Chapter 12: Data in space and in text

Thus far we've abstracted away from thinking about how people/places/things in our dataset relate to eachother. Relationships can be complicated. People relate through roles in families, statuses social networks, positions at work, or simple locations in a city/neighborhood. Public policies, people, and businesses overlap in all kinds of interesting and complex relationships. For example, after marijuana legalization, states have restricted (1) who can purchase marijuana, (2) who can sell marijuana, and (3) how close these dispensaries can be to certain buildings (especially schools). This simple example highlights the complexity in thinking about how policies interact with the people and businesses they govern—and the space these people, businesses, and policies inhabit.

In reality, data do not live in static, boring spreadsheets where each row can be treated as an independent observation, isolated from relationships with any other observation. Consequently, as data science evolves, data scientists are developing methods to better harness, represent, and understand the complex relationships inherent to (and underlying) public policy. Two particularly important and common ways in which data science employs more specialized data are (1) analyzing spatial data and (2) natural-language processing of text-based data. It is difficult to imagine public policies without also thinking about the places/relationships affected by the police and the text/language that communicate the police (in official legal texts and in less official, social-media texts).

Spatial-data analysis

The first things many people think of when they think of *spatial data* are maps. Maps convey *spatial relationships* between people, policies, places, and other objects. As such, well-made (and aesthetically pleasing) maps can be extradinarily helpful in understanding and communicating policy. However, maps are only tool in the toolbox of spatial-data analysis.

Consider five spatially intensive examples of problems in public policy:

- Find the ten neighborhoods in a city with the longest response path from police and fire departments.
- Use real-time satellite imagery to detect smoke and generate early warnings of forest fires.
- Detect changes in high-resolution satellite imagery that suggest building additions that do not match any building permits.
- Estimate the impact of bar openings on nearby crime, business patterns, and property values.
- Determine whether low-income households are particularly prone and risks from floods and sea-level-rise.

Spatial data, defined

What makes something *spatial data*? In general, the answer is that your dataset includes attributes that allow you to relate the observations/objects in space. Often, these attributes are two variables that place the observations on a grid, for example, latitude on longitude (though any x and y will work). Sometimes you'll have additional information, for example the elevation/altitude of the observation or the ways in which the observations are connected (in space or otherwise)—but let's start with the two-dimension case.

A typical (2-D) spatial dataset will thus have some identifier for an individual (ID, name, etc.), some interesting features, and the aforementioned coordinates. For example, here are cities in the United States (a dataset contained in the `maps` package)

```r
# Example dataset: US cities
maps::us.cities %>% as_tibble()
```

```
## # A tibble: 1,005 x 6
##    name           country.etc    pop   lat   long capital
##    <chr>          <chr>        <int> <dbl>  <dbl>   <int>
##  1 Abilene TX     TX          113888  32.4  -99.7       0
```

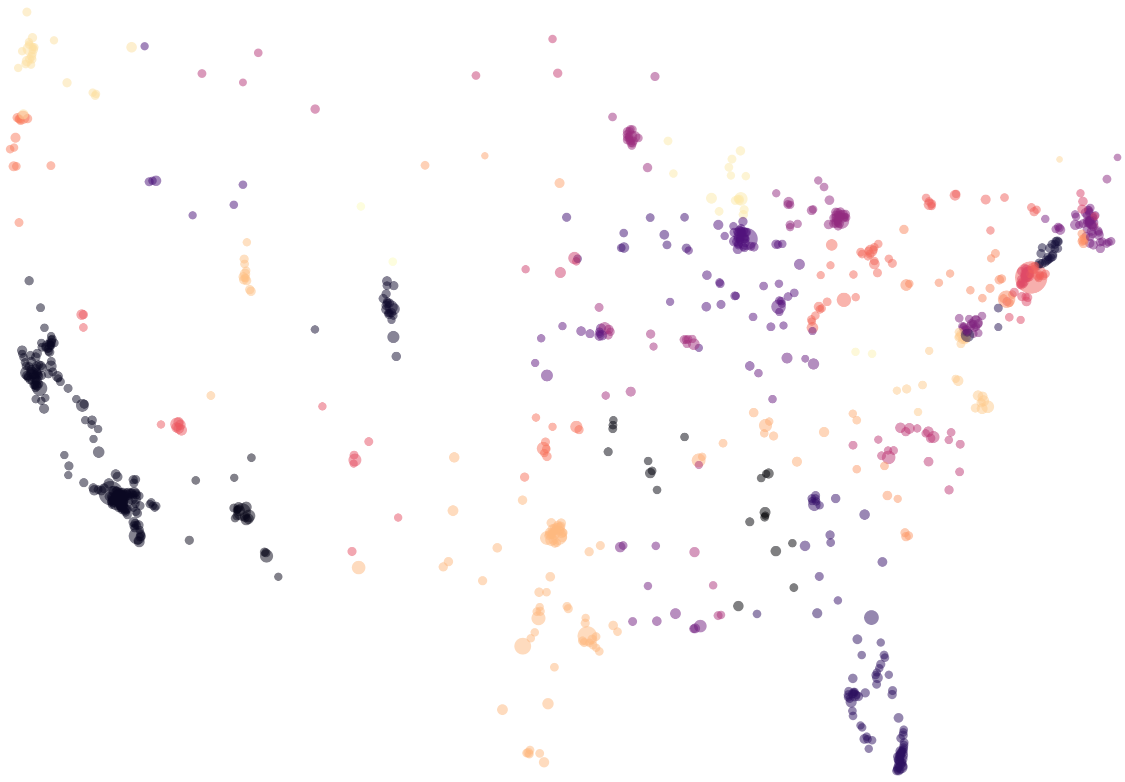


Figure 1: Spatial data example: U.S. cities plotted, colored by state, sized by population

```
##  2 Akron OH       OH          206634  41.1  -81.5       0
##  3 Alameda CA     CA           70069  37.8 -122.        0
##  4 Albany GA      GA           75510  31.6  -84.2       0
##  5 Albany NY      NY           93576  42.7  -73.8       2
##  6 Albany OR      OR           45535  44.6 -123.        0
##  7 Albuquerque NM NM          494962  35.1 -107.        0
##  8 Alexandria LA  LA           44933  31.3  -92.5       0
##  9 Alexandria VA  VA          127159  38.8  -77.1       0
## 10 Alhambra CA    CA           88857  34.1 -118.        0
## # ... with 995 more rows
```

which we can plot to achieve a (rough) map

```
ggplot(
  data = maps::us.cities %>% filter(country.etc %>% is_in(c("AK", "HI")) %>% not()),
  aes(x = long, y = lat, size = pop, color = country.etc)
) +
geom_point(alpha = 0.5) +
scale_color_viridis_d(option = "magma") +
scale_size(range = c(0.5, 5)) +
theme_map() +
theme(legend.position = "none")
```

Two classes of spatial data

Variables in spatial datasets broadly fit into two categories: *fields* and *objects*.

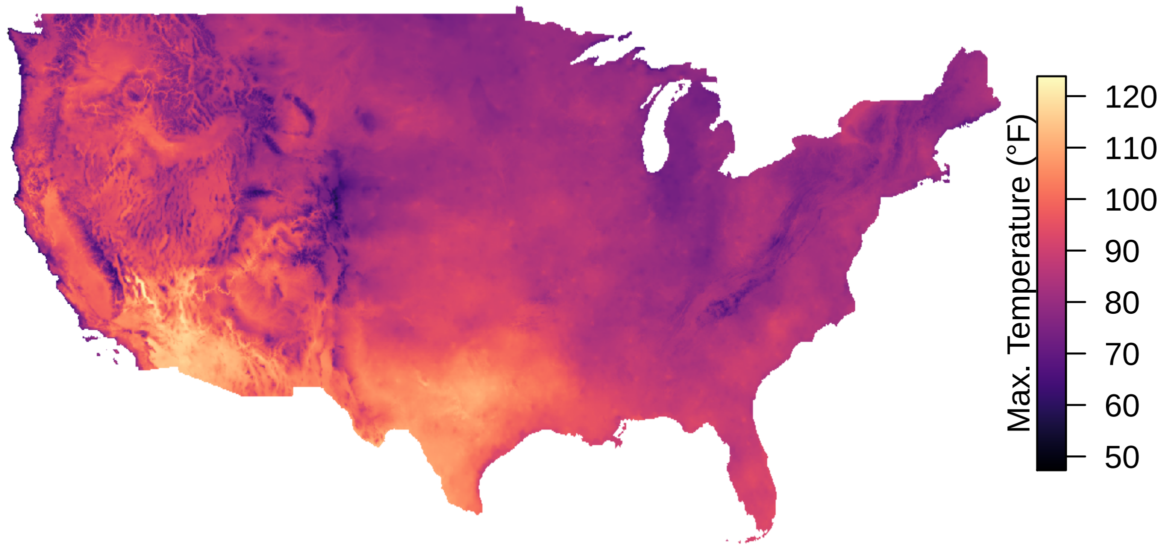


Figure 2: Example of a field: Maximum daily temperature in the United States on 24 July 2018

Fields represent spatially continuous concepts like temperature, humidity, air quality, or soil quality—concepts that take on values everywhere throughout space.

TODO Image example for fields (temperature?)

Objects are variables that are discrete, disconnected items like cities, building footprints, property lines, rivers, or road networks. Objects

TODO Image example for objects (roads?)

Because continuous fields and discrete objects are conceptually so different, we use different classes of spatial data to analyze them. To quantify continuous fields, we use *rasters*, whereas we use *vectors* to portray discrete fields.

Rasters

Rasters are grids. Each grid cell of a raster is associated with a value that summarizes the portion of the field contained in that grid (raster) cell. You can think of rasters as discretized or pixelated versions of the continuous field. Rasters are defined by their **extent***—the swath of space that they cover—and by their **resolution**—the size of the grid cells. If your raster's cells are quite small, then you have a* high-resolution* raster, which should do a good job of characterizing nearly any field. The drawback: such a raster may be relatively large in size and take more time to work with. Low-resolution rasters (large raster cells) can do a decent job of approximating fields whose values do not change much at smaller scales. However, as the figure below illustrates, low-resolution rasters *can* miss potentially important variation when that variation occurs at a very fine/local scale. Depending upon your application, missing out on the fine, hyper-local variation may not matter to your project—or it could wreck the entire project.

As defined above, rasters are just grids (often used to represent spatially continuous fields. So rasters really boil down to **coordinates** (generally in two dimensions) plus a **value** attached to the coordinates. The value denotes the raster's respresentation of the field within the grid cell centered at the coordinates.

For example, consider a very simple raster with 9 total cells. We can write it down as a data frame with 9 rows and 3 columns—two columns for coordinates and one column for the value.

```
simple_df = data.frame(
  x = c(1, 2, 3, 1, 2, 3, 1, 2, 3),
```

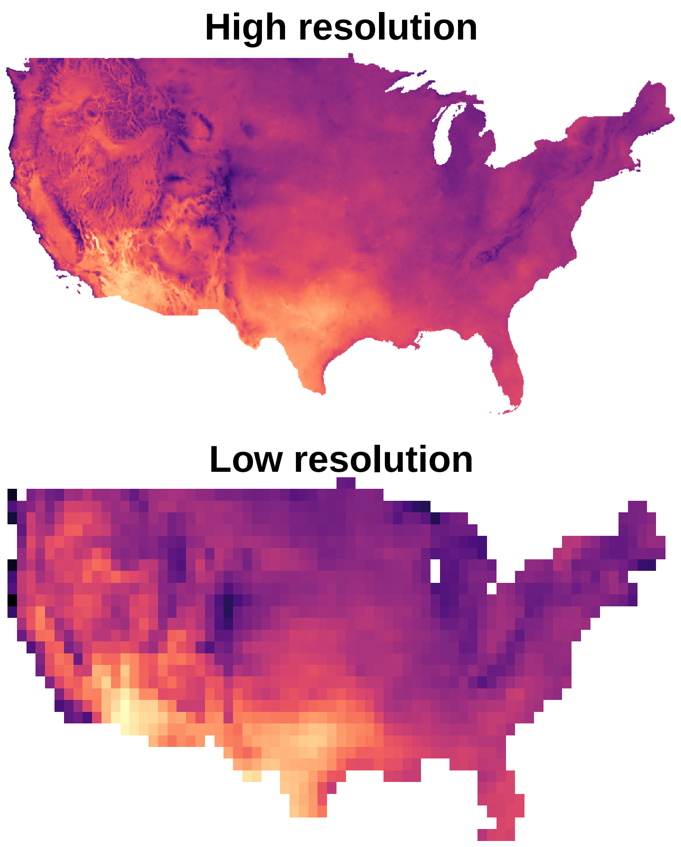


Figure 3: Comparing high- and low-resolution rasters. The raster on the right is approximately 1km resolution; the raster on the left is approximately 200km resolution.

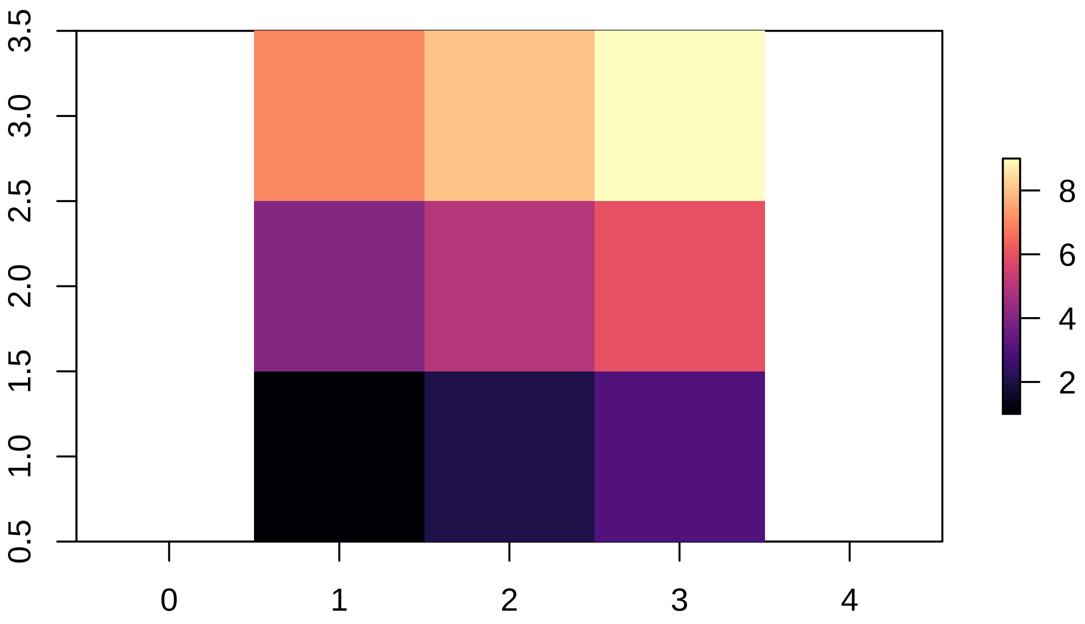
```
  y = c(1, 1, 1, 2, 2, 2, 3, 3, 3),
  value = 1:9
)
simple_df
```

```
##   x y value
## 1 1 1     1
## 2 2 1     2
## 3 3 1     3
## 4 1 2     4
## 5 2 2     5
## 6 3 2     6
## 7 1 3     7
## 8 2 3     8
## 9 3 3     9
```

```
# Convert from data frame to raster
simple_raster = simple_df %>% rasterFromXYZ()
simple_raster
```

```
## class      : RasterLayer
## dimensions : 3, 3, 9  (nrow, ncol, ncell)
## resolution : 1, 1  (x, y)
## extent     : 0.5, 3.5, 0.5, 3.5  (xmin, xmax, ymin, ymax)
## crs        : NA
## source     : memory
## names      : value
## values     : 1, 9  (min, max)
```

```
plot(
  simple_raster,
  col = magma(1e3),
  box = F
)
```



0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
1
2
3
4
2
4
6
8