Taylor Arnold • Lauren Tilton

Humanities Data in R

Exploring Networks, Geospatial Data, Images, and Text

Springer

Taylor Arnold
Yale University
New Haven, CT, USA

Lauren Tilton
Yale University
New Haven, CT, USA

ISSN 2199-0956 ISSN 2199-0964 (electronic)
Quantitative Methods in the Humanities and Social Sciences
ISBN 978-3-319-20701-8 ISBN 978-3-319-20702-5 (eBook)
DOI 10.1007/978-3-319-20702-5

Library of Congress Control Number: 2015945113

Springer Cham Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer International Publishing AG Switzerland is part of Springer Science+Business Media (www.springer.com)

Chapter 7
Geospatial Data

Abstract In this chapter, we will look at how to work with raster and vector spatial data in R. Particular attention is placed on the process of merging various data types to create enriched datasets for further analysis.

7.1 Introduction

There is a popular phrase thrown around by those working with spatial data claiming that "80 % of data contains a spatial component", likely dating to a weaker statement made by Franklin and Hane specifically regarding data contained in government databases [6]. While actually quantifying the "amount of data" with a spatial component is likely impossible (and meaningless), the premise that a majority of datasets contain some spatial information is a valid one. Consider a dataset containing a record for every item held in a particular public library. It may contain explicit geospatial data such as the address of the branch where each item is housed, but there is a substantial amount of implicit spatial data which could also be added and explored such as the location of first publication or the birthplaces of the authors. Given the preponderance of geospatial data in general, and its particular importance to work in the humanities, this chapter introduces methods for exploring and visualizing a number of spatial data types within the R language.

Another popular set of tools for working with geospatial data are geographic information systems (GIS) such as the open source QGIS [12] and the proprietary ArcGIS [1]. Almost all of the same analytic functionality offered by these systems is also available within R, though ArcGIS does benefit from coming with a massive amount of ready to use data. We mention these here because GIS is sometimes mis-attributed as a type of *analysis*, rather than a type of *tool*.

T. Arnold, L. Tilton, *Humanities Data in R*, Quantitative Methods
in the Humanities and Social Sciences, DOI 10.1007/978-3-319-20702-5_7

7.2 From Scatter Plots to Maps

Basic geospatial plots can be thought of as simple scatter plots with an x-coordinate equal to longitude and a y-coordinate equal to latitude. Consider the following dataset of major border crossings into West Berlin between 1961 and 1989.[1]

```
> berlin <- read.csv(file="data/ch07/berlinBorderCrossing.csv",
+                    as.is=TRUE)
> berlin
                                    Crossing Latitude Longitude
1                          Bornholmer Straße 52.55474  13.39766
2                             Chausseestraße 52.54048  13.36966
3                           Invalidenstraße 52.52790  13.37406
4                           Friedrichstraße 52.50762  13.39042
5                    Heinrich-Heine-Straße 52.50472  13.41161
6                           Oberbaumbrücke 52.50197  13.44587
7                              Sonnenallee 52.46167  13.47844
8                  Friedrichstraße station 52.52028  13.38704
9  Grenzübergangsstelle Drewitz-Dreilinden 52.41547  13.19716
10                        Glienicker Brücke 52.41349  13.09014
11                               Heerstraße 52.52906  13.11922
12                     Berlin-Heiligensee 52.62806  13.24160
```

We will start by just identifying the crossings clustered in the city center, specifically crossing 1–6 and 8. These could have been determined by an initial plot, though here we will just take them as known. A simple labeled plot suffices to visually display these locations, as seen in Fig. 7.1a.

```
> cityCenterId <- c(1:6,8)
> plot(x=berlin$Longitude[cityCenterId],
+      y=berlin$Latitude[cityCenterId],
+      pch=19,
+      axes=FALSE,
+      xlim=range(berlin$Longitude[cityCenterId]))
> text(x=berlin$Longitude[cityCenterId],
+      y=berlin$Latitude[cityCenterId],
+      labels=berlin$Crossing[cityCenterId],
+      pos=4,
+      col="blue")
```

Unless someone has a very good sense of Berlin's geography, however, this plot is not particularly interesting. No additional information such as streets, political borders, parks, and other landmarks is present.

It would be great if we could display a map of the city underneath the points, similar to the way a web-based platform such as Google or Apple displays a street map underneath search results. Fortunately, this is easy to do in R using the **snippets** package [14].[2] The function `osmap` pulls map images from an application known as a *tile map service*, a web-based application which stores a set of small map images

[1] Rough locations are from http://www.berlin.de/mauer/grenzuebergaenge/index/index.de.php; exact coordinates were determined by hand.

[2] See Chap. 11 for special installation instructions for this package.

in a hierarchical fashion [11]. When trying to load a map on a website with a large amount of tiles or with a slow connection, often square patches of the map sporadically appear. Each of these patches is a single image served by a tile map service. By default, the `osmap` function determines the coordinates of the current plot, pulls approximately 25 tiles to cover the plot region from OpenStreetMaps (the namesake of the function) [10], and overlays these map images onto the current plot.

```
> library(snippets)
> plot(x=berlin$Longitude[cityCenterId],
+      y=berlin$Latitude[cityCenterId],
+      pch=19,
+      axes=FALSE,
+      xlim=range(berlin$Longitude[cityCenterId]))
> osmap()
> points(x=berlin$Longitude[cityCenterId],
+        y=berlin$Latitude[cityCenterId],
+        pch=19)
```

Notice the need to make a call to the `points` function due to the original points being covered up by the map tiles. The output is shown in Fig. 7.1b. We have succeeded in adding a significant amount of additional detail but unfortunately the resulting maps have come out rather distorted. From a pragmatic standpoint, most of the features are legible, but on aesthetic grounds it would be better if we could adjust this distortion. When running the code in an interactive R session, it is easy to simply resize the window as appropriate. If plotting to a graphics device, a bit of trial and error will all quickly suffice in producing a reasonable plot.

What if you need the map to fit into a pre-determined image size, or simply do not want to be bothered with trial and error? There is an additional input parameter to the default `plot` function in R that comes to the rescue. When set, the `asp` parameter sets the *aspect ratio*, the ratio of the scale on the y-axis to the scale of the x-axis, of a plot. If this is set to the ratio of the length of a degree of latitude to the length of a degree of longitude, the resulting map will be undistorted. Such an effect is achieved by extending the range of one of the axes. The correct ratio changes depending on the latitude, with a value of 1 at the equator and 2 at 60°N and 60°S; the exact value is given by the inverse of absolute value of the cosine of the degree of latitude (in radians). The parameter is not overly sensitive for small plots; here it is sufficient to take the exact ratio implied by the first data point (approximately 1.64):

```
> plot(x=berlin$Longitude[cityCenterId],
+      y=berlin$Latitude[cityCenterId],
+      pch=19,
+      axes=FALSE,
+      asp=1/abs(cos(berlin$Latitude[1]*pi/180)),
+      xlim=range(berlin$Longitude) + c(0,0.2))
```

Following this command with a call to `osmap` and re-plotting the points as before yields Fig. 7.1c. Notice that the new plot extends slightly farther in the horizontal directions in order to maintain our desired aspect ratio.

Figure 7.1: Location of the primary civilian ground border crossings between West Berlin and East Germany between 1961 and 1989. (**a**) Geospatial data as a scatter plot. (**b**) Open street map data of the city center. (**c**) View of crossings with adjust aspect ratio.

We have now constructed a map which is both aesthetically pleasing and useful in adding a geographical perspective to the West Berlin border crossing dataset. It highlights, for example, how arbitrarily the city was split in half, adding perspective to the challenges faced by those living on either side of wall in post-war Berlin. The tiles we used to construct map are based on modern-day roads and landmarks, not all of which existed in 1961. Could we use historic data for the maps?

There are no technical limitations to setting up a tile map service based on historic data. Configuring and maintaining a tile map service, however, is quite involved and outside the scope of this text. Doing so would also require access to a substantial amount of historic data in order to be able to construct the map tiles; such data does not typically exist in an easily digestible format (if at all). A solution to this problem, which should suffice for most uses, is to pull tiles from a server that is built from modern data but does so in a minimalistic fashion. For example, Stamen Design provides high contrast black and white minimalistic tiles for free under a fairly permissive Creative Commons licence. Using these tiles requires only one additional parameter to the `osmap` function.

```
> osmap(tiles.url="http://c.tile.stamen.com/toner/")
```

The result of this map, Fig. 7.2a, still labels a few major streets and parks, which existed well before the 1960s, and removes many of the potentially anachronistic features. We also think it looks better when printed in black and white, but, of course, customize to your taste. The map certainly is not perfect, but it does a reasonable job of representing post-war Berlin without the hundreds, if not thousands, of hours spent curating historical street data and building a tile service. Another solution for historic locations and areas that have recently undergone rapid development is plain terrain maps.[3] Fortunately, many more open tile map services exist with varying features and aesthetics.[4]

Finally, let us briefly consider building a larger plot of the entire border crossing dataset to include those locations that are not within the city center. Programmatically altering the map region is very easy; the same exact code without the subsetting command, `[cityCenterId]`, produces the larger map without any changes required to the `osmap` function. The output shown in Fig. 7.2b demonstrates that the new tiles have been pulled and the aspect ratio handled correctly. Notice that the larger map has not just pulled *more* tiles but has instead pulled a *different* set of tiles. The same number of images were requested from Stamen Design's servers in both versions of the map. Each tile in the larger region plot simply shows a less detailed version of a larger area (notice the omitted streets). The labeling has also changed to be appropriate for the lower resolution plot: the Großer Tiergarten is still shaded in but now lacks a label, whereas a label has been added to indicate the position of the city of Berlin (which is over-plotted by our text for Friedrichstraße).

[3] Stamen Design also produces a nice terrain tile map service under the same permissive license at: http://maps.stamen.com/terrain-background.

[4] See the links at http://wiki.openstreetmap.org/wiki/Tile_servers for a good starting point.

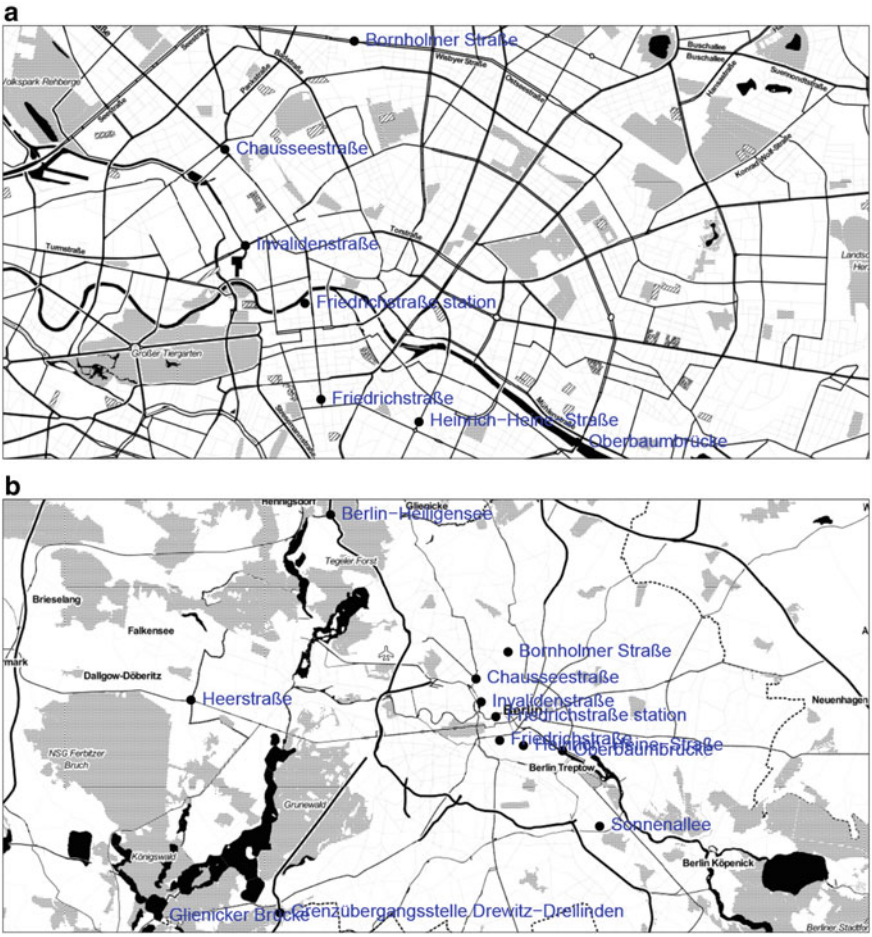

Figure 7.2: Location of the primary civilian ground border crossings between West Berlin and East Germany between 1961 and 1989. (**a**) Geospatial data as a scatter plot. (**b**) Open street map data of the city center.

7.3 Map Projections and Input Formats

A tile map service is a great way to quickly turn a basic geospatial scatter plot into a publication ready visualization with a single extra line of code. If we want to add anything more involved a different approach is required. A map tile has no information *per se* about roads, cities, or geographic features. It is simply a collection of pixels, an example of a *raster graphics format*, and nothing more than an image. This is the reason that our previous example of mapping two plots with slightly different zoom levels required an entirely new set of tiles. In contrast to raster data, *vector graphics* are instead content aware, with information regarding

the shape of every element embedded into the data. Metadata, such as an element name or classification, is often also attached to each object, such as a road or body of water, in a vector graphic. Vector graphics often require significantly larger file sizes, which is why they are not used to send data through web-based mapping services. However, the additional information they contain allow a wide range of geospatial visualizations and analyses, which we will explore for the remainder of this chapter.

Shapefiles are a very popular format for storing vectorized geospatial data. Developed by ESRI, the company responsible for the proprietary ArcGIS software system, the standards for the format were published as an open data specification in a 1998 white paper [13]. Data are distributed in this format by organizations such as the United Nations, the European Union, and the United States Census Bureau. The term shapefile is a misnomer, as the format actually requires a collection of three or more files with various file extensions to be co-located in the same directory (often distributed as a zip archive). At a minimum the `.shp`, `.shx`, and `.dbf` files are required; a `.prj` is almost always also included.

Two R packages will provide us with the majority of the functionality needed to work with vectorized geospatial data: **sp** and **maptools** [4, 2]. Loading a shapefile into R can be done with a single function call; the input string should point to the filenames for the data without specifying the extension (e.g., .shp). To start, we load a file provided by the US Census Bureau providing basic census data from 2010 at a state level attached to geospatial vector data defining the shape of all 50 states.

```
> library(sp)
> library(maptools)
> state <- readShapeSpatial(fn="data/ch07/State_2010Census_DP1")
> class(state)
[1] "SpatialPolygonsDataFrame"
attr(,"package")
[1] "sp"
> dim(state)
[1]  52 195
> dim(as.data.frame(state))
[1]  52 195
```

The resulting object is a `SpatialPolygonsDataFrame`, a special class constructed by the **sp** package. In addition to purely geospatial information, it also has an internal data frame with metadata for each shape in the shapefile. We were able to access this internal data frame with the `as.data.frame` function. In this case, the data frame has 195 columns, primarily containing raw counts of un-sampled, self-reported data from the 2010 Census. The 52 rows correspond to the 50 states, Washington D.C., and Puerto Rico. Unlike many other R packages we have used that define customized objects, the **sp** package did not define a way of nicely printing spatial polygon data frames; printing the object simply prints a long unstructured list of (usually) unintelligible output.

It will be easier to display a map of the United States in the limited space of this text by first removing Alaska and Hawaii. Taking a subset of the objects in a spatial

polygon data frame conveniently uses the same notation used for taking a subset of rows in a standard data frame.

```
> index <- (as.data.frame(state)$STUSPS10 %in% c("AK", "HI"))
> state <- state[!index,]
> dim(state)
[1]  50 195
```

A basic plot of a spatial polygon object can be generated by calling the `plot` function of the object.

```
> plot(state)
```

Figure 7.3a shows the output of this command. As a basic plot of spatial data, it works quite well. As vector data, we can output this plot to a device of any size without the output becoming grainy, a powerful benefit of this type of data.

The spatial plot we already produced plotted shapes the same way we treated scatter plots in Sect. 7.2 by treating raw longitude and latitude as the x and y coordinates of the plot. Such a simplification was fine when dealing with data at the scale of a city. However, when looking at large scales, it is often advantageous to plot spatial data in a projection format that accounts for the spherical nature of the world. Converting between projection formats can be done within R; the first step involves identifying which projection our current data was originally loaded in with (here, just longitude and latitude) and assigning it to the `proj4string` of the spatial polygon data frame. The notation used to define projections is known as PROJ.4 [5]. It is very extensive, and rather than explaining it in detail, we will simply lay out how to use it to describe two of the most useful variants.

```
> projectionObj <- CRS(projargs="+proj=longlat")
> proj4string(state) <- projectionObj
```

The process of converting from one projection to another is handled by the R package **rgdal** [3]. It takes an existing spatial object and a new projection format and returns a copy of its input in the new projection format. Here, we will convert to the Universal Transverse Mercator projection [8, 9]. This projection requires specifying the desired reference "zone" (a number between 1 and 60), which roughly specifies where the "middle" of the map should be. For the continental United States, zone 14 works particularly well.

```
> library(rgdal)
> stateTrans <- spTransform(x=state,
+     CRSobj=CRS("+proj=utm +zone=14"))
```

Calling the plot function directly on the projected object `stateTrans` produces the outline of the states shown in Fig. 7.3b. More information will help make the map legible, so let us add the state names to this plot. A direct call to the `text` function using latitude and longitude would not be sufficient because the shapes are now plotted in a new coordinate system. To determine the location of the new shapes the `gCentroid` function is used; it returns the *centroid* (the geometric center) of each state in the projected coordinate system. Finally, using the coordinates from these centroids, a call to the `text` function produces the final figure.

```
> centroid <- gCentroid(spgeom=stateTrans, byid=TRUE)
> head(centroid)
SpatialPoints:
          x       y
0 -198256.2 4796062
1 2289521.4 4749735
2 1885001.1 4602892
3 -155119.1 3830282
4 2444657.6 4555463
5 2797519.2 4985976
Coordinate Reference System (CRS) arguments: +proj=utm
+zone=14 +ellps=WGS84
> text(x=centroid$x,
+      y=centroid$y,
+      label=as.data.frame(stateTrans)$NAME10,
+      cex=0.7)
```

The resulting map produces a distinctly less distorted representation of the United States.

Now that we have shown the basics of loading, plotting, and projecting vector data in R, we finish this section by showing how to visualize the census data on our map. We will look at housing data from the 2010 US Census.

A researcher might be interested in how housing is used in the United States, focused on the locations of recreational housing. They are confident that Florida will be a popular state for snowbirds seeking respite from the snow in the northeast. To confirm this assumption and to see what other states have a high density of recreational homes, they turn to the census. First, the proportion of housing units (DP0180001) declared as being "seasonal, recreational, or occasional" (DP0180008) for each state is calculated and converted to quantized bin IDs.

```
> stateTransData <- as.data.frame(state)
> perHouseRec <- stateTransData$DP0180008 / stateTransData$DP0180001
> bins <- quantile(perHouseRec, seq(0,1,length.out=8))
> binId <- findInterval(perHouseRec, bins)
```

Figure 7.3: (**a**) Geospatial data as a scatter plot. (**b**) Open street map data of the city center. (**c**) View of additional crossings farther outside the city center.

These bins are scaled to a factor of 5 (trial and error to achieve a visually pleasing plot) and simply given as an additional parameter to the plot function.

```
> densityVals <- seq_len(length(bins)) * 5
> plot(stateTrans, density=densityVals[binId])
```

The output is shown in Fig. 7.3c, with a higher proportion of seasonal or recreational houses corresponding to darker shaded states. The relatively high values in Florida, Arizona, South Carolina, and Maine correlate with the commonly held assumption that these are popular spots for Winter/Summer homes.

7.4 Enriching Tabular Data with Geospatial Data

The power of vectorized geospatial data can be best appreciated by seeing how it is used to add context to other datasets. In this section, we study a dataset of metadata corresponding to documentary photographs taken between 1935 and 1945 for the US Federal Government. For each of the 90,000 geo-locatable records (each record corresponds to a single photographic negative) information is available regarding the date, name of the photographer, and the approximate latitude and longitude where the photograph is believed to have been taken. Given that the photographers were often given a great deal of leeway regarding where they traveled, and collectively covered a vast proportion of the country, an interesting academic question arises around comparing and contrasting the subject matter each photographer chose to capture. Here, we address this question by merging the photographic metadata with county-level US Census data to help characterize differences in the locales that were visited.

We begin by loading into R the tabular metadata and removing data rows with missing geospatial data. We further limit the data set to the top 20 photographers (by count) in the collection because the remaining photographers took a very small number of photographs, often just one or two.

```
> z <- read.csv(file="data/ch07/photoDatasetAllRaw.csv", as.is=TRUE)
> z <- z[!is.na(z$latitude) & !is.na(z$longitude) & !is.na(z$pname),]
> pnameSet <- names(sort(table(z$pname),TRUE))[1:20]
> z <- z[z$pname %in% pnameSet,]
> head(z)
             cnumber       pname year longitude latitude
1 LC-DIG-fsac-1a33849 Jack Delano 1941 -71.31617 42.63342
2 LC-DIG-fsac-1a33850 Jack Delano 1940 -71.31617 42.63342
3 LC-DIG-fsac-1a33851 Jack Delano 1940 -71.01838 42.08343
4 LC-DIG-fsac-1a33852 Jack Delano 1940 -71.01838 42.08343
5 LC-DIG-fsac-1a33853 Jack Delano 1940 -71.01838 42.08343
6 LC-DIG-fsac-1a33857 Jack Delano 1941 -71.31617 42.63342
```

We need to convert this data frame into a spatial object in order to manipulate it with the **sp** package. Previously we had loaded the state shapes files as a spatial polygon data frame. Each record in the photographic metadata is a point rather than a polygon; a method exists for constructing a very similar object called a spatial

7.6 Further Extensions

A natural next step from this chapter is the text *Applied Spatial Data Analysis with R* [4]. It first extends the methods mentioned here to a wider range input formats, projections, and applications. In the second half of the text more analytic techniques for modeling spatial data are explored. The level of presentation is very similar to this text and, similarly, emphasizes applied data analysis and the "how-to" of programming in R. The text *Spatial statistics and modeling* by Carlo Gaetan and Xavier Guyon provides a far more extensive introduction to spatial statistics but requires a background in mathematical analysis and theoretical statistics [7].

Practice

1. Construct a small dataset of important places in your life, possibly including things such as your home location, work location, and favorite coffee shop. Manually add latitude and longitude to the data (can be extracted from many online tools such as Google Maps) and create plots using the `snippets` function. Try increasing and decreasing the extent of the plot by adding/removing locations. What features appear and disappear as the map area becomes smaller?

2. Replicate the plot shown in Fig. 7.3a and save the output in these formats: tiff, pdf, jpeg, png, bmp. Use the default height and width for the plots. What differences do you see between the file sizes of the output? Now construct plots with twice the height and width of the originals. Compare the new and old file sizes. Do they all scale up the same way? Why or why not? When might you prefer a jpeg over a pdf (and vice versa)?

3. We have already worked with point and polygons in this chapter, but there is one other data type we did not use: SpatialLines. The US Census Bureau maintains a shape files (by state) for all primary and secondary streets in the United States, downloadable here: `ftp://ftp2.census.gov/geo/tiger/TIGER2014/ROADS/`. Read in these shapefiles with the command `readShapeLines`. What type of object is returned? It can be manipulated almost exactly like its point and polygon variants; try plotting for instance. Produce a plot with only the primary roads (indicated in the attached data frame). Now produce a plot where all of the roads are present, but the primary roads are colored in red.

References

[1] Hawthorn L Beyer. Hawth's analysis tools for arcgis, 2004.

[2] Roger Bivand and Nicholas Lewin-Koh. *maptools: Tools for reading and handling spatial objects*, 2014. URL `http://CRAN.R-project.org/package=maptools`. R package version 0.8-30.

[3] Roger Bivand, Tim Keitt, and Barry Rowlingson. *rgdal: Bindings for the Geospatial Data Abstraction Library*, 2014. URL http://CRAN.R-project.org/package=rgdal. R package version 0.9-1.

[4] Roger S Bivand, Edzer J Pebesma, Virgilio Gómez-Rubio, and Edzer Jan Pebesma. *Applied spatial data analysis with R*, volume 747248717. Springer, 2008.

[5] Gerald Evenden. Proj. 4–cartographic projections library. *Source code and documentation available from trac. osgeo. org/proj*, 1990.

[6] Carl Franklin and Paula Hane. An introduction to geographic information systems: Linking maps to databases [and] maps for the rest of us: Affordable and fun. *Database*, 15(2):12–15, 1992.

[7] Carlo Gaetan, Xavier Guyon, and Kevin Bleakley. *Spatial statistics and modeling*, volume 271. Springer, 2010.

[8] E Grafarend. The optimal universal transverse mercator projection. In *Geodetic Theory Today*, pages 51–51. Springer, 1995.

[9] John W Hager, James F Behensky, and Brad W Drew. The universal grids: Universal transverse mercator (utm) and universal polar stereographic (ups). edition 1. Technical report, DTIC Document, 1989.

[10] Mordechai Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *Pervasive Computing, IEEE*, 7(4):12–18, 2008.

[11] Yun Feng Nie, Hu Xu, and Hai Ling Liu. The design and implementation of tile map service. *Advanced Materials Research*, 159:714–719, 2011.

[12] DT QGis. Quantum gis geographic information system. *Open Source Geospatial Foundation Project*, 2011.

[13] ESRI Software. *ESRI Shapefile Technical Description*, 1998. URL https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf.

[14] Simon Urbanek. *snippets: Code snippets, mostly visualization-related*. URL http://www.rforge.net/snippets/. R package version 0.1-0.