated pump, which is also displayed in the center of the map. In light red, we find the first type of spatial outliers. These are streets with high density but surrounded by low density. Finally, in light blue we find the other type of spatial outlier: streets with low densities surrounded by other streets with high density.

The substantive interpretation of a LISA map needs to relate its output to the original intention of the analyst who created the map. In this case, our original idea was to find support in the data for John Snow's thesis that cholera deaths were caused by a source that could be traced back to a contaminated water pump. The results seem to largely support this view. First, the LISA statistic identifies a few

clusters of high densities surrounded by other high densities, discrediting the idea that cholera deaths were not concentrated in specific parts of the street network. Second, the location of all of these HH clusters centers around only one pump, which in turn is the one that ended up being contaminated.

Of course, the results are not entirely clean; they almost never are with real data analysis. Not every single street segment around the pump is identified as a cluster, while we find others that could potentially be linked to a different pump (although when one looks at the location of all clusters, the pattern is clear). At this point it is important to remember issues in the data collection and the use of an approximation for the underlying population. Some of that could be at work here. Also, since this is real world data, many other factors that we are not accounting for in this analysis could also be affecting this. However, it is important to note that, despite all of those shortcomings, the analysis points into very much the same direction that John Snow concluded more than 150 years ago. What it adds to his original assessment is the power and robustness that comes with statistical inference and does not with visualization only. Some might have objected that, although convincing, there was no statistical evidence behind his original map, and hence it could have still been the result of a purely random process in which water had no role in transmitting cholera. Upon the results presented here, such a view is much more difficult to sustain.

17.4 Concluding Remarks

This chapter deals with reproducibility and Open Science, specifically in the realm of regional science. The growing emphasis on geographically referenced data of increasing size and interest in quantitative approaches leads to an increasing need for training in workflow design and guidance in choosing appropriate tools. We argue that a proper workflow design has substantial benefits, including reproducibility (obviously) and efficiency. If it is possible to easily recreate the analysis and the resulting output in presentation or paper format, then slight changes induced by referees, supervisor or editors can be quickly processed. This is not only important in terms of time saving, but also in terms of accountability and transparency. In more practical terms, we illustrate the advocated approach by reproducing John Snow's famous cholera analysis from the nineteenth century, using a combination of R and Python code. The analysis includes contemporary spatial analytic methods, such as measuring global and local spatial autocorrelation measures.

In general, it is not so much the reproducible part but the openness part that some researchers find hard and counterintuitive to deal with. This is because the "publish or perish" ethos that dominates modern academic culture also rails against openness. Why open up all resources of your research so that others might benefit and scoop you in publishing first? A straightforward rebuttal to this would be: "Why publish then after all if you are hesitant to make all materials public?" And if you agree about this, why open up not only after the final phase when the paper has been accepted, but earlier in the research cycle? Some researchers are so extreme in this

that they even share the writing of their research proposals with the outside world. Remember, with versioning control systems, such as Git, you can always prove, via timestamps, that you came up with the idea earlier than someone else.

Complete openness and thus complete reproducibility is often not feasible in the social sciences. Data could be proprietary or privacy-protected and expert interviews or case studies are notoriously hard to reproduce. And sometimes, you do in fact face cutthroat competition to get your research proposal rewarded or paper accepted. However, opening up your research, whether in an early, late or final phase definitely can reward you with large benefits. Mostly, because your research becomes more visible and is thus recognized earlier and credited. However, and most importantly, the scientific community most likely benefits the most as results, procedures, code and data are disseminated faster, more efficiently and with a much wider scope. As Rey (2009) has argued, free revealing of information can lead to increased private gains for the scientist as well as enhancing scientific knowledge production.

References

- Arribas-Bel D (2016) Geographic data science'15. <http://darribas.org/gds15>
- Arribas-Bel D, de Graaff T (2015) Woow-ii: workshop on open workflows. *Region* 2(2):1–2
- BusinessDictionary (2016) Workflow [Online; accessed 15-June-2016]. <http://www.businessdictionary.com/definition/workflow.html>
- Case A, Deaton A (2015) Rising morbidity and mortality in midlife among white non-hispanic americans in the 21st century. *Proc Natl Acad Sci* 112(49):15078–15083
- Gandrud C (2013) *Reproducible research with R and R studio*. CRC, Boca Raton, FL
- Healy K (2011) Choosing your workflow applications. *Pol Methodologist* 18(2):9–18
- Hempel S (2006) The medical detective: John Snow and the mystery of cholera. Granta, London
- Perez F (2015) Ipython: from interactive computing to computational narratives. In: 2015 AAAS Annual Meeting (12–16 February 2015)
- Rey SJ (2009) Show me the code: spatial analysis and open source. *J Geogr Syst* 11:191–207
- Rey SJ (2014) Open regional science. *Ann Reg Sci* 52(3):825–837
- Stodden V, Leisch F, Peng RD (2014) *Implementing reproducible research*. CRC, Boca Raton, FL

Daniel Arribas-Bel is a Lecturer in Geographic Data Science at the University of Liverpool. He has held positions as Lecturer in Human Geography at the University of Birmingham, postdoctoral researcher at the Department of Spatial Economics at the VU University (Amsterdam), and postdoctoral researcher at the GeoDa Center for Geospatial Analysis and Computation at Arizona State University. Trained as an economist, Dani is interested in the spatial structure of cities and in the quantitative and computational methods required to leverage the power of the large amount of urban data increasingly becoming available. He is also part of the team of core developers of PySAL, the open-source library written in Python for spatial analysis.

Thomas de Graaff is assistant professor at the Department of Spatial Economics, Free University Amsterdam. His primary research interests are spatial interactions between households and firms; spatial econometrics; migration patterns; regional

performance; and reproducibility of scientific research. Previous positions were at the Netherlands Bureau of Economic Policy Analysis (CPB) and the Netherland Environmental Assessment Agency (PBL). Dr. De Graaff earned the Ph.D. in economics from the Department of Spatial Economics at the Free University Amsterdam in 2002.

Sergio Rey is professor, School of Geographical Sciences and Urban Planning, Arizona State University (ASU). His research interests focus on the development, implementation, and application of advanced methods of spatial and space-time data analysis. His substantive foci include regional inequality, convergence and growth dynamics as well as neighborhood change, segregation dynamics, spatial criminology and industrial networks. Previous faculty positions were at the Department of Geography, San Diego State University and a visiting professor at the Department of Economics, University of Queensland. Dr. Rey earned the Ph.D. in geography from the University of California Santa Babara in 1994.