12. Spatial Data

Self-Paced Workshop #12

Prof. Jack Reilly

Fall 2026

Download the scripts: dwv_w12_1_bosnia.R · dwv_w12_2_dmas.R

GIS is not some special, unique kind of data that can only be accessed with specialized proprietary software. It is a set of files, organized in a certain way, and like any data, you can access, manipulate, wrangle, and visualize it in R. It lives in files, on your drive or across the network, and we still do what we have done all semester — manipulate cells of numbers that stand for things in the world, then show them visually. Geography brings its own constraints and complications (and opportunities!), but nothing about it is magic.

Part 1. Points on a Shapefile: The Bosnian War

The data we'll use for this workshop come from the assigned reference reading, Weidmann's *Data Management for Social Scientists*, chapter 11, posted with the book's companion files.

- `ged.csv` lists violent events from the UCDP Georeferenced Event Dataset (Sundberg and Melander 2013), with `best` as the best estimate of deaths.
- `bosnia.shp` holds the country's prewar municipalities (Weidmann 2011).

The boxes share one R session; run them top to bottom.

Step 1. Get the data

A shapefile is really several files (`.shp`, `.shx`, `.dbf`) that must sit together. This box downloads all of them into a `bosnia` folder, if it isn't there already. The downloadable script does the same in its working folder, `~/desktop/working/dwv26`, so it runs as-is in your own R. You can also copy the `bosnia` folder there from the Course Drive.

```
library(dplyr)
library(sf)

if (!file.exists("bosnia/bosnia.shp")) {
  dir.create("bosnia", showWarnings = FALSE)
  dm <- "https://raw.githubusercontent.com/nilsbw/dmbook-setup/main/ch11/"
  for (f in c("ged.csv", "bosnia.shp", "bosnia.shx", "bosnia.dbf"))
```

```
    download.file(paste0(dm, f), file.path("bosnia", f), mode = "wb")
}
list.files("bosnia")
```

Step 2. Points

As discussed in lecture, most geographic mapping — at least for economic, political and social data — comes down to three things: points, lines and polygons. This is *vector* data. (Its counterpart, *raster* data, describes something continuous across an area, like elevation; we will meet it with the terrain maps.) Here we load the event data, inspect it, and plot its basic geographic points: the sites of conflict.

First, load the event data and look at it. Each row is one event, with a start date, a location (`latitude` and `longitude`) and a death count (`best`):

```
events <- read.csv("bosnia/ged.csv")
summary(events)
```

For now, this is still just a regular data frame. To make R treat it as geography, we have to declare its spatial element: which columns hold the coordinates (longitude first, since it is the x), and which coordinate reference system they are in. `crs = 4326` is WGS 84, the ordinary latitude and longitude your phone's GPS uses.

```
events <- st_as_sf(events, coords = c("longitude", "latitude"), crs = 4326)
events
```

`events` is now a *simple feature collection*. The two coordinate columns have become one `geometry` column of `POINT`s, and R reports the bounding box and the CRS.

> **What is sf?**
>
> `sf` stands for *simple features*, an international standard for storing points, lines and polygons. The package keeps geographic data in an ordinary data frame: each row is one feature (an event, a municipality), the attributes are regular columns, and the shape sits in one extra `geometry` column. So the tidyverse tools you already know, like `filter()`, `mutate()` and `group_by()`, work on maps too, and `ggplot2` draws them with `geom_sf()`. `sf` also reads and writes nearly every geographic file format, and it handles coordinate systems and spatial operations like joins. It replaced the older `sp` package, so you'll see `sp` in older tutorials.

Now plot it. `plot(events)` draws one small map per column, colored by that column's values; wrapping the data in `st_geometry()` plots just the points:

```
plot(events)
plot(st_geometry(events))
```

Step 3. Polygons

Points don't mean much without some reference to look at them against. So here we'll load the shapefile, which gives us borders: the polygons of Bosnia's municipalities. Think of a

shapefile as supplying the *shape* of the units we care about (here municipalities; elsewhere cities, states or counties). We then merge attributes onto those shapes to visualize them.

```
bosnia <- st_read("bosnia/bosnia.shp", crs = 4326)
bosnia
```

Then plot it:

```
plot(bosnia)
```

Gibberish! As with the events, `plot()` draws one map per column (`id` and `name`) and shades each municipality by values that mean nothing on a map. Let's be more specific: just the outlines, in light grey, with the events added on top (`add = TRUE` draws over the existing plot instead of starting a new one, and `pch = 16` makes each point a solid dot):

```
plot(st_geometry(bosnia), col = "lightgrey")
plot(st_geometry(events), pch = 16, add = TRUE)
```

Wow! That was . . . easy? Not so fast. The last 10% of a graph (especially a map) takes 90% of the effort. We still want to color the events by severity, and to add a title and a legend. Here's a first try at the color:

```
plot(st_geometry(bosnia), col = "lightgrey")
plot(st_geometry(events), col = events$best, pch = 16, add = TRUE)
```

Drat. *Definitely not 90% of the way there after all.*

Step 4. A color scale

A funny thing happened in Step 3: we never told R what kind of color we wanted, or what the colors should mean. So at first it colored the polygons effectively at random, and then, once we gave the polygons a grey and plotted points over them with `col = events$best`, it gave the points random colors as well.[1]

Here, we'll fix that. Let's make a color scale to color conflicts (our points) by severity. `gradient_n_pal()` from the `scales` package builds a palette function that runs from black to red. `rescale()` then turns each death count into a number between 0 and 1, measured across the data's range (`rng`, its smallest and largest values), and `black_red()` turns that number into a color: 0 is black, 1 is red. Finally, we plot with those colors:

```
library(scales)
black_red <- gradient_n_pal(c("black", "red"))

vals <- events$best
rng  <- range(vals, na.rm = TRUE)
z    <- rescale(vals, to = c(0, 1), from = rng)
ptcol <- black_red(z)
```

[1] Not quite random. Given plain numbers for `col`, R treats each one as a position in its eight-color default palette (`palette()`), starting over after eight. So an event with 9 deaths gets the same color as an event with 1, and an event with 0 deaths is drawn in color 0, the background, and vanishes.

```
plot(st_geometry(bosnia), col = "lightgrey")
plot(st_geometry(events), col = ptcol, pch = 16, add = TRUE)
```

This color looks . . . peculiar. Only one thing appears to be colored! Why? We probably have at least one severe outlier. Let's investigate with a histogram:

```
library(lattice)
histogram(events$best)
```

This is what a histogram looks like when there are a great many events at the bottom and a few (in this case, very terrible) outliers. To find the biggest events, sort the data by `best` and look at the last ten, which are the ten highest:

```
events_sorted <- events[order(events$best), ]
tail(events_sorted$best, 10)
```

We have an 8,106, a 3,000, a few between 100 and 300, and then everything else. Because the scale runs from 0 to 8,106, almost every event sits near 0 and comes out black.

Step 5. Splitting out the outliers

It turns out that we had some significant outliers in our plot. New strategy: split the events into separate *channels* (normal, large, and outlier), and use two different scales — both the color and the shape of the point itself — to indicate severity.

First, we'll flag the groups, so you can check which events fall where:

```
events$outlier <- events$best %in% c(3000, 8106)
events$large <- events$best %in% c(144, 150, 197, 270, 300)
```

Now, we'll break our point dataset in three: "normal", "large" and "outlier". `ev_reg`, the normal events, keeps everything from 0 to 101 deaths. Then we rebuild the color scale over the normal events alone, so the outliers no longer stretch it:

```
large_vals    <- c(144, 150, 197, 270, 300)
outlier_vals  <- c(3000, 8106)

ev_out   <- events[!is.na(events$best) & events$best %in% outlier_vals, ]
ev_large <- events[!is.na(events$best) & events$best %in% large_vals, ]
ev_reg   <- events[!is.na(events$best) &
                     !(events$best %in% c(large_vals, outlier_vals)) &
                     events$best >= 0 & events$best <= 101, ]

vals <- ev_reg$best
rng  <- range(vals, na.rm = TRUE)
z    <- rescale(vals, to = c(0, 1), from = rng)
ptcol <- black_red(z)
```

```
plot(st_geometry(bosnia), col = "lightgrey")
plot(st_geometry(ev_reg), col = ptcol, pch = 16, add = TRUE)
```

Better? Still dark. Two ways to push the scale toward red: center it on a typical value, or bend it with an exponent.

First, centering. What if we made the middle of the color scale the mean of the variable? Check the mean and median, then use `rescale_mid()` to put the scale's midpoint at 7, close to the mean:

```
mean(ev_reg$best)
median(ev_reg$best)

z <- rescale_mid(vals, to = c(0, 1), from = rng, mid = 7)
ptcol <- black_red(z)

plot(st_geometry(bosnia), col = "lightgrey")
plot(st_geometry(ev_reg), col = ptcol, pch = 16, add = TRUE)
```

Second, bending. Rescale as before, then raise each value to the power 0.7. Every value is between 0 and 1, so a power below 1 pushes it *up*, and the colors shift toward red earlier:

```
z <- rescale(vals, to = c(0, 1), from = rng)
z <- z ^ 0.7
ptcol <- black_red(z)

plot(st_geometry(bosnia), col = "lightgrey")
plot(st_geometry(ev_reg), col = ptcol, pch = 16, add = TRUE)
```

Even with the large events and the outliers taken out, this is still a tough distribution to color effectively. The histogram shows why:

```
histogram(ev_reg$best)
```

It's still just a very skewed dataset. We could keep playing with the color through transformations and midpoints, but it's a tough challenge. Instead, let's add our second and third data layers.

Step 6. Putting it together

Now we add the others. We draw the base map with a title (`main`), then the normal events with the color scale. The large events go on top as bigger maroon squares (`pch = 15`; `cex = 2` doubles their size), and the two outliers as the biggest triangles (`pch = 17`, `cex = 3`), one maroon and one red. Last comes a legend in the bottom-left corner; `bty = "o"` draws a box around it.

```
plot(st_geometry(bosnia), col = "lightgrey", main="Bosnian Civil War")
plot(st_geometry(ev_reg), col = ptcol, pch = 16, add = TRUE)
plot(st_geometry(ev_large), col = "maroon", pch = 15, cex=2, add = TRUE)
```

```
plot(st_geometry(ev_out[1,]), col = "maroon", pch = 17, cex=3, add = TRUE)
plot(st_geometry(ev_out[2,]), col = "red", pch = 17, cex=3, add = TRUE)

legend("bottomleft",
       legend = c("Small Event (0-101 casualties)",
                  "Large Event (102-300)",
                  "Major Event (3000)",
                  "Major Event (8106)"),
       pch    = c(16, 15, 17, 17),
       pt.cex = c(0.9, 1.3, 1.8, 2.2),
       col    = c("red", "#800000", "#800000", "red"),
       bty    = "o")
```

So what's familiar here? Most things! We are layering, we are playing with colors (exhaustively), and we are making choices about how to represent the data. The only real differences are the work we did on the data up front, and that one of our data sources is a geographic shapefile.

Step 7. A spatial join

We can also merge the two datasets into a single object. `st_join()` matches each event to the municipality it falls inside, which lets us calculate statistics more easily with the tidyverse. Here, we count events by municipality. Both files have a column called `id`, so the joined data calls the event's `id.x` and the municipality's `id.y`. `as.data.frame()` turns the result back into an ordinary data frame before we group and count:

```
joined <- st_join(events, bosnia)

eventcounts <- joined |>
  as.data.frame() |>
  group_by(id.y) |>
  count(name = "num_events")

eventcounts
```

Part 2. Other Shapefiles

Finding geographic data is where you will spend much of your time. Valuable sources include the US Census Bureau (data and shapefiles), OpenStreetMap (roads and some geographic features), and the shapefiles you may find floating around the web.

Additional scripts from class cover the Census, OpenStreetMap, and a shapefile of TV markets (DMAs) included on the Course Drive. They are all on the Week 12 in-class workshop page, with downloads and previews of each script.

DMA files on the Course Drive