Chapter 17: Working with Geospatial Data

Qadeem Qureshi, Brandon Whiteley, Mac McDaniel, Rebecca Lescay

2022-04-19

Background

Geospatial data is a collection of points, most commonly geographic coordinates, that can provide context to complex relationships. These relationships often provide us with the ability to understand how variables can relate to geographic location. The difference between geospatial data and a vector of numbers is the meaning associated with locations and their relation to some other variables. As it is, a collection of coordinates in some larger database provides little to no meaning and can be misrepresented in regression-based analysis. On the other hand, shapefiles in R and other applications provide a rich and meaningful way to determine geospatial relationships between location and other variables. Visualizations of shapefiles require knowledge of projection - the idea of transforming three-dimensional coordinate systems into corresponding two-dimensional systems while minimizing loss of positioning.

Motivation

Baumer begins by detailing the idea of cholera, its spreadability, and its relation to water pumps in nearby vicinities. This was originally done by physician John Snow who determined cholera and its spreadability through water and not air. The way through which this was determined was by mapping an area in London and then also projecting on it the deaths and locations of water pumps. Interestingly, people nearest to water pumps had the highest rate of death. Similarly, Baumer explores the relationship between voting Republican or Democrat in North Carolina and how it showcases the effect of gerrymandering. Geospatial data can be used to analyze trends, as well as relationships between location and other variables that are otherwise unrelated.

Structure of Spatial Data

Spatial data is most commonly presented in the shapefile format or the KML format. These files are composed of shapes such as points, lines, and polygons. Unlike data frames, they are rich in data and allow for drawing boundary objects as well as providing a schema as to how to draw them.

Shapefiles are not a single container of data. Instead, they can include files with extensions `.shp`, `.shx`, and `.dbf`, commonly stored in the same directory. While there is a multitude of R packages that can make use of shapefiles, for this purpose we use a tidyverse-friendly `sf` package.

SF Package Overview

Begin by including the package into your R editor. This can be done by the following:

```
library(sf)
```

```
## Linking to GEOS 3.9.1, GDAL 3.2.1, PROJ 7.2.1; sf_use_s2() is TRUE
```

Next, download the SnowGIS shapefiles. You can unzip this directory and notice that there are several files associated with the Pumps and Cholera data. These are referred to layers which R's `SF` package will combine internally to represent a single shapefile. To view the layers and their corresponding features, utilze the following:

```
dsn <- fs::path("SnowGIS_SHP")
st_layers(dsn)
```

```
## Driver: ESRI Shapefile
## Available layers:
##       layer_name geometry_type features fields
## 1 Cholera_Deaths         Point      250      2
## 2          Pumps         Point        8      1
```

To load a layer into a R data.frame, you can do the following:

```
cholera_deaths_df <- st_read(dsn, layer="Cholera_Deaths")
```

```
cholera_deaths_df
```

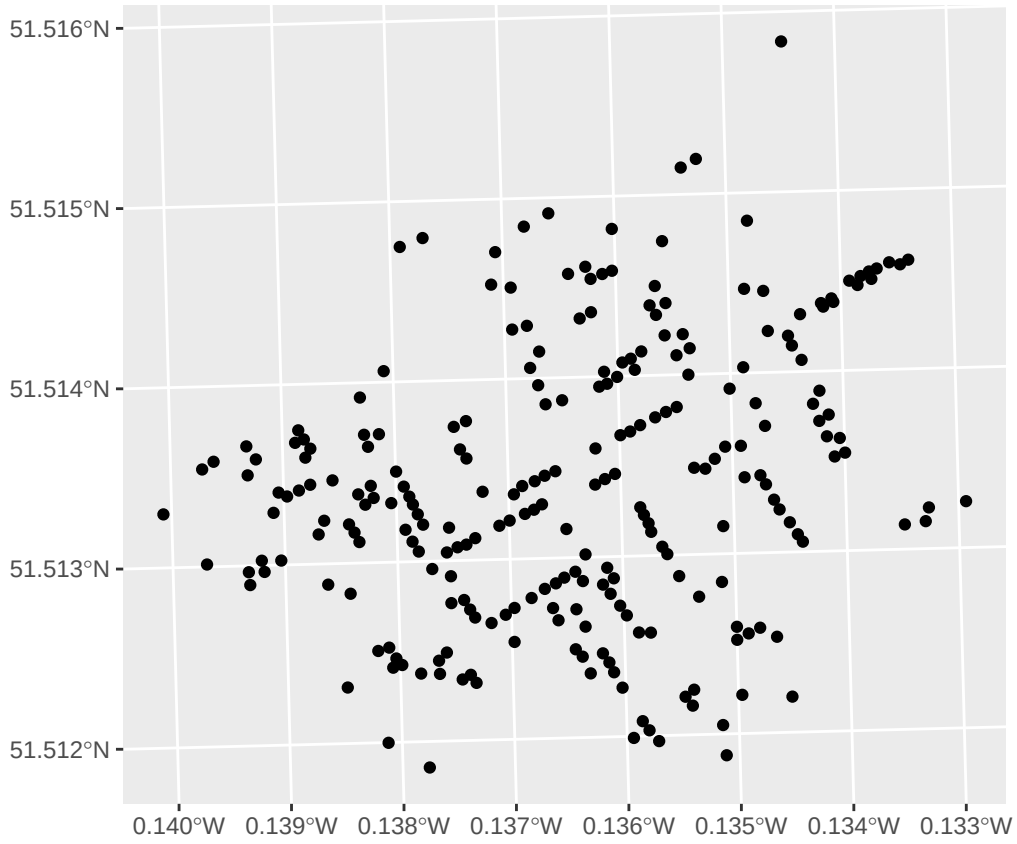
```
## Simple feature collection with 250 features and 2 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 529160.3 ymin: 180857.9 xmax: 529655.9 ymax: 181306.2
## Projected CRS: OSGB 1936 / British National Grid
## First 10 features:
##    Id Count                  geometry
## 1   0     3 POINT (529308.7 181031.4)
## 2   0     2 POINT (529312.2 181025.2)
## 3   0     1 POINT (529314.4 181020.3)
## 4   0     1 POINT (529317.4 181014.3)
## 5   0     4 POINT (529320.7 181007.9)
## 6   0     2   POINT (529336.7 181006)
## 7   0     2 POINT (529290.1 181024.4)
## 8   0     2   POINT (529301 181021.2)
## 9   0     3   POINT (529285 181020.2)
## 10  0     2 POINT (529288.4 181031.8)
```

This data frame is composed of the `Id`, `Count`, and `Geometry` which represent the location, death count, and the location respectively.

Plotting Shapefiles in R

To plot a data frame, we can utilize the package `ggplot2`. In ggplot, there is a geom associated with mapping shapefile data frames. This is known as: `geom_sf`. A simple map of the Cholera Deaths without context is represented by:

```
library(ggplot2)

ggplot(cholera_deaths_df) + geom_sf()
```



Again, this is a meaningless map given that there is no underlying image to base this data on. To associate an image with this plot, we can leverage the OpenStreeMap (OSM) map within the shapefile folder downloaded earlier. The `ggspatial` package has the command `annotation_map_tile` that allows for loading OSM tiles of a location. For this example, we can do:

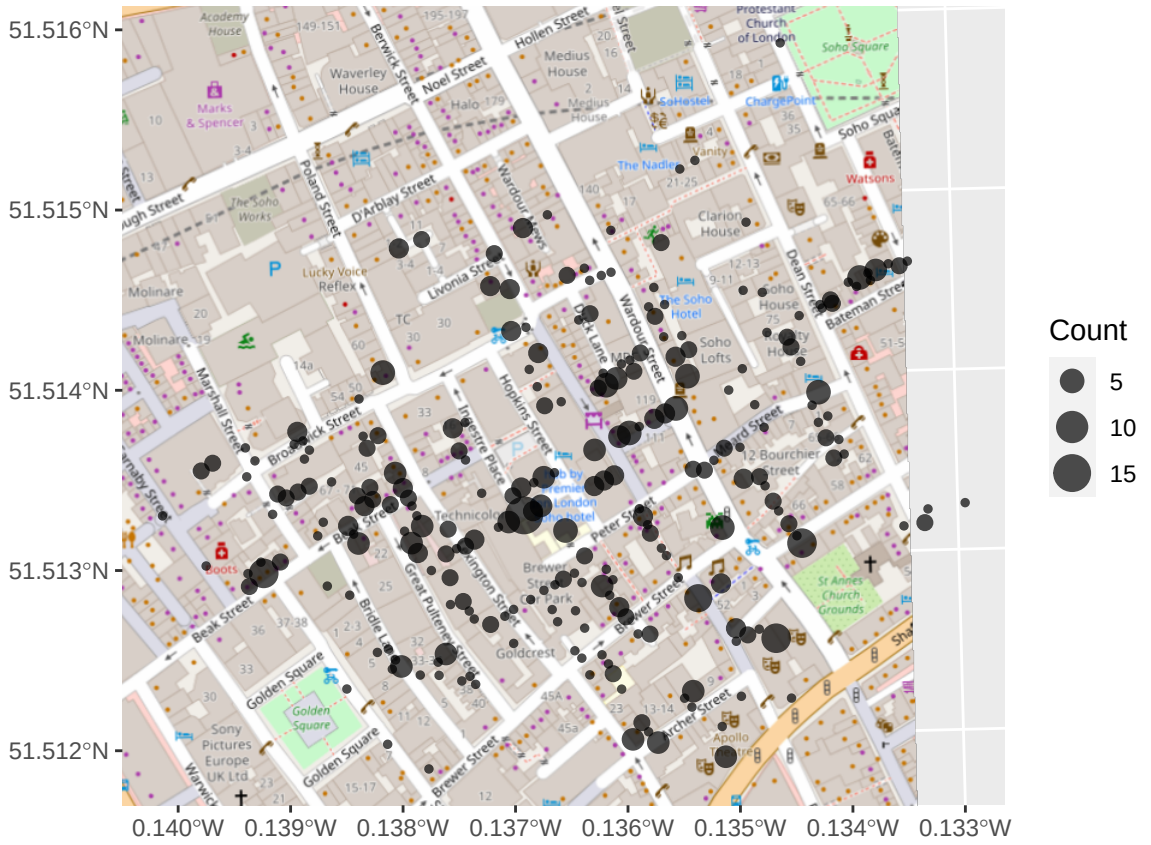
```
library(ggspatial)
```

```
## Warning: package 'ggspatial' was built under R version 4.1.3
```

```
ggplot(cholera_deaths_df) +
  annotation_map_tile(type = "osm", zoomin = 0) +
  geom_sf(aes(size = Count), alpha = 0.7)
```

```
## Loading required namespace: raster
```

```
## Zoom: 17
```



This is a lot better than the plot earlier, however, it does not transform the coordinates correctly. This is seen with the gray area on the right side of the plot as well as the offset location of the points. To view the system that the `cholera_deaths_df` uses, we can use the `st_crs` command.

```
st_crs(cholera_deaths_df)
```

```
## Coordinate Reference System:
##   User input: OSGB 1936 / British National Grid
##   wkt:
## PROJCRS["OSGB 1936 / British National Grid",
##     BASEGEOGCRS["OSGB 1936",
##         DATUM["OSGB 1936",
##             ELLIPSOID["Airy 1830",6377563.396,299.3249646,
##                 LENGTHUNIT["metre",1]]],
##         PRIMEM["Greenwich",0,
##             ANGLEUNIT["degree",0.0174532925199433]],
##         ID["EPSG",4277]],
##     CONVERSION["British National Grid",
##         METHOD["Transverse Mercator",
##             ID["EPSG",9807]],
##         PARAMETER["Latitude of natural origin",49,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8801]],
##         PARAMETER["Longitude of natural origin",-2,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8802]],
```

```
##         PARAMETER["Scale factor at natural origin",0.9996012717,
##             SCALEUNIT["unity",1],
##             ID["EPSG",8805]],
##         PARAMETER["False easting",400000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8806]],
##         PARAMETER["False northing",-100000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8807]]],
##     CS[Cartesian,2],
##         AXIS["(E)",east,
##             ORDER[1],
##             LENGTHUNIT["metre",1]],
##         AXIS["(N)",north,
##             ORDER[2],
##             LENGTHUNIT["metre",1]],
##     USAGE[
##         SCOPE["Engineering survey, topographic mapping."],
##         AREA["United Kingdom (UK) - offshore to boundary of UKCS within 49Â°45'N to 61Â°N and 9Â°W t
##         BBOX[49.75,-9,61.01,2.01]],
##     ID["EPSG",27700]]
```

At the very end of this, you can see the data is in the format of `EPSG:27700`. OpenStreeMap, on the other hand, has coordinates in `ESPG:4326`. To convert from one system to another, we can use the `st_transform` command. This then produces:

```
cholera_deaths_df_4326 <- cholera_deaths_df %>% st_transform(4326)

chloera_deaths_plot <- ggplot(cholera_deaths_df_4326) +
  annotation_map_tile(type = "osm", zoomin = 0) +
  geom_sf(aes(size = Count), alpha = 0.7)
chloera_deaths_plot
```

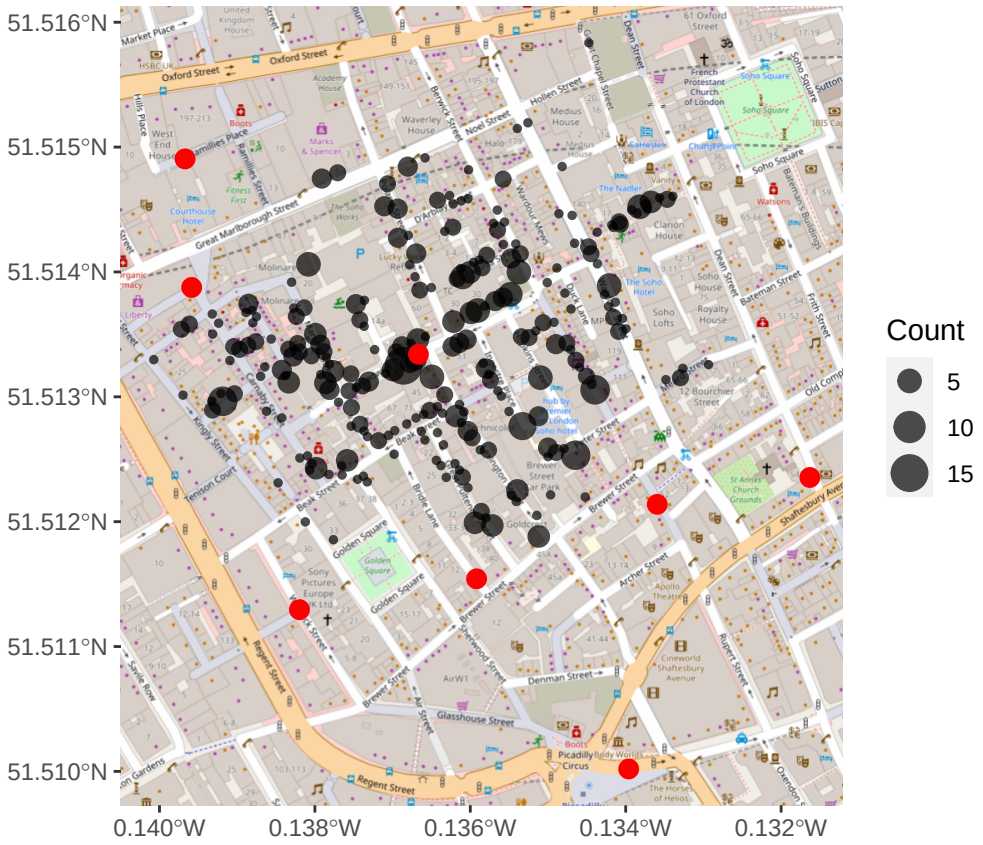
```
## Zoom: 17
```

Now that this is in the correct format, we can add the Pumps associated with the additional layer. This can be done through the code above used to load the Cholera Deaths. The code to load and add the pumps to the map is the following:

```
pumps_deaths_df <- st_read(dsn, layer="Pumps") %>% st_transform(4326)
```

```
chloera_deaths_plot + geom_sf(data = pumps_deaths_df, size = 3, color = "red")
```

```
## Zoom: 17
```



As a result of this, the areas around the pump seemingly had the higest rate of Cholera deaths.

Extended usage

In the exercises portion of the Baumer chapter 17 presentation, we can see the question:

`Problem 4 (Hard): Use the tidycensus package to conduct a spatial analysis of the Census data it contains for your home state. Can you illustrate how the demography of your state varies spatially?`

To assess this, we must examine the `tidycensus` package. This package allows us to interact with the Census API provided by the US government. However, in order to do so, we will need to obtain a key. You can either use my key or sign up for your own Here.

The objective is to (1) gather all data from the Census related to Oklahoma, (2) Use race as a means by which we can distinguish the population, (3) plot this data. Interacting with the Census API requires knowledge of the variables available. Luckily, the `tidycensus` package allows us to view these variables. We can start using this package by:

```
library(tidycensus)
```

```
## Warning: package 'tidycensus' was built under R version 4.1.3
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v tibble  3.1.6     v dplyr   1.0.8
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
## v purrr   0.3.4

## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
census_api_key("65e8134806aaa15b6b8244ff0f74941a07f7a741", install = TRUE, overwrite =  TRUE)
```

```
## Your original .Renviron will be backed up and stored in your R HOME directory if needed.

## Your API key has been stored in your .Renviron and can be accessed by Sys.getenv("CENSUS_API_KEY").
## To use now, restart R or run `readRenviron("~/.Renviron")`
```

```
variables <- load_variables(2020, "pl", cache = TRUE)

head(variables)
```

```
## # A tibble: 6 x 3
##   name    label                                                concept
##   <chr>   <chr>                                                <chr>
## 1 H1_001N " !!Total:"                                          OCCUPANCY STATUS
## 2 H1_002N " !!Total:!!Occupied"                                OCCUPANCY STATUS
## 3 H1_003N " !!Total:!!Vacant"                                  OCCUPANCY STATUS
## 4 P1_001N " !!Total:"                                          RACE
## 5 P1_002N " !!Total:!!Population of one race:"                 RACE
## 6 P1_003N " !!Total:!!Population of one race:!!White alone"    RACE
```

After exploring these variables, we need to locate the races we would like to view. For the purpose of this problem, we have chosen to only include the following: Asian, Black, Hispanic, and White. We can set this into a `racevars` which will contain the variables we want to plot.

```
racevars <- c(White = "P2_005N",
              Black = "P2_006N",
              Asian = "P2_008N",
              Hispanic = "P2_002N")
```

The `tidycensus` package allows for interaction with the decade based Census or the American Community Survey. For this, we want to use the 2010-2020 data from the Census. To load this into R, we do:

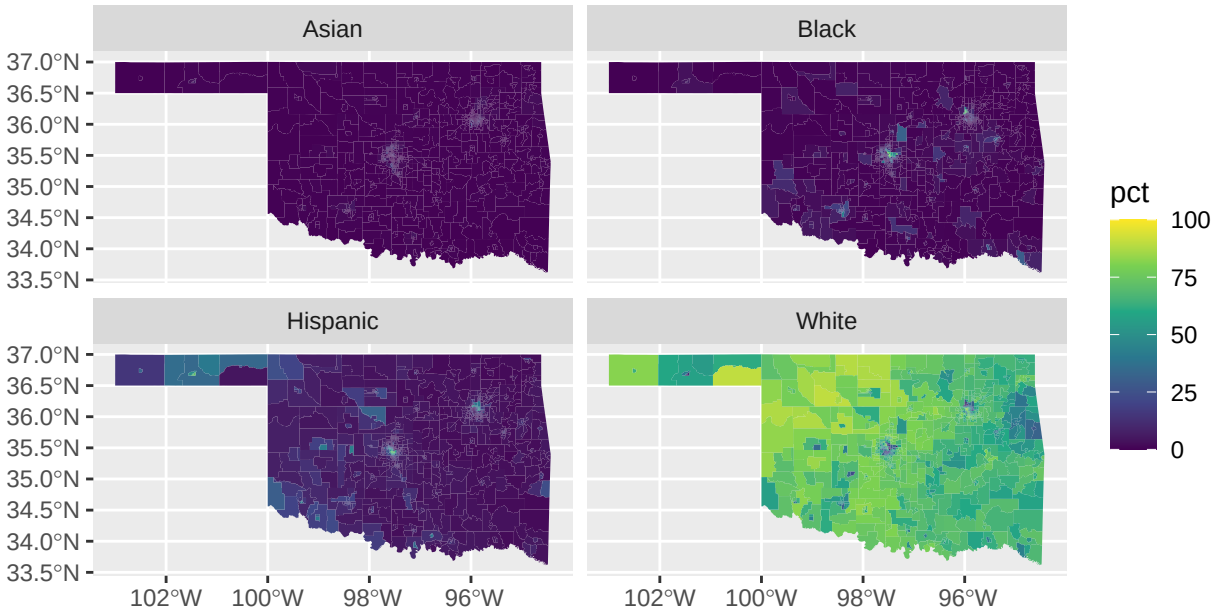
```
Oklahoma_df <- get_decennial(geography = "tract", variables = racevars,
                         state = "OK", geometry = TRUE,
                         summary_var = "P1_001N", year = 2020)
```

```
## Getting data from the 2020 decennial Census

## Downloading feature geometry from the Census website.  To cache shapefiles for use in future sessions

## Using the PL 94-171 Redistricting Data summary file
```

```
## Note: 2020 decennial Census data use differential privacy, a technique that
## introduces errors into data to preserve respondent confidentiality.
## i Small counts should be interpreted with caution.
## i See https://www.census.gov/library/fact-sheets/2021/protecting-the-confidentiality-of-the-2020-cens
## This message is displayed once per session.
```

The `Oklahoma_df` is a data frame containing spatial data along with population per county. To plot this data frame by race, we can facet by race. We can use ggplot2 to showcase this graph as:

```
Oklahoma_df %>%
  mutate(pct = 100 * (value / summary_value)) %>%
  ggplot(aes(fill = pct)) +
  facet_wrap(~variable) +
  geom_sf(color = NA) +
  scale_fill_viridis_c()
```



Summary

R is a versatile language with support packages that allow for analyzing geospatial data. As shown, rich spatial data can help assess the relationships between spatial location and other user defined variables. The implications of such data can be benficial to understanding the effects of political, cultural, and social factors. Ultimately, rich spatial data provides a means through which we can utilize location beyond numerical analysis.