2023 554 R Notes on Spatial Data Analysis

Jon Wakefield
Departments of Biostatistics and Statistics
University of Washington

2023-01-09

R for Spatial Analysis

`R` has extensive spatial capabilities, the Spatial task view is here

Some of the notes that follow are build on Roger Bivand's notes taken from the latter site, and these are based on Bivand et al. (2013), which is a good reference book of the spatial capabilities of R at that point.

See Spatial Data Science: With Applications in R by Pebesma and Bivand, for a more up to date version.

Another resource is the book Geocomputation with R

To get R code alone then load the `knitr` library and then type

```
purl("2023-554-Spatial-Classes.Rmd")
```

from the directory with this file in.

Overview of Spatial Classes

Class definitions are objects that contain the formal definition of a class of R objects, and are usually referred to as an S4 class.

Spatial classes were defined to represent and handle spatial data, so that data can be exchanged between different classes - they are different from regular classes since they need to contain information about spatial locations and their coordinate reference systems

- The `sp` library was traditionally the workhorse for representing spatial data.
- Now `sf` is being increasingly used, details are here

These notes will focus on `sf`.

The `sf` Representation

The `sf` is the successor package to R packages formerly used in spatial data including `sp`, `rgeos` and the vector parts of `rgdal`, providing an interface to certain tidyverse packages.

This package reads and writes data through GDAL, and uses GEOS, s2geometry, and PROJ.

The `sf` package has `sf` objects, a sub-class of a `data.frame` or a `tibble`. These objects contain at least one geometry list-column of class `sfc`, where each list element contains the geometry as an R object of class `sfg`.

All functions are prefixed with `st_` to indicate "spatial type" to make them more easily searchable on the command line.

You can load the package by running `library(sf)`, but note that if you've never used this package before, you should run `install.packages('sf')` in your R console (and then load the package with the `library(sf)` command).

Creating a Spatial Object

We can use the `st_read` function (part of the `sf` package) to read in a spatial file.

```
file <- system.file("gpkg/nc.gpkg", package = "sf")
nc <- st_read(file)
## Reading layer `nc.gpkg' from data source
##   `/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/sf/gpkg/nc.gpkg'
##   using driver `GPKG'
## Simple feature collection with 100 features and 14 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -84.32385 ymin: 33.88199 xmax: -75.45698 ymax: 36.58965
## Geodetic CRS:  NAD27
```

Note that the `st_read()` function takes in two arguments, the `dsn`, or data source name, and the layer. Our provided data has only one layer, so that argument is omitted in our example. You can check the available layers by querying:

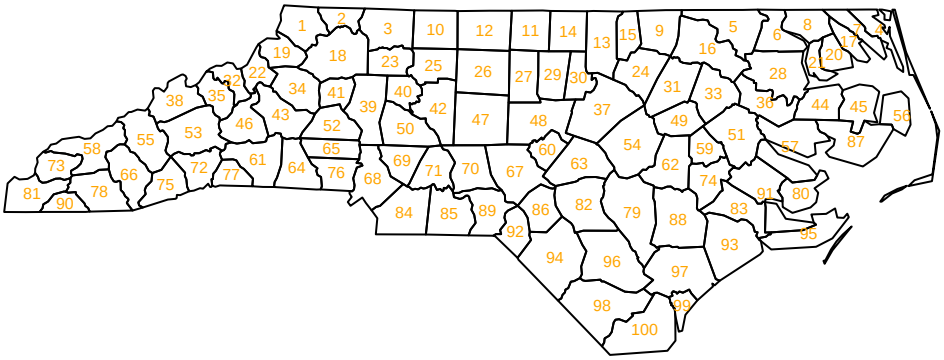
```
st_layers(file)
## Driver: GPKG
## Available layers:
##   layer_name geometry_type features fields crs_name
## 1    nc.gpkg Multi Polygon      100     14    NAD27
```

Let's look at the first few lines

```
head(nc, n = 3)
## Simple feature collection with 3 features and 14 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -81.74107 ymin: 36.23388 xmax: -80.43531 ymax: 36.58965
## Geodetic CRS:  NAD27
##    AREA PERIMETER CNTY_ CNTY_ID      NAME  FIPS FIPSNO CRESS_ID BIR74 SID74
## 1 0.114     1.442  1825    1825      Ashe 37009  37009        5  1091     1
## 2 0.061     1.231  1827    1827 Alleghany 37005  37005        3   487     0
## 3 0.143     1.630  1828    1828     Surry 37171  37171       86  3188     5
##   NWBIR74 BIR79 SID79 NWBIR79                              geom
## 1      10  1364     0      19 MULTIPOLYGON (((-81.47276 3...
## 2      10   542     3      12 MULTIPOLYGON (((-81.23989 3...
## 3     208  3616     6     260 MULTIPOLYGON (((-80.45634 3...
```

Let's plot the county boundaries and add a county label.

```
plot(st_geometry(nc))
cc1 <- st_coordinates(st_centroid(st_geometry(nc)))
text(cc1, labels = 1:nrow(nc), col = "orange", cex = 0.5)
```



It is also possible to subset this data and plot it. You can run each line sequentially in R to see what each line of code adds to the plot. Note that this is written in base R, not with `ggplot()` where each line of code builds upon the plot.

We plot 7 areas only, and then highlight the first 5.

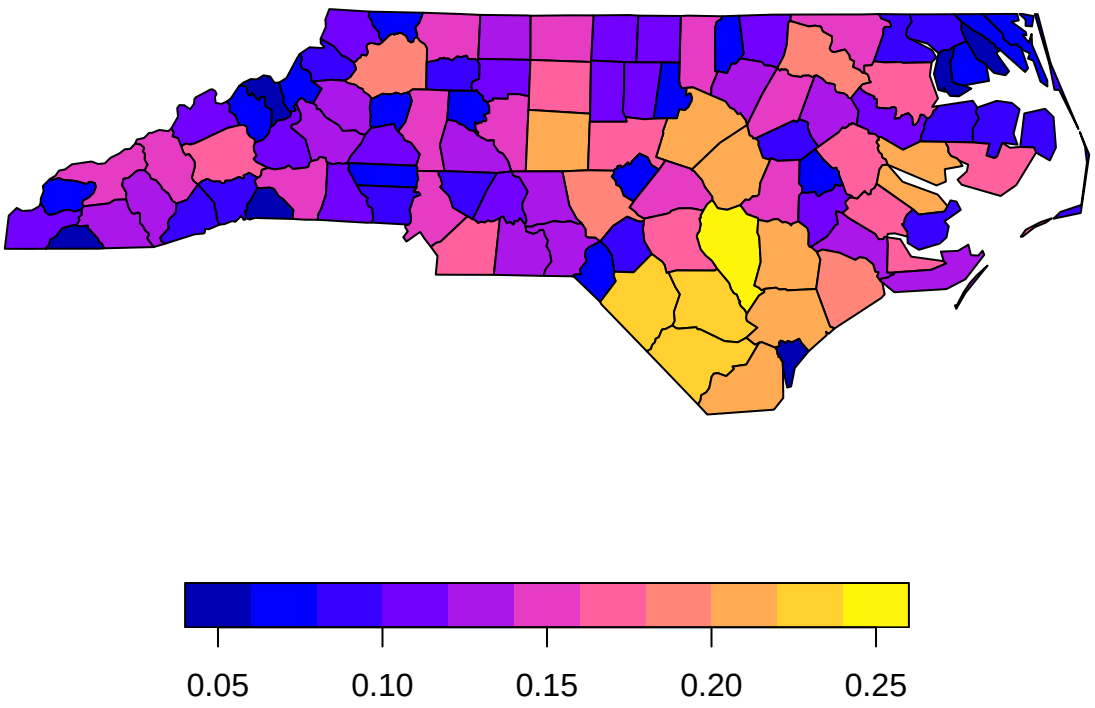
```
# Subset the data
nc5 <- nc[1:5, ]
nc7 <- nc[1:7, ]

# Plot the data
plot(st_geometry(nc7))
plot(st_geometry(nc5), add = TRUE, border = "brown")
cc <- st_coordinates(st_centroid(st_geometry(nc7)))
text(cc, labels = 1:nrow(nc7), col = "blue")
```

Visualizing Spatial Data

As you can see below, we can create a plot of the area sizes in each county in North Carolina (this is the first feature in the data frame). The area part comes into play when we index the `nc` object with a 1. If you change this to a 2, you'll see that this now plots the perimeter of each county instead, as perimeters is the second feature.

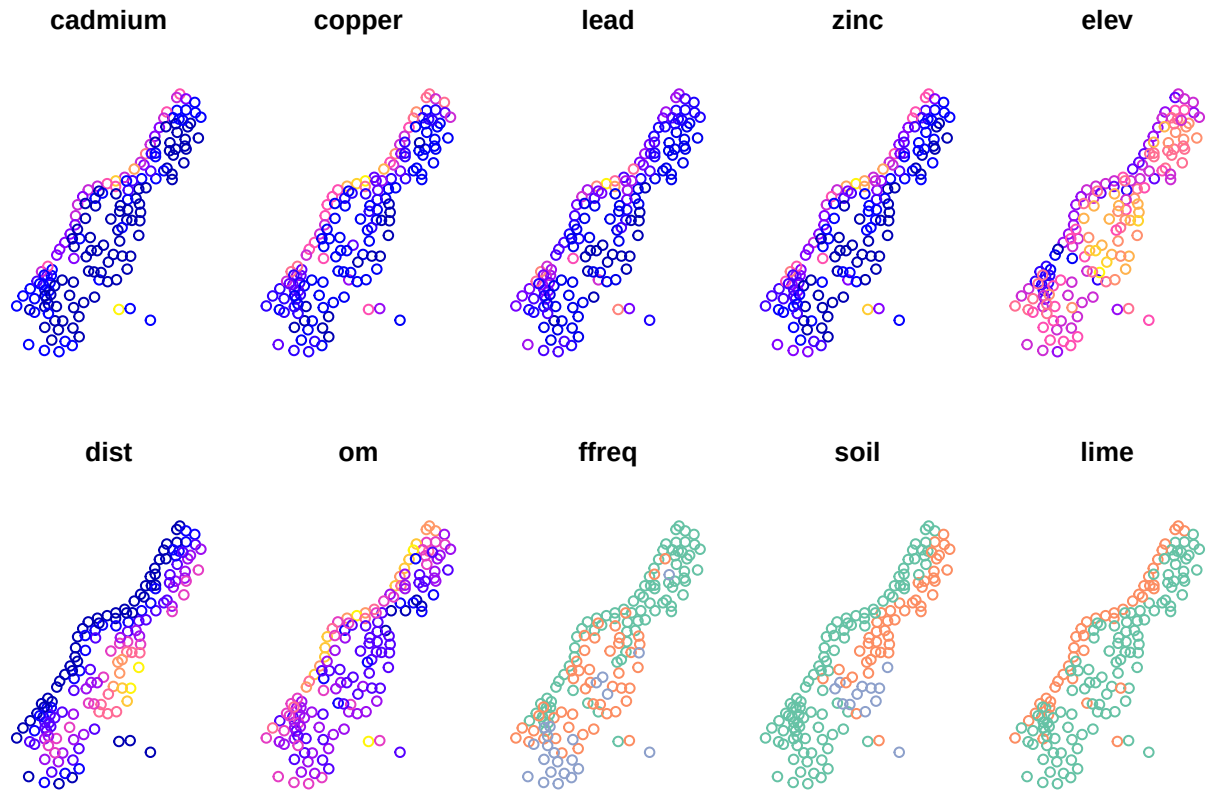
```
par(mar = c(0, 0, 1, 0))
plot(nc[1], reset = FALSE)  # reset = FALSE: we want to add to a plot with a legend
```




```
# plot(nc[1,1], col = 'grey', add = TRUE) This would grey out the
# first county
```

We can also look at the `meuse` dataset from the `sp` package and convert the data into a `sp` object, namely a `SpatialPointsDataFrame` by using the `coordinates()` function. Then, using `st_as_sf()`, we can go from an `sp` object to an `sf` object.

```
library(sp)
data(meuse, package = "sp")
coordinates(meuse) = ~x + y
m.sf = st_as_sf(meuse)
head(m.sf, n = 3)
## Simple feature collection with 3 features and 12 fields
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 181025 ymin: 333537 xmax: 181165 ymax: 333611
## CRS:           NA
##   cadmium copper lead zinc  elev       dist   om ffreq soil lime landuse dist.m
## 1    11.7     85  299 1022 7.909 0.00135803 13.6     1    1    1      Ah     50
## 2     8.6     81  277 1141 6.983 0.01222430 14.0     1    1    1      Ah     30
## 3     6.5     68  199  640 7.800 0.10302900 13.0     1    1    1      Ah    150
##                geometry
## 1 POINT (181072 333611)
## 2 POINT (181025 333558)
## 3 POINT (181165 333537)
opar = par(mar = rep(0, 4))
plot(m.sf)
```



Reading Shapefiles

ESRI (a company one of whose products is ArcGIS) shapefiles consist of three files, and this is a common form.

- The first file (`*.shp`) contains the geography of each shape.
- The second file (`*.shx`) is an index file which contains record offsets.
- The third file (`*.dbf`) contains feature attributes with one record per feature.

The Washington state Geospatial Data Archive (wagda) can be accessed here and contains data that we can read in.

As an example, consider Washington county data that was downloaded from wagda.

The data consists of the three files: `wacounty.shp`, `wacounty.shx`, `wacounty.dbf`.

The following code reads in these data and then draws a county level map of 1990 populations, and a map with centroids.

First load the libraries.

```
library(maps)  # for background map outlines
```

Note that there are problems with the files, which are sorted by using the `repair=T` argument.

The data can be downloaded from here: here

But we read in directly:

```
download.file("http://faculty.washington.edu/jonno/SISMIDmaterial/wacounty.shp",
    destfile = "wacounty.shp")
download.file("http://faculty.washington.edu/jonno/SISMIDmaterial/wacounty.shx",
    destfile = "wacounty.shx")
```

```
download.file("http://faculty.washington.edu/jonno/SISMIDmaterial/wacounty.dbf",
    destfile = "wacounty.dbf")
wacounty = st_read(dsn = ".", layer = "wacounty")
## Reading layer `wacounty' from data source `/Users/jonno/Dropbox/554-2023/STAB' using driver `ESRI Sh
## Simple feature collection with 39 features and 6 fields (with 1 geometry empty)
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -124.7312 ymin: 45.5434 xmax: -116.915 ymax: 49.0026
## CRS:           NA
class(wacounty)
## [1] "sf"         "data.frame"
head(wacounty, n = 3)
## Simple feature collection with 3 features and 6 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -119.8756 ymin: 45.8358 xmax: -116.915 ymax: 47.2616
## CRS:           NA
##            AreaName AreaKey INTPTLAT  INTPTLNG TotPop90 CNTY
## 1  WA, Adams County   53001 46.98899 -118.5569    13603    1
## 2 WA, Asotin County   53003 46.18248 -117.1850    17605    3
## 3 WA, Benton County   53005 46.24764 -119.5015   112560    5
##                          geometry
## 1 MULTIPOLYGON (((-118.9784 4...
## 2 MULTIPOLYGON (((-117.2276 4...
## 3 MULTIPOLYGON (((-119.8728 4...
```

Let's see what these variables look like: we see county names and FIPS codes.

```
names(wacounty)
## [1] "AreaName" "AreaKey"  "INTPTLAT" "INTPTLNG" "TotPop90" "CNTY"     "geometry"
wacounty$AreaName[1:3]
## [1] "WA, Adams County"  "WA, Asotin County" "WA, Benton County"
wacounty$AreaKey[1:3]
## [1] "53001" "53003" "53005"
# head(wacounty)
```

Drawing a map

We look at some variables.

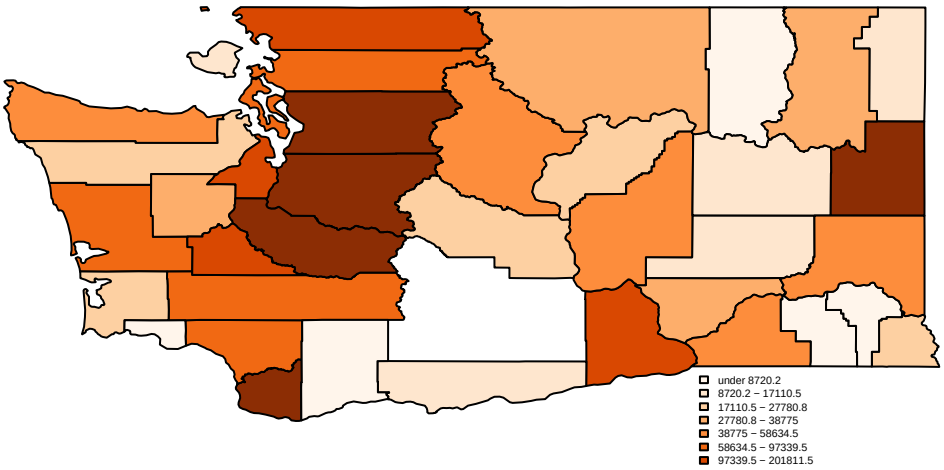
```
wacounty$INTPTLAT[1:3]  # latitude
## [1] 46.98899 46.18248 46.24764
wacounty$INTPTLNG[1:3]  # longitude
## [1] -118.5569 -117.1850 -119.5015
wacounty$CNTY[1:3]
## [1] "1" "3" "5"
wacounty$TotPop90[1:3]
## [1]  13603  17605 112560
```

We look at some variables, and then set up the colors to map. We map 1990 Washington population counts by census tracts.

```
# function to make legend; copy from maptools package
leglabs = function(vec, under = "under", over = "over", between = "-") {
    x <- vec
    lx <- length(x)
```

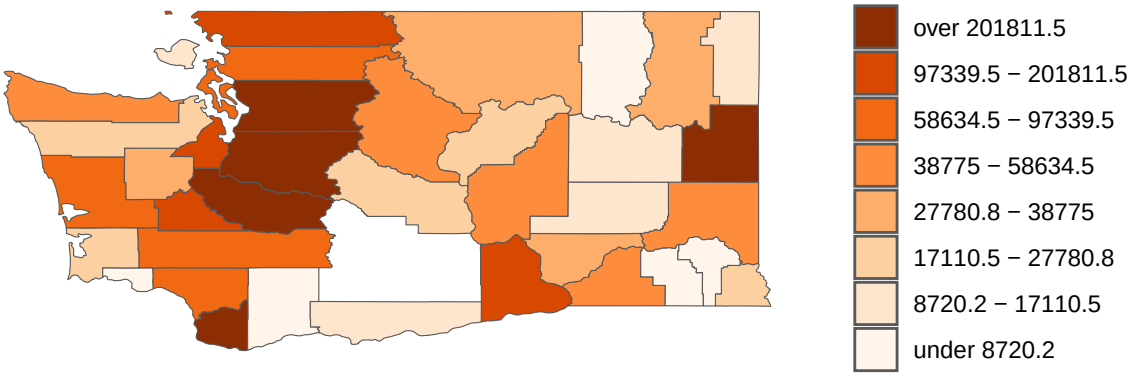
```
    if (lx < 3)
        stop("vector too short")
    res <- character(lx - 1)
    res[1] <- paste(under, x[2])
    for (i in 2:(lx - 2)) res[i] <- paste(x[i], between, x[i + 1])
    res[lx - 1] <- paste(over, x[lx - 1])
    res
}
```

```
plotvar <- wacounty$TotPop90  # variable we want to map
summary(plotvar)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2248   17110   38775  124787   97340 1507319
nclr <- 8  # next few lines set up the color scheme for plotting
plotclr <- brewer.pal(nclr, "Oranges")
brks <- round(quantile(plotvar, probs = seq(0, 1, 1/(nclr))), digits = 1)
colornum <- findInterval(plotvar, brks, all.inside = T)
colcode <- plotclr[colornum]
plot(st_geometry(wacounty), col = colcode)
legend(-119, 46, legend = leglabs(round(brks, digits = 1)), fill = plotclr,
    cex = 0.35, bty = "n")
```



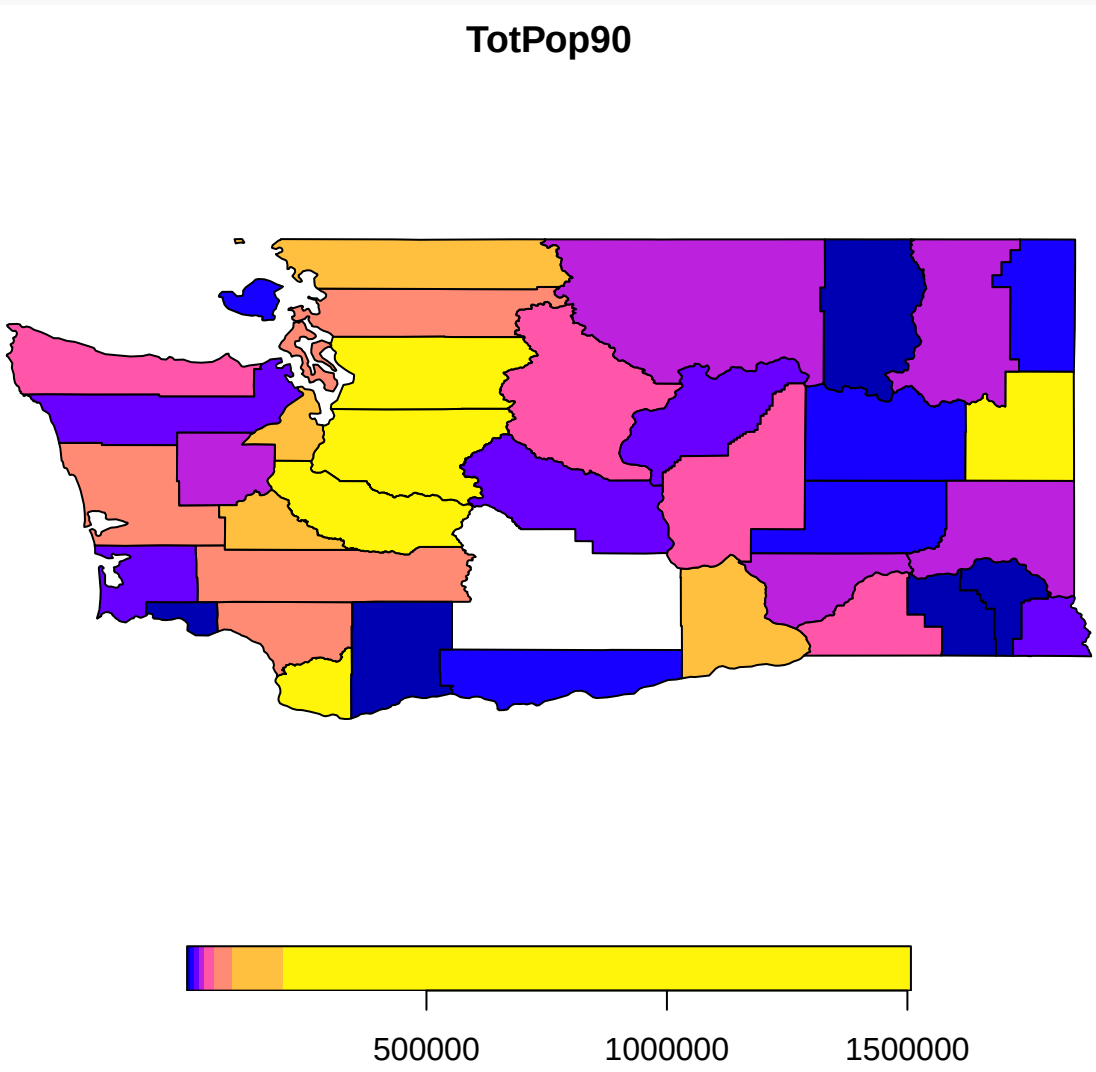
In ggplot:

```
ggplot(data = wacounty) + geom_sf(aes(fill = colcode)) + scale_fill_manual(values = rev(plotclr),
    labels = rev(leglabs(round(brks, digits = 1)))) + theme_bw() + coord_sf() +
    xlab("") + ylab("") + theme(legend.title = element_blank(), panel.grid = element_blank(),
    panel.border = element_blank(), axis.ticks = element_blank(), axis.text = element_blank())
```



As another alternative we can use the `plot` function, which uses base plot for spatial data with attributes. We map the 1990 Washington population counts by county.

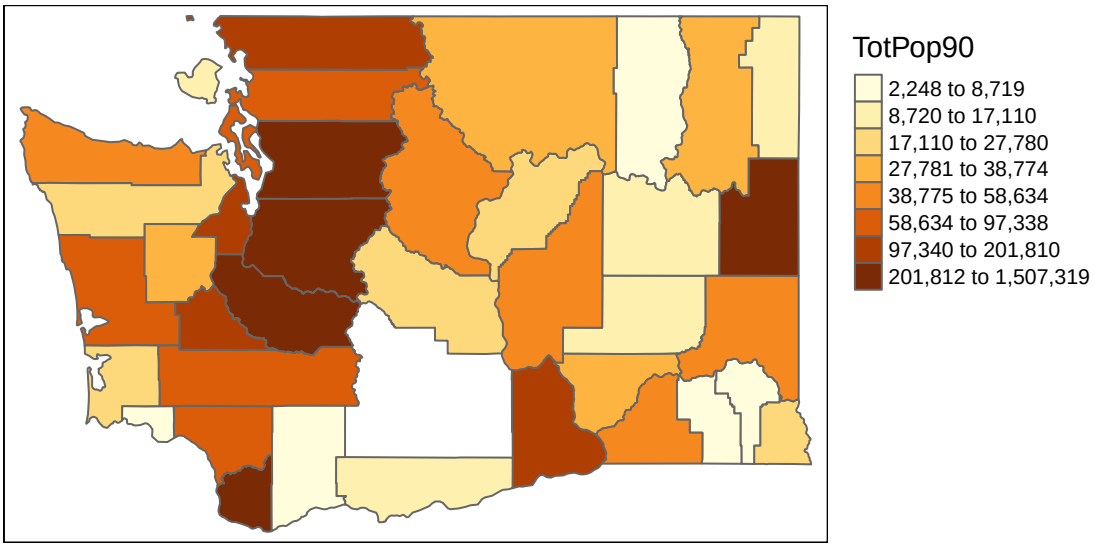
```
plot(wacounty["TotPop90"], breaks = brks)
```



We can also use `tmap`:

```
library(tmap)
sel = !st_is_empty(wacounty)
tm_shape(st_set_crs(wacounty, 4326)[sel, ]) + tm_polygons("TotPop90", breaks = brks) +
```

```
    tm_layout(legend.outside = TRUE)
```



For illustration, we define our own cutpoints by hand.

```
summary(wacounty$TotPop90)
wacounty$cats <- cut(wacounty$TotPop90, breaks = c(0, 17000, 39000, 1e+05,
    1520000), labels = c("<17,000", "17,000-39,000", "39,000-100,000",
    ">100,000"))
tm_shape(st_set_crs(wacounty, 4326)[sel, ]) + tm_polygons("cats") + tm_layout(legend.outside = TRUE)
```

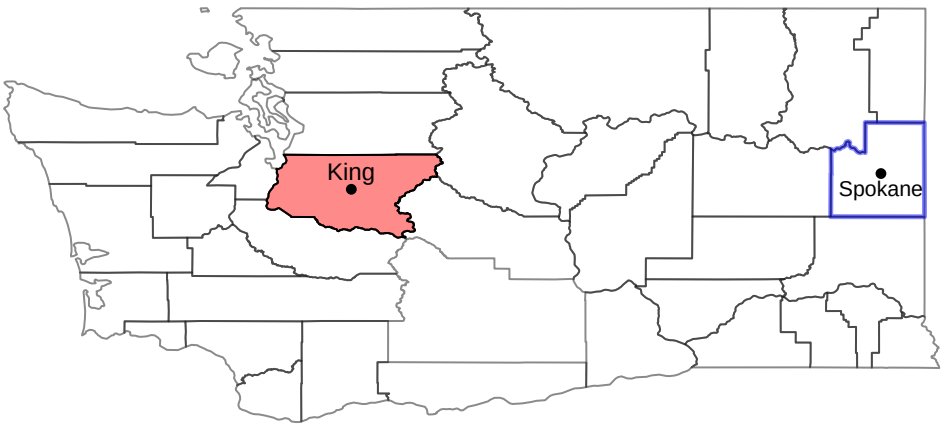
We now highlight a county

```
# identify counties of interest
xx = which(wacounty$CNTY == 33)
xx2 = which(wacounty$CNTY == 63)

# plot the whole state
plot(st_geometry(wacounty), border = "#00000075")

# highlight counties of interest
plot(st_geometry(wacounty[xx, ]), col = "#ff000075", add = T)
plot(st_geometry(wacounty[xx2, ]), col = NA, border = "#0000ff75", add = T,
    lwd = 2.5)

# Add some labels
text(st_coordinates(st_centroid(st_geometry(wacounty[xx, ]))), "King",
    cex = 0.75, pos = 3, offset = 0.25)
text(st_coordinates(st_centroid(st_geometry(wacounty[xx2, ]))), "Spokane",
    cex = 0.7, pos = 1, offset = 0.25)
points(st_coordinates(st_centroid(st_geometry(wacounty[c(xx, xx2), ]))),
    pch = 16, cex = 0.75)
```

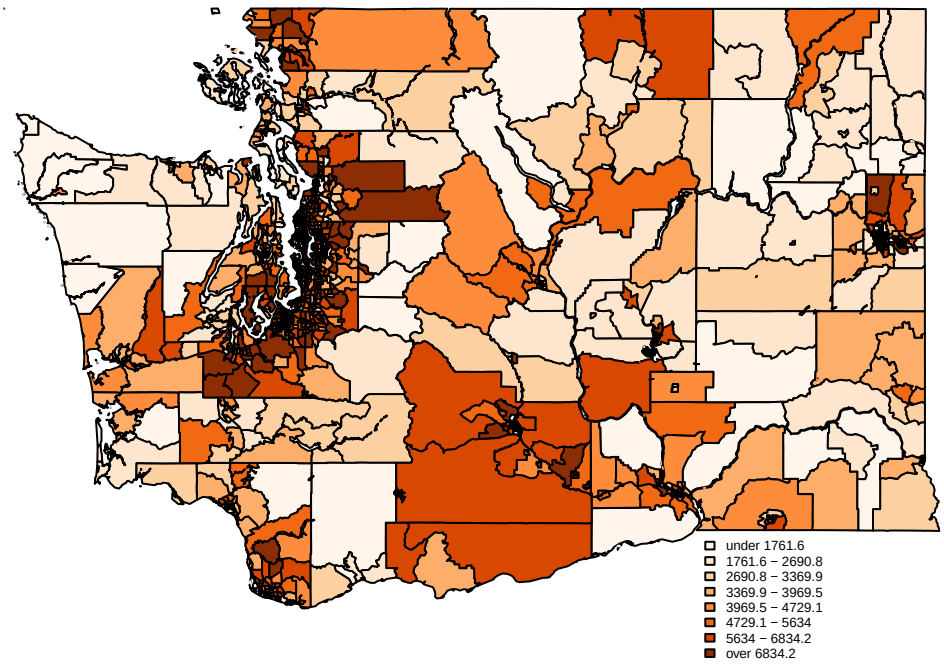


Now let's repeat for census tracts.

```
## Reading Shapefiles
download.file("http://faculty.washington.edu/jonno/SISMIDmaterial/watract.shp",
    destfile = "watract.shp")
download.file("http://faculty.washington.edu/jonno/SISMIDmaterial/watract.shx",
    destfile = "watract.shx")
download.file("http://faculty.washington.edu/jonno/SISMIDmaterial/watract.dbf",
    destfile = "watract.dbf")
watract = st_read(dsn = ".", layer = "watract")
## Reading layer `watract' from data source `/Users/jonno/Dropbox/554-2023/STAB' using driver `ESRI Sha
## Simple feature collection with 1152 features and 7 fields (with 1 geometry empty)
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -124.8485 ymin: 45.55 xmax: -116.9174 ymax: 49.0025
## CRS:           NA
```

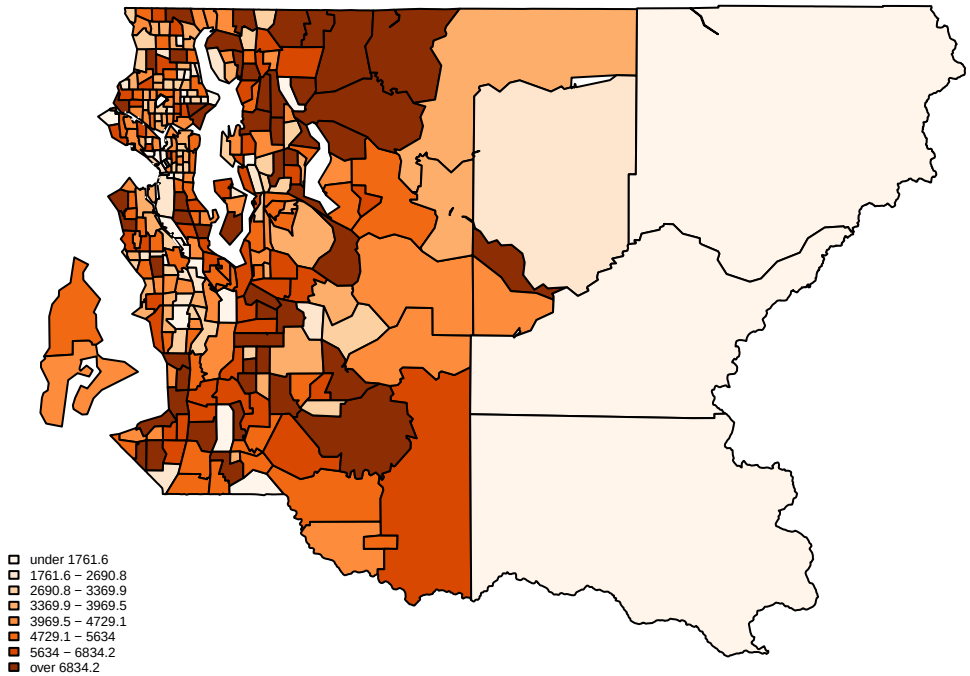
Drawing a census tract map: we repeat but now map populations at the census tract level.

```
watract <- st_read("watract.shp") |>
    st_make_valid() |>
    st_set_crs(4326)
## Reading layer `watract' from data source
##   `/Users/jonno/Dropbox/554-2023/STAB/watract.shp' using driver `ESRI Shapefile'
## Simple feature collection with 1152 features and 7 fields (with 1 geometry empty)
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -124.8485 ymin: 45.55 xmax: -116.9174 ymax: 49.0025
## CRS:           NA
names(watract)
## [1] "AreaName" "AreaKey"  "INTPTLAT" "INTPTLNG" "TotPop90" "TRACT"    "CNTY"
## [8] "geometry"
plotvar <- watract$TotPop90  # variable we want to map
brks <- round(quantile(plotvar, probs = seq(0, 1, 1/(nclr))), digits = 1)
colornum <- findInterval(plotvar, brks, all.inside = T)
colcode <- plotclr[colornum]
plot(st_geometry(watract), col = colcode)
legend(-119, 46, legend = leglabs(round(brks, digits = 1)), fill = plotclr,
    cex = 0.4, bty = "n")
```



We zoom in on King County

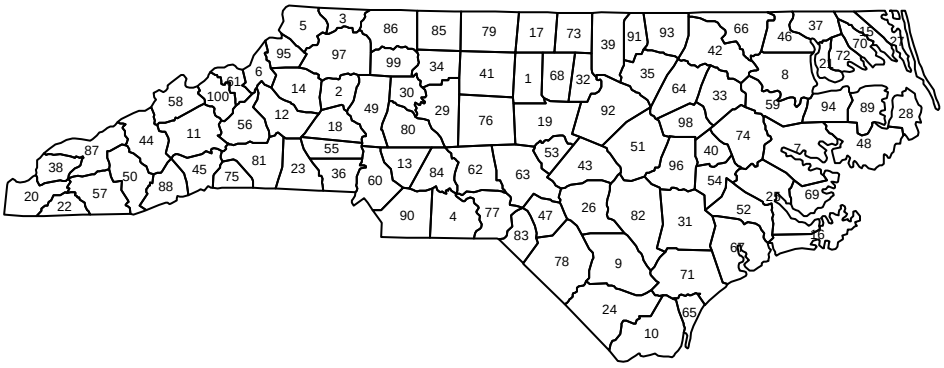
```
xx = which(watract$CNTY == 33)
plot(st_geometry(watract[xx, ]), col = colcode[xx])
legend("bottomleft", legend = leglabs(round(brks, digits = 1)), fill = plotclr,
    cex = 0.4, bty = "n")
```



A county map of North Carolina with text

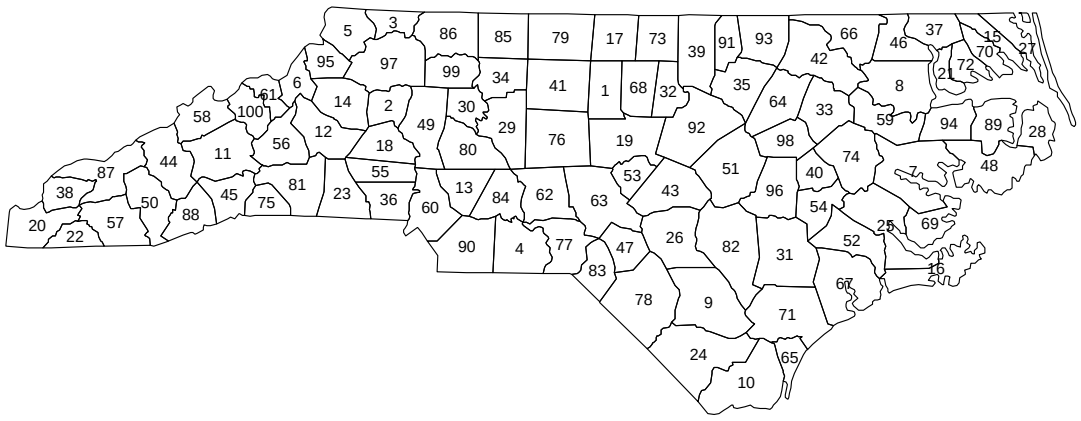
```
library(maps)
map("county", "North Carolina", fill = TRUE, plot = FALSE) |>
    st_as_sf() |>
```

```
    st_make_valid() -> nc
centroids = st_centroid(st_geometry(nc)) |>
    st_coordinates()
plot(st_geometry(nc))
text(centroids[, 1], centroids[, 2], 1:100, offset = 0, cex = 0.4)
```



In ggplot:

```
ggplot() + geom_sf(data = nc, col = "black", fill = NA) + geom_text(aes(x = centroids[,
    1], y = centroids[, 2], label = 1:100), size = 2) + theme_bw() + coord_sf() +
    xlab("") + ylab("") + theme(legend.title = element_blank(), panel.grid = element_blank(),
    panel.border = element_blank(), axis.ticks = element_blank(), axis.text = element_blank())
```



John Snow Example

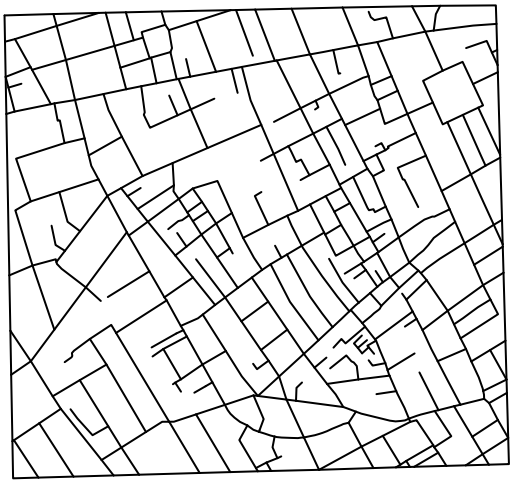
For fun, let's look at the poster child of health mapping.

The Snow data consists of the relevant 1854 London streets, the location of 578 deaths from cholera, and the position of 13 water pumps (wells) that can be used to re-create John Snow's map showing deaths from cholera in the area surrounding Broad Street, London in the 1854 outbreak.

```
library(HistData)
data(Snow.deaths)
data(Snow.pumps)
data(Snow.streets)
# data(Snow.polygons)
```

We first create an `sfc` object containing the coordinates of the streets using the `st_linestring()` function; `st_sfc()` combines all the individual `LINESTRING` pieces:

```
# Streets
slist <- split(Snow.streets[, c("x", "y")], as.factor(Snow.streets[, "street"]))
m <- lapply(slist, as.matrix) |>
    lapply(st_linestring) |>
    st_sfc()
plot(m)
```



Display the streets and then add the deaths and pumps (with labels). The red squares are deaths, blue triangles are pumps

```
plot(m, col = "gray")
# deaths:
p <- st_as_sf(Snow.deaths, coords = c("x", "y"))
plot(p, add = TRUE, col = "red", pch = 15, cex = 0.6)
# pumps:
spp <- st_as_sf(Snow.pumps, coords = c("x", "y"))
plot(spp, add = TRUE, col = "blue", pch = 17, cex = 1)
text(Snow.pumps[, c("x", "y")], labels = Snow.pumps$label, pos = 1, cex = 0.8)
```