Working with Geographic Data

Greg Ridgeway (gridge@upenn.edu)

July 11, 2021

Introduction

Geographic data includes

- point - crime locations, locations of patrol cars
- lines - roads, paths, routes
- polygons - area within city boundaries, areas within 1000 ft of a school

Geographic data also include the data that go along with each of these shapes such as the area or length of the geographic object, the name of the object (e.g. City of Los Angeles), and other characteristics of the geography (e.g. population, date established).

Many questions about crime and the justice system involve the use of geographic data. In this section we will work toward answering questions about the race distribution of residents inside Los Angeles gang injunction zones, the number of crimes with 100 feet of Wilshire Blvd, and examine crimes near Metrorail stations.

We will be learning how to use the `sf` package for managing spatial data, the `rgeos` package for manipulating spatial objects, and the `jsonlite` package to look at modern methods for accessing data.

Exploring Los Angeles gang injunction maps

To start, load the `sf` (simple features) package to get access to all the essential spatial tools. Also load the `lubridate` package since we'll need to work with dates along the way.

```
library(sf)
```

```
Linking to GEOS 3.9.0, GDAL 3.2.1, PROJ 7.2.1
```

```
library(lubridate)
```

```
Attaching package: 'lubridate'
```

```
The following objects are masked from 'package:base':

    date, intersect, setdiff, union
```

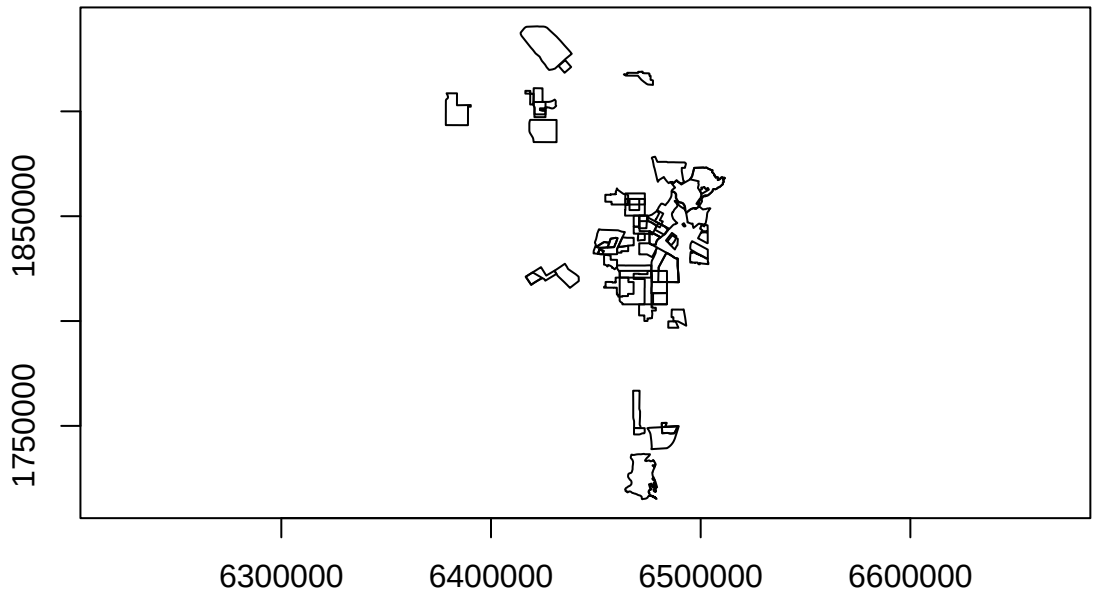
All of the spatial functions in the `sf` package have a prefix `st_`(spatial/temporal). We'll first read in `allinjunction.shp`, a shapefile containing the geographic definition of Los Angeles gang injunctions. You should have a collection of four files related to `allinjunctions`, a .dbf file, a .prj file, a .shx file, and a .shp file. even though the `st_read()` function appears to just ask for the .shp file, you need to have all four files in the same folder.

```
mapSZ <- st_read("11_shapefiles_and_data/allinjunctions.shp")
```

```
Reading layer 'allinjunctions' from data source
  'Z:\Penn\CRIM602\notes\R4crim\11_shapefiles_and_data\allinjunctions.shp'
  using driver 'ESRI Shapefile'
Simple feature collection with 65 features and 13 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: 6378443 ymin: 1715015 xmax: 6511590 ymax: 1940541
Projected CRS: Lambert_Conformal_Conic
```

Let's take a look at what we have read in. `mapSZ` has a lot of data packed into it that we will explore. To make the plot we need to ask R to just extract the geometry.

```
plot(st_geometry(mapSZ))
axis(1); axis(2); box()
```



We've added the x and y axis so that you note the scale. We can check how the geography is projected.

```
st_crs(mapSZ)
```

```
Coordinate Reference System:
  User input: Lambert_Conformal_Conic
  wkt:
PROJCRS["Lambert_Conformal_Conic",
    BASEGEOGCRS["NAD83",
        DATUM["North American Datum 1983",
            ELLIPSOID["GRS 1980",6378137,298.257222101,
                LENGTHUNIT["metre",1]],
            ID["EPSG",6269]],
        PRIMEM["Greenwich",0,
            ANGLEUNIT["Degree",0.0174532925199433]]],
    CONVERSION["unnamed",
        METHOD["Lambert Conic Conformal (2SP)",
            ID["EPSG",9802]],
        PARAMETER["Latitude of false origin",33.5,
            ANGLEUNIT["Degree",0.0174532925199433],
            ID["EPSG",8821]],
        PARAMETER["Longitude of false origin",-118,
```

```
            ANGLEUNIT["Degree",0.0174532925199433],
            ID["EPSG",8822]],
        PARAMETER["Latitude of 1st standard parallel",34.0333333333333,
            ANGLEUNIT["Degree",0.0174532925199433],
            ID["EPSG",8823]],
        PARAMETER["Latitude of 2nd standard parallel",35.4666666666667,
            ANGLEUNIT["Degree",0.0174532925199433],
            ID["EPSG",8824]],
        PARAMETER["Easting at false origin",6561666.66666667,
            LENGTHUNIT["US survey foot",0.304800609601219],
            ID["EPSG",8826]],
        PARAMETER["Northing at false origin",1640416.66666667,
            LENGTHUNIT["US survey foot",0.304800609601219],
            ID["EPSG",8827]]],
    CS[Cartesian,2],
        AXIS["(E)",east,
            ORDER[1],
            LENGTHUNIT["US survey foot",0.304800609601219,
                ID["EPSG",9003]]],
        AXIS["(N)",north,
            ORDER[2],
            LENGTHUNIT["US survey foot",0.304800609601219,
                ID["EPSG",9003]]]]
```

Of greatest importance is to notice that the projection is not latitude and longitude, although this is clearly the case from the previous plot. The coordinate system is the Lambert Conic Conformal (LCC) tuned specifically for the Los Angeles area. This coordinate system is oriented for the North American continental plate (NAD83), so precise that this coordinate system moves as North America tectonic plate moves (2cm per year!). Also note that the unit of measurement is in feet. Whenever we compute a distance or area with these data, the units will be in feet or square feet.

Let's examine the data attached to each polygon. Here are the first three rows.

```
mapSZ[1:3,]
```

```
Simple feature collection with 3 features and 13 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: 6378443 ymin: 1893403 xmax: 6438243 ymax: 1924413
Projected CRS: Lambert_Conformal_Conic
       AREA PERIMETER GANG_INJ11 GANG_INJ_1 SHADESYM           NAME Inj_
1  18152790  17048.67          2          9        8       Foothill   09
2  11423653  20863.36          3          1      760 Langdon Street   04
3 129459650  54419.87          4         11      140    Canoga Park   11
   case_no      Safety_Zn    LAPD_Div      Pre_Date      Perm_Date
1 PC027254       Foothill    Foothill          <NA>  Aug. 22, 2001
2 LC048292 Langdon Street  Devonshire  May 20, 1999  Feb. 17, 2000
3 BC267153    Canoga Park West Valley Feb. 25, 2002 April 24, 2002
```

```
             gang_name                       geometry
1 Pacoima Project Boys POLYGON ((6435396 1924413, ...
2       Langdon Street POLYGON ((6418722 1908442, ...
3  Canoga Park Alabama POLYGON ((6379791 1908614, ...
```

Each polygon in the map is associated with a specific gang injunction. The data attached to each polygon gives details about the associated injunction, such as the name of the injunction, in which LAPD division it is located, dates of the preliminary and permanent injunction, and the name of the gang that the injunction targets.

We can extract the coordinates of an injunction. Let's grab the coordinates of the polygon for the first injunction.

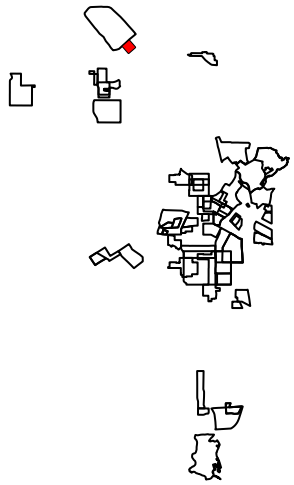
```
st_coordinates(mapSZ[1,])
```

```
            X       Y L1 L2
 [1,] 6435396 1924413  1  1
 [2,] 6435607 1924174  1  1
 [3,] 6435792 1923960  1  1
 [4,] 6435872 1923867  1  1
 [5,] 6436158 1923545  1  1
 [6,] 6436349 1923325  1  1
 [7,] 6437295 1922239  1  1
 [8,] 6438231 1921163  1  1
 [9,] 6438243 1921150  1  1
[10,] 6437992 1920931  1  1
[11,] 6437743 1920714  1  1
[12,] 6437495 1920498  1  1
[13,] 6437246 1920282  1  1
[14,] 6436874 1919958  1  1
[15,] 6436472 1919608  1  1
[16,] 6435940 1919144  1  1
[17,] 6435771 1918998  1  1
[18,] 6435633 1918878  1  1
[19,] 6435311 1918597  1  1
[20,] 6435143 1918451  1  1
[21,] 6435086 1918401  1  1
[22,] 6434689 1918857  1  1
[23,] 6434139 1919485  1  1
[24,] 6433857 1919808  1  1
[25,] 6433742 1919938  1  1
[26,] 6433189 1920572  1  1
[27,] 6433040 1920743  1  1
[28,] 6432908 1920894  1  1
[29,] 6432706 1921120  1  1
[30,] 6432500 1921348  1  1
[31,] 6432467 1921385  1  1
[32,] 6432279 1921596  1  1
```

```
[33,] 6432227 1921658  1  1
[34,] 6432296 1921718  1  1
[35,] 6432466 1921866  1  1
[36,] 6433399 1922679  1  1
[37,] 6433404 1922683  1  1
[38,] 6433857 1923073  1  1
[39,] 6434399 1923545  1  1
[40,] 6434790 1923885  1  1
[41,] 6434822 1923914  1  1
[42,] 6435396 1924413  1  1
```

Let's highlight the first injunction in our map. We can use `subset()` to select shapes using any feature listed in `names(mapSZ)`. We'll select it using its case number. Use `add=TRUE` to add the second plot to the first.

```
plot(st_geometry(mapSZ))
plot(st_geometry(subset(mapSZ, case_no=="PC027254")),
     col="red",
     border=NA,
     add=TRUE)
```



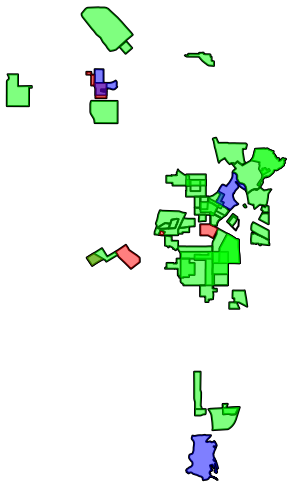
Now we can see this tiny injunction shaded in red at the top of the map.

Turning back to the data attached to the map, we need to do some clean up on the dates. They are not in standard form and include some typos. We'll fix the spelling errors and use `lubridate` to standardize those dates. We'll also add a `startDate` feature as the smaller of the preliminary injunction date and the permanent injunction date using the pairwise minimum function `pmin()`.

```
mapSZ$Pre_Date <- as.character(mapSZ$Pre_Date)
mapSZ$Pre_Date <- gsub("Jne", "June", mapSZ$Pre_Date)
mapSZ$Pre_Date <- gsub("Sept\\.", "September", mapSZ$Pre_Date)
mapSZ$Pre_Date <- mdy(mapSZ$Pre_Date)
mapSZ$Perm_Date <- as.character(mapSZ$Perm_Date)
mapSZ$Perm_Date <- gsub("Sept\\.", "September ", mapSZ$Perm_Date)
mapSZ$Perm_Date <- mdy(mapSZ$Perm_Date)
mapSZ$startDate <- pmin(mapSZ$Pre_Date, mapSZ$Perm_Date, na.rm = TRUE)
```

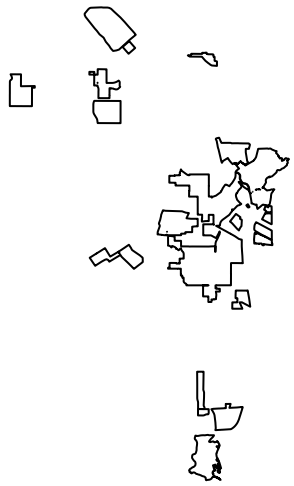
Now let's highlight the injunctions before 2000 in red, those between 2000 and 2010 in green, and those after 2010 in blue. Since many of the polygons overlap, we're going to make the colors a little transparent so that we can see the overlap. `rgb()` is a function for generating colors by mixing the primary source colors red, green, and blue. The function has four parameters. The first three tell R how much red, green, and blue, respectively, to mix together where 0 tells R to use none of that color and 1 tells r to use all of that color. The fourth parameter sets the transparency. So to make a red that is half transparent we use `rgb(1, 0, 0, 0.5)`.

```
plot(st_geometry(mapSZ))
plot(st_geometry(subset(mapSZ, year(startDate)< 2000)),
     col=rgb(1,0,0,0.5), border=NA, add=TRUE)
plot(st_geometry(subset(mapSZ,year(startDate)>=2000 & year(startDate)<2010)),
     col=rgb(0,1,0,0.5),border=NA,add=TRUE)
plot(st_geometry(subset(mapSZ,year(startDate)>2010)),
     col=rgb(0,0,1,0.5),border=NA,add=TRUE)
```



When we loaded up the **sf** package, we also gained access to the GEOS library of geographic operations. For example, we can union (combine) all of the polygons together into one shape.

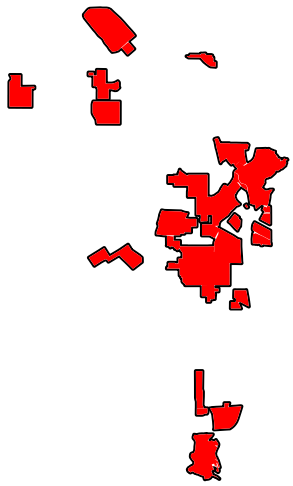
```
mapSZunion <- st_union(mapSZ)
plot(st_geometry(mapSZunion))
```



Any overlapping injunctions have been combined into one polygon. `mapSZunion` now contains this unioned collection of polygons. Note that `mapSZunion` no longer has any data attached to it. Once we union polygons together, it is no longer obvious how to combine their associated data.

Let's draw a polygon defining the area of Los Angeles that is within 500 feet of an injunction. First, we will double check the units this map uses.

```
st_crs(mapSZunion)$units
# create a buffer 500 feet around the injunctions
mapSZ500ft <- st_buffer(mapSZunion, dist=500)
plot(st_geometry(mapSZ500ft))
plot(st_geometry(mapSZunion), col="red", border=NA, add=TRUE)
```

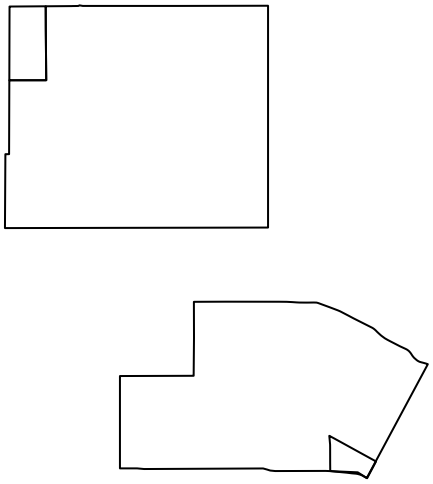


```
[1] "us-ft"
```

Every injunction area now has a black line outlining the 500-foot buffer.

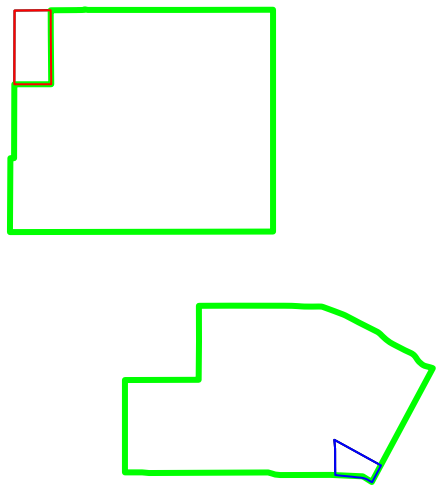
Previously we had to clean up some typos on the injunction dates data. The data can also have errors in the geography that requires fixing. Have a look at the MS13 gang injunction.

```
mapSZms13 <- subset(mapSZ, case_no=="BC311766")
plot(st_geometry(mapSZms13))
```



The injunction has two mutually exclusive polygons that define the injunction. Both have strange artifacts. Examining the`mapSZms13` object we can see that it has four polygons. Let's color them so we can see which one is which. The ones with the smallest areas must be the artifacts.

```
mapSZms13
plot(st_geometry(mapSZms13))
plot(st_geometry(mapSZms13[c(1,3),]), add=TRUE, border="green", lwd=3)
plot(st_geometry(mapSZms13[2,]), add=TRUE, border="red", lwd=1)
plot(st_geometry(mapSZms13[4,]), add=TRUE, border="blue", lwd=1)
```



```
Simple feature collection with 4 features and 14 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: 6463941 ymin: 1841325 xmax: 6479076 ymax: 1858250
Projected CRS: Lambert_Conformal_Conic
       AREA PERIMETER GANG_INJ11 GANG_INJ_1 SHADESYM                   NAME
11 70459367 34712.862         13          1        0 Rampart/East Hollywood
12  3452577  7902.383         14          1        0 Rampart/East Hollywood
16 49977447 32248.724         18          2        0 Rampart/East Hollywood
20  1403021  5211.601         22          2        0 Rampart/East Hollywood
   Inj_  case_no                    Safety_Zn    LAPD_Div   Pre_Date  Perm_Date
11   18 BC311766 Rampart/East Hollywood (MS) Hwd/Wil/Rmp 2004-04-08 2004-05-10
12   18 BC311766 Rampart/East Hollywood (MS) Hwd/Wil/Rmp 2004-04-08 2004-05-10
16   18 BC311766 Rampart/East Hollywood (MS) Hwd/Wil/Rmp 2004-04-08 2004-05-10
20   18 BC311766 Rampart/East Hollywood (MS) Hwd/Wil/Rmp 2004-04-08 2004-05-10
          gang_name                          geometry  startDate
11 Mara Salvatrucha POLYGON ((6470191 1858229, ... 2004-04-08
12 Mara Salvatrucha POLYGON ((6465399 1858217, ... 2004-04-08
16 Mara Salvatrucha POLYGON ((6468057 1844983, ... 2004-04-08
20 Mara Salvatrucha POLYGON ((6477222 1841927, ... 2004-04-08
```

The Los Angeles City Attorney's Office has the correct injunction posted on its website here. Let's

clear out the weird artifacts to repair the gang injunction geometry. We can use `st_union()` to combine all the polygons together.

```
a <- st_union(mapSZms13)
a
```

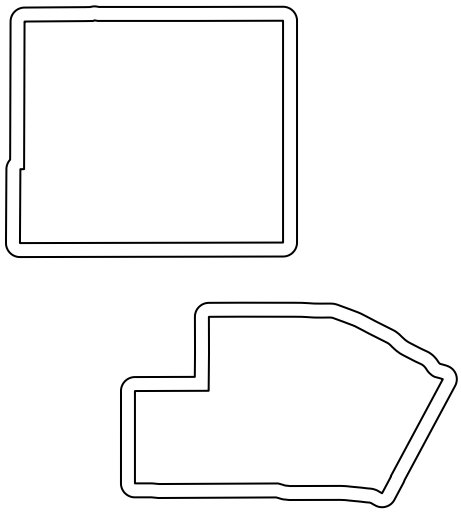
```
MULTIPOLYGON (((6473357 1850297, 6470707 185029...

Geometry set for 1 feature
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 6463941 ymin: 1841325 xmax: 6479076 ymax: 1858250
Projected CRS: Lambert_Conformal_Conic
```

Remember that `st_union()` will eliminate the associated data elements. Let's borrow all the data from the first polygon, combine it with our unioned polygons, and use `st_sf()` to make a new simple features object.

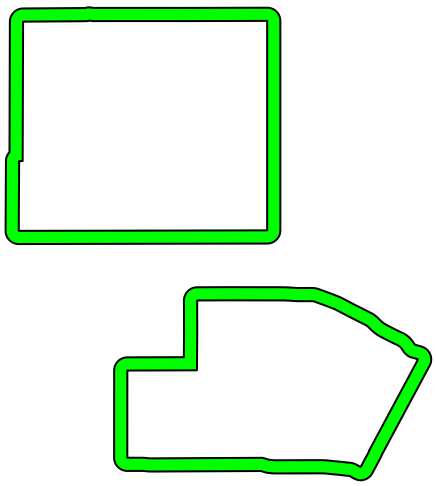
```
mapSZms13 <- st_sf(mapSZms13[1,c("NAME","case_no","Safety_Zn","gang_name","startDate")],
                   geometry=a)
```

Now let's plot the final MS13 gang injunction safety zone and color in a 500-foot buffer around it. `st_difference()` computes the "difference" between two geometric objects. Here we take the polygon defined by being 500 feet out from the MS13 injunction area and "subtract" the injunction area leaving a sort of donut around the injunction area.

```
plot(st_geometry(mapSZms13))
plot(st_geometry(st_buffer(mapSZms13, dist=500)), add=TRUE)
```



```
mapSZmapBuf <- st_difference(st_geometry(st_buffer(mapSZms13, dist=500)),
                             st_geometry(mapSZms13))
plot(st_geometry(mapSZmapBuf), col="green")
```



Exercises

1. Find the largest and smallest safety zones (use `st_area(mapSZ)`)
2. Plot all the safety zones. Color the largest in one color and the smallest in another color
3. Use `st_overlaps(mapSZ, mapSZ, sparse=FALSE)` or `print(st_overlaps(mapSZ, mapSZ, sparse=TRUE), n=Inf, max_nb=Inf)` to find two safety zones that overlap (not just touch at the edges)
4. With the two safety zones that you found in the previous question, plot using three different colors the first safety zone, second safety zone, and their intersection (hint: use `st_intersection()`)

Using TIGER files from the US Census to merge in other geographic data

THE US Census Bureau provides numerous useful geographic data files. We will use their TIGER files to get a map of the City of Los Angeles and we will get the census tracts that intersect with the city. Once you know an area's census tract, you can obtain data on the population of the area. All of the tiger files are available at https://www.census.gov/cgi-bin/geo/shapefiles/index.php.

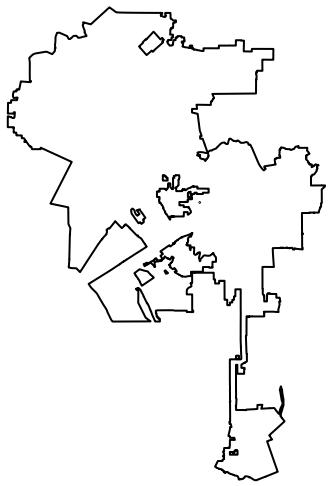
First, we will extract an outline of the city. The file `tl_2019_06_place.shp` file is a TIGER line file, created in 2019, for state number 6 (California is 6th in alphabetical order), and contains all of the places (cities and towns). Here's the entire state.

```
mapCAplaces <- st_read("11_shapefiles_and_data/tl_2019_06_place.shp")
plot(st_geometry(mapCAplaces))
```

```
Reading layer 'tl_2019_06_place' from data source
  'Z:\Penn\CRIM602\notes\R4crim\11_shapefiles_and_data\tl_2019_06_place.shp'
  using driver 'ESRI Shapefile'
Simple feature collection with 1521 features and 16 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -124.2695 ymin: 32.53433 xmax: -114.229 ymax: 41.99317
Geodetic CRS:  NAD83
```

And here is just the part of that shapefile containing Los Angeles.

```
mapLA <- subset(mapCAplaces, NAMELSAD=="Los Angeles city")
plot(st_geometry(mapLA))
```



Now let's load in the census tracts for all of California.

```
mapCens <- st_read("11_shapefiles_and_data/tl_2019_06_tract.shp")
```

```
Reading layer 'tl_2019_06_tract' from data source
  'Z:\Penn\CRIM602\notes\R4crim\11_shapefiles_and_data\tl_2019_06_tract.shp'
  using driver 'ESRI Shapefile'
Simple feature collection with 8057 features and 12 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -124.482 ymin: 32.52883 xmax: -114.1312 ymax: 42.0095
Geodetic CRS:  NAD83
```

`mapCens` contains polygons for all census tracts in California. That's a lot more than we need. we just need the ones that overlap with Los Angeles. `st_intersects()` can help us determine which census tracts are in Los Angeles. However, the following gives us a warning about assuming planar coordinates.

```
st_overlaps(mapCens, mapLA)
```

```
although coordinates are longitude/latitude, st_overlaps assumes that they are planar
```

Both `mapLA` and `mapCens` use the latitude/longitude coordinate system, which is not the same as the coordinate system we are using for the gang injunctions.

```
st_crs(mapLA)
```

```
Coordinate Reference System:
  User input: NAD83
  wkt:
GEOGCRS["NAD83",
    DATUM["North American Datum 1983",
        ELLIPSOID["GRS 1980",6378137,298.257222101,
            LENGTHUNIT["metre",1]]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["latitude",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433]],
        AXIS["longitude",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
    ID["EPSG",4269]]
```

```
st_crs(mapCens)
```

```
Coordinate Reference System:
  User input: NAD83
  wkt:
GEOGCRS["NAD83",
    DATUM["North American Datum 1983",
        ELLIPSOID["GRS 1980",6378137,298.257222101,
            LENGTHUNIT["metre",1]]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["latitude",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433]],
        AXIS["longitude",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
    ID["EPSG",4269]]
```

```
st_crs(mapSZ)
```

```
Coordinate Reference System:
```

```
  User input: Lambert_Conformal_Conic
  wkt:
PROJCRS["Lambert_Conformal_Conic",
    BASEGEOGCRS["NAD83",
        DATUM["North American Datum 1983",
            ELLIPSOID["GRS 1980",6378137,298.257222101,
                LENGTHUNIT["metre",1]],
            ID["EPSG",6269]],
        PRIMEM["Greenwich",0,
            ANGLEUNIT["Degree",0.0174532925199433]]],
    CONVERSION["unnamed",
        METHOD["Lambert Conic Conformal (2SP)",
            ID["EPSG",9802]],
        PARAMETER["Latitude of false origin",33.5,
            ANGLEUNIT["Degree",0.0174532925199433],
            ID["EPSG",8821]],
        PARAMETER["Longitude of false origin",-118,
            ANGLEUNIT["Degree",0.0174532925199433],
            ID["EPSG",8822]],
        PARAMETER["Latitude of 1st standard parallel",34.0333333333333,
            ANGLEUNIT["Degree",0.0174532925199433],
            ID["EPSG",8823]],
        PARAMETER["Latitude of 2nd standard parallel",35.4666666666667,
            ANGLEUNIT["Degree",0.0174532925199433],
            ID["EPSG",8824]],
        PARAMETER["Easting at false origin",6561666.66666667,
            LENGTHUNIT["US survey foot",0.304800609601219],
            ID["EPSG",8826]],
        PARAMETER["Northing at false origin",1640416.66666667,
            LENGTHUNIT["US survey foot",0.304800609601219],
            ID["EPSG",8827]]],
    CS[Cartesian,2],
        AXIS["(E)",east,
            ORDER[1],
            LENGTHUNIT["US survey foot",0.304800609601219,
                ID["EPSG",9003]]],
        AXIS["(N)",north,
            ORDER[2],
            LENGTHUNIT["US survey foot",0.304800609601219,
                ID["EPSG",9003]]]]
```

Furthermore, `st_intersects()` and most other geographic functions do not work, or do not work well, with latitude and longitude. We should really work with all of our spatial objects having the same projection. We can transform `mapLA` and `mapCens` to have the same coordinate system as our injunction area map, `mapSZ`, which uses a projection (LCC) different from latitude/longitude.

```
mapCens <- st_transform(mapCens, crs=st_crs(mapSZ))
mapLA   <- st_transform(mapLA,   crs=st_crs(mapSZ))
```

Now we can ask R to tell us for each census tract whether or not it intersections with the Los Angeles map. The result of `st_intersects()` is a list where `a[[1]]` will tell us which of the polygons on `mapLA` intersects with the first polygon in `mapCens`. Since `mapLA` only has one polygon, the `a[[1]]` will either be empty or 1. Therefore, to create an indicator of intersecting Los Angeles we just need to know whether the length of each element of `a` exceeds 0 (is not empty). We'll create a new column in the `mapCens` data containing a `TRUE`/`FALSE` indicator of whether that census tract is in Los Angeles or not.

```
a <- st_intersects(mapCens, mapLA)
# should equal the number of census tracts
length(a)
```

```
[1] 8057
```

```
mapCens$inLA <- lengths(a) > 0
```

Equivalently, we could have asked `st_intersects()` to create a list of census tracts that intersect with `mapLA`, by reversing `mapLA` and `mapCens` in `st_intersects()`.

```
a <- st_intersects(mapLA, mapCens)
# should equal 1, there's only one shape in mapLA
length(a)
```

```
[1] 1
```

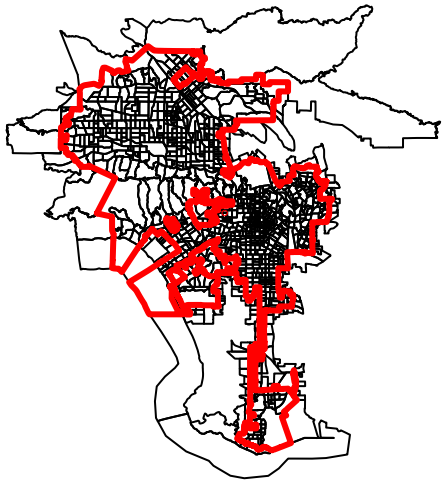
```
# show the indices of the first 10 census tracts that intersect with mapLA
a[[1]][1:10]
```

```
 [1]  1  2  3  4  5  6 17 18 19 20
```

```
# set inLA to FALSE for all, then replace those in a[[1]] with TRUE
mapCens$inLA <- FALSE
mapCens$inLA[a[[1]]] <- TRUE
```

Let's check that the census tracts with `TRUE` for `inLA` actually intersect Los Angeles.

```
mapCens <- subset(mapCens, inLA)
plot(st_geometry(mapCens))
plot(st_geometry(mapLA), add=TRUE, border="red", lwd=3)
```

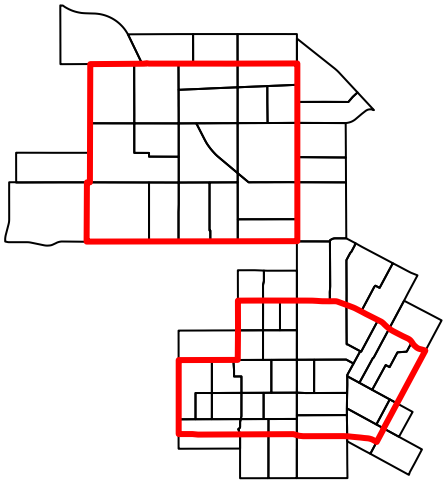


Which census tracts cover the MS13 safety zone?

```
st_intersects(mapSZms13, mapCens)
```

```
Sparse geometry binary predicate list of length 1, where the predicate
was 'intersects'
 1: 18, 95, 101, 102, 103, 110, 133, 141, 165, 199, ...
```

```
a <- st_intersects(mapSZms13, mapCens)
mapCens$inMS13 <- FALSE
mapCens$inMS13[a[[1]]] <- TRUE
plot(st_geometry(subset(mapCens, inMS13)))
plot(st_geometry(mapSZms13), border="red", lwd=3, add=TRUE)
```



Exercise

5. Census tracts that just touch the boundary of the safety zone are included. To eliminate, rather than use `st_intersects()` with `mapSZms13`, use `st_intersects()` with `st_buffer()` with a negative `dist` to select census tracts

Merge in demographic data from the American Community Survey

The full census of the United States occurs every ten years, but in between those surveys the Census Bureau collects data through the American Community Survey (ACS) by selecting a sample of households. These surveys have a lot of information about people and neighborhoods. We are just going to use the ACS to gather race data on the residents within census tracts.

JSON (JavaScript Object Notation) is a very common protocol for moving data. The ACS provides JSON access to its data. There are other ways of accessing ACS data, like downloading the entire ACS dataset, but we're going to use JSON so that you become familiar with how JSON works. Also, when we only need a small amount of information (just race data from particular census tracts) it can save a lot of effort when compared with downloading and processing the full ACS dataset.

First, let's load the `jsonlite` library.

```
library(jsonlite)
```

Here's how you can access the ACS data on the total population of the United States in 2019.

```
fromJSON("https://api.census.gov/data/2019/acs/acs5?get=NAME,B01001_001E&for=us:*")
```

```
     [,1]            [,2]          [,3]
[1,] "NAME"          "B01001_001E" "us"
[2,] "United States" "324697795"   "1"
```

Let's deconstruct this URL. First, we're accessing data from the 2019 ACS data using `http://api.census.gov/data/2019/acs/`. Second, we're using the date from the ACS sample collected over the last five years to estimate the total population... that's the `acs5` part. Third, we're accessing variable `B01001_001E`, which contains an estimate (the E at the end is for estimate) of the number of people in the United States. This we needed to track down, but the Social Explorer website, https://www.socialexplorer.com/data/ACS2019_5yr/metadata/, makes this easier. Lastly, we asked `for=us:*`, meaning for the entire United States.

If we want the total number of people in specific census tracts, then we can make this request.

```
fromJSON("https://api.census.gov/data/2019/acs/acs5?get=B01001_001E&for=tract:204920,205110&in=
```

```
     [,1]          [,2]    [,3]     [,4]
[1,] "B01001_001E" "state" "county" "tract"
[2,] "3904"        "06"    "037"    "205110"
[3,] "2751"        "06"    "037"    "204920"
```

Here we have requested population data (variable `B01001_001E`) for two specific census tracts (204920 and 205110) from California (state 06) in Los Angeles County (county 037).

If we want the total population in each tract in Los Angeles County, just change the tract list to an *.

```
a <- fromJSON("https://api.census.gov/data/2019/acs/acs5?get=B01001_001E&for=tract:*&in=state:0
# there are over 2000 census tracts in LA County. Show the first 10
a[1:10,]
```

```
      [,1]          [,2]    [,3]     [,4]
 [1,] "B01001_001E" "state" "county" "tract"
 [2,] "2373"        "06"    "037"    "482702"
 [3,] "7267"        "06"    "037"    "500201"
 [4,] "4988"        "06"    "037"    "500202"
 [5,] "2973"        "06"    "037"    "500300"
 [6,] "2703"        "06"    "037"    "500500"
 [7,] "6363"        "06"    "037"    "500900"
 [8,] "3669"        "06"    "037"    "501400"
 [9,] "2272"        "06"    "037"    "501501"
[10,] "3311"        "06"    "037"    "501802"
```

You can find a lot more examples at https://api.census.gov/data/2019/acs/acs5/examples.html.

Now let's get something more complete that we can merge into our geographic data. For each census tract in Los Angeles County, we will extract the total population (B03002001), the number of non-Hispanic white residents (B03002003), non-Hispanic black residents (B03002004), and Hispanic residents (B03002012).

```
dataRace <- fromJSON("https://api.census.gov/data/2019/acs/acs5?get=B03002_001E,B03002_003E,B0
```

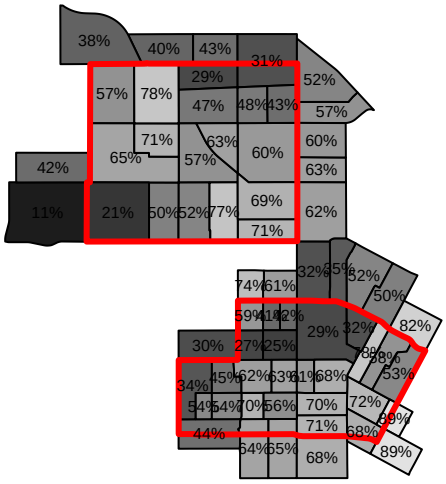
We will convert the matrix to a data frame, leaving out the first row of the matrix that has the column names.

```
a <- data.frame(dataRace[-1,])
names(a) <- dataRace[1,]
names(a)[1:4] <- c("total","white","black","hisp")
dataRace <- a
for(i in c("total","white","black","hisp"))
   dataRace[[i]] <- as.numeric(dataRace[[i]])
# compute number of residents of other race groups
dataRace$other <- with(dataRace,total-white-black-hisp)
dataRace[1:3,]
```

```
  total white black hisp state county  tract other
1  2373   178    27 1009    06    037 482702  1159
2  7267  4170    28 2211    06    037 500201   858
3  4988  1508    33 2281    06    037 500202  1166
```

Now we have a data frame that links the census tract numbers to populations and race data. Let's add race information to the MS13 injunction data.

```
# match tract IDs and merge in % hispanic
i <- match(mapCens$TRACTCE, dataRace$tract)
mapCens$pctHisp <- with(dataRace[i,],
   ifelse(total>0 & !is.na(hisp), hisp/total, 0))

# choose shade of gray depending on percent hispanic
col <- with(mapCens, gray(pctHisp[inMS13]))
plot(st_geometry(subset(mapCens,inMS13)), col=col)
plot(st_geometry(mapSZms13), border="red", lwd=3, add=TRUE)

# overlay with the percent hispanic
labs <- with(mapCens, paste0(round(100*pctHisp[inMS13]),"%"))
text(st_coordinates(st_centroid(subset(mapCens,inMS13))),
     labels=labs,
     cex=0.5)
```



Exercise

6. Create a map of all census tracts in the City of Los Angeles within 1 mile of one of the safety zones (you choose which safety zone)
7. Color each area based on a census feature (e.g. % non-white, or some other feature from the ACS data)
8. Add the polygon with your injunction zone
9. Add other injunction zones that intersect with your map

Working with point data using Los Angeles crime data

The Los Angeles Police Department (LAPD) posts all of its crime data at Los Angeles' open data portal. We're interested in just the 2010-2019 crime data held in the 2010-2019 crime data file.

There are several ways we could go about retrieving the data. One method is to ask R to download the data to our computer and then use `read.csv()` to import it. Conveniently, the Los Angeles open data portal allows "SoQL" queries, meaning that we can use SQL-like where clauses. By default, SoQL will limit the result to 1,000 rows, so I've modified the limit to 5 million, more than enough to get all the 2019 crime data.

```
# Method #1
download.file("https://data.lacity.org/resource/63jg-8b9z.csv?$where=date_extract_y(date_occ)=
              destfile = "11_shapefiles_and_data/LAPD crime data 2019.csv")
dataCrime <- read.csv("11_shapefiles_and_data/LAPD crime data 2019.csv",
                      as.is=TRUE)
```

Alternatively, we can just skip the download and ask R to directly read in the data from the Los Angeles data portal. The only downside to this is if you mess up your data, then you will need to download it all over again, which can be slow.

```
# Method #2
dataCrime <- read.csv("https://data.lacity.org/resource/63jg-8b9z.csv?$where=date_extract_y(da
                      as.is=TRUE)
```

Let's download all 10 years so that we can explore changes in crime over time.

```
download.file("https://data.lacity.org/resource/63jg-8b9z.csv?$limit=5000000",
              destfile = "11_shapefiles_and_data/LAPD crime data 2010-2019.csv")
dataCrime <- read.csv("11_shapefiles_and_data/LAPD crime data 2010-2019.csv",
                      as.is=TRUE)
```

Like always, let's peek at the first three rows to see what we have.

```
nrow(dataCrime)
dataCrime[1:3,]
```

```
[1] 2117816
     dr_no               date_rptd                 date_occ time_occ area
1  1307355 2010-02-20T00:00:00.000 2010-02-20T00:00:00.000     1350   13
2 11401303 2010-09-13T00:00:00.000 2010-09-12T00:00:00.000       45   14
3 70309629 2010-08-09T00:00:00.000 2010-08-09T00:00:00.000     1515   13
  area_name rpt_dist_no part_1_2 crm_cd
1    Newton        1385        2    900
2   Pacific        1485        2    740
3    Newton        1324        2    946
                                          crm_cd_desc        mocodes
1                            VIOLATION OF COURT ORDER 0913 1814 2000
2 VANDALISM - FELONY ($400 & OVER, ALL CHURCH VANDALISMS)           0329
3                           OTHER MISCELLANEOUS CRIME           0344
  vict_age vict_sex vict_descent premis_cd           premis_desc
1       48        M            H       501 SINGLE FAMILY DWELLING
2        0        M            W       101                STREET
3        0        M            H       103                 ALLEY
  weapon_used_cd weapon_desc status  status_desc crm_cd_1 crm_cd_2 crm_cd_3
1             NA                AA Adult Arrest      900       NA       NA
2             NA                IC  Invest Cont      740       NA       NA
```

```
3             NA                  IC  Invest Cont      946       NA       NA
  crm_cd_4                                  location
1       NA  300 E  GAGE                         AV
2       NA         SEPULVEDA                    BL
3       NA 1300 E  21ST                         ST
                     cross_street     lat       lon
1                                  33.9825 -118.2695
2 MANCHESTER                    AV 33.9599 -118.3962
3                                  34.0224 -118.2524
```

Now let's just keep the data that is not missing the latitude or longitude and convert our dataframe into a simple features spatial object.

```
dataCrime <- subset(dataCrime, !is.na(lat) & !is.na(lon))

dataCrime <- st_as_sf(dataCrime,
                      coords=c("lon","lat"),
                      crs=4326)
```

Setting `crs=4326` tells R that this spatial object has coordinates in latitude and longitude. Try to remember that EPSG 4326 refers to latitude and longitude.

Now we need to reproject the data into the coordinate system to match the injunction safety zone map.

```
dataCrime <- st_transform(dataCrime, st_crs(mapSZms13))
```

Let's now identify which crimes occurred within one mile of the MS13 injunction. We did some checking and noted that LAPD areas 1, 2, 3, 6, 7, 11, and 20 intersected with the MS13 safety zone, so we picked out just those crimes and plotted them each in different colors.

```
plot(st_geometry(st_buffer(mapSZms13, dist=5280)))
plot(st_geometry(mapSZms13), border="red", lwd=3, add=TRUE)
for(iArea in c(1,2,3,6,7,11,20))
   # unique() saves R from plotting duplicate points on top of each other
   plot(st_geometry(unique(subset(dataCrime, area==iArea, select=geometry))),
        col=iArea,
        pch=16,
        cex=0.5,
        add=TRUE)
```

A very useful operation is to find out which crimes occurred inside, near, or farther outside an area. We'll figure out which crimes occurred inside the safety zone, in a one-mile buffer around the safety zone, or more than a mile away from the safety zone. We'll subset to just those crimes that occurred in areas near the MS13 safety zone. This step is not essential, but it can save some computer time. There's no need for R to try to figure out if crimes in LAPD Area 4 fell inside the MS13 safety zone. All of those crimes are much more than a mile from the safety zone.

```
# just get those crimes in areas near the MS13 safety zone
dataCrimeMS13 <- subset(dataCrime, area %in% c(1,2,3,6,7,11,20))
```

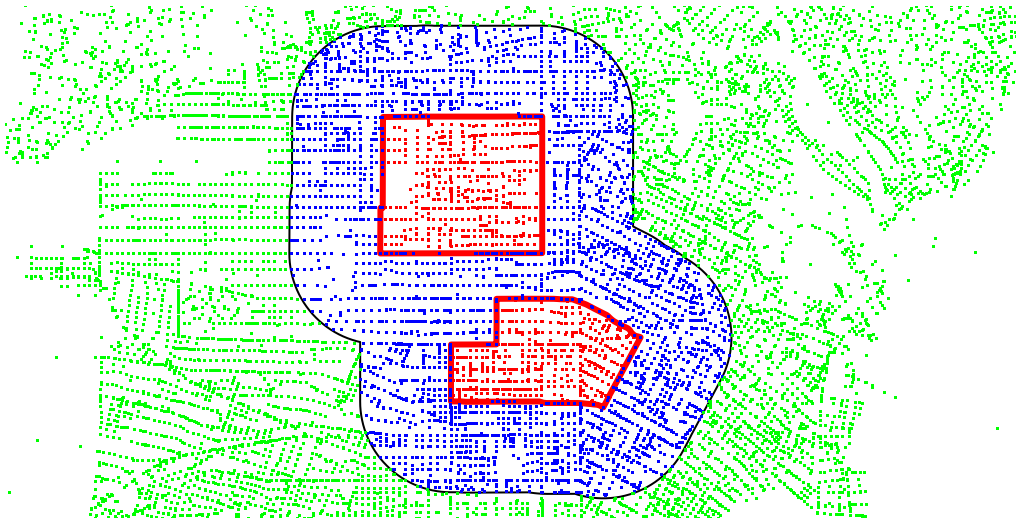
We're going to use two different methods so you learn about different ways of solving these problems. The first method will use the now familiar `st_intersects()` function. In our `datacrimeMS13` data frame we are going to make a new column that labels whether a crime is inside the injunction safety zone (SZ), within a one-mile buffer around the safety zone (buffer), or beyond the buffer (outside).

```
# create a variable to label the crime's location
dataCrimeMS13$place1 <- "outside"
i <- st_intersects(mapSZms13, dataCrimeMS13)[[1]]
dataCrimeMS13$place1[i] <- "SZ"
# can ignore warnings about attribute variables
i <- st_intersects(st_difference(st_buffer(mapSZms13, dist=5280),
                                 mapSZms13),
```

```
                    dataCrimeMS13)[[1]]
dataCrimeMS13$place1[i] <- "buffer"
```

Let's check that all the crimes are correctly labeled.

```
plot(st_geometry(st_buffer(mapSZms13, dist=5280)))
plot(st_geometry(mapSZms13), border="red", lwd=3, add=TRUE)
plot(st_geometry(subset(dataCrimeMS13, place1=="SZ")),
     pch=".", col="red", add=TRUE)
plot(st_geometry(subset(dataCrimeMS13, place1=="buffer")),
     pch=".", col="blue", add=TRUE)
plot(st_geometry(subset(dataCrimeMS13, place1=="outside")),
     pch=".", col="green", add=TRUE)
```



So using `st_intersects()` can correctly label the locations of different crimes. We'll also show you how to use `st_join()`, a spatial version of the joins that we did when studying SQL. First, we will make a new spatial object with three polygons, the MS13 safety zone, the buffer, and the region outside the buffer. We'll label those three polygons and use `st_join()` to ask each crime in which polygon they fall.

```
# combine the geometries of the three polygons
mapA <- c(st_geometry(mapSZms13),
```

```
          st_geometry(st_difference(st_buffer(mapSZms13, dist=5280),
                                    mapSZms13)),
          st_geometry(st_difference(st_buffer(mapSZms13, dist=80*5280),
                                    st_buffer(mapSZms13, dist=5280))))
# create an sf object
mapA <- st_sf(place2=c("SZ","buffer","outside"),
              geom=mapA)
plot(mapA)
```


```
dataCrimeMS13 <- st_join(dataCrimeMS13, mapA)
```

`st_join()` will add a new column `place2` to the `dataCrimeMS13` data frame containing the label of the polygon in which it landed.

```
dataCrimeMS13[1:3,]
```

```
Simple feature collection with 3 features and 28 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 6461916 ymin: 1836559 xmax: 6486290 ymax: 1859520
Projected CRS: Lambert_Conformal_Conic
```

```
      dr_no               date_rptd                date_occ time_occ area
4  90631215 2010-01-05T00:00:00.000 2010-01-05T00:00:00.000      150    6
5 100100501 2010-01-03T00:00:00.000 2010-01-02T00:00:00.000     2100    1
6 100100506 2010-01-05T00:00:00.000 2010-01-04T00:00:00.000     1650    1
  area_name rpt_dist_no part_1_2 crm_cd
4 Hollywood         646        2    900
5   Central         176        1    122
6   Central         162        1    442
                             crm_cd_desc        mocodes vict_age vict_sex
4                 VIOLATION OF COURT ORDER 1100 0400 1402       47        F
5                          RAPE, ATTEMPTED           0400       47        F
6 SHOPLIFTING - PETTY THEFT ($950 & UNDER)      0344 1402       23        M
  vict_descent premis_cd      premis_desc weapon_used_cd
4            W       101           STREET            102
5            H       103            ALLEY            400
6            B       404 DEPARTMENT STORE             NA
                                    weapon_desc status  status_desc crm_cd_1
4                                      HAND GUN     IC  Invest Cont      900
5 STRONG-ARM (HANDS, FIST, FEET OR BODILY FORCE)     IC  Invest Cont      122
6                                                    AA Adult Arrest      442
  crm_cd_2 crm_cd_3 crm_cd_4                              location
4      998       NA       NA        CAHUENGA                    BL
5       NA       NA       NA        8TH                         ST
6       NA       NA       NA 700 W  7TH                         ST
                     cross_street  place1  place2                geometry
4 HOLLYWOOD                    BL  buffer  buffer POINT (6461916 1859520)
5 SAN PEDRO                    ST outside outside POINT (6486290 1836559)
6                                 outside outside POINT (6483602 1839950)
```

And we can confirm that they produce the same results.

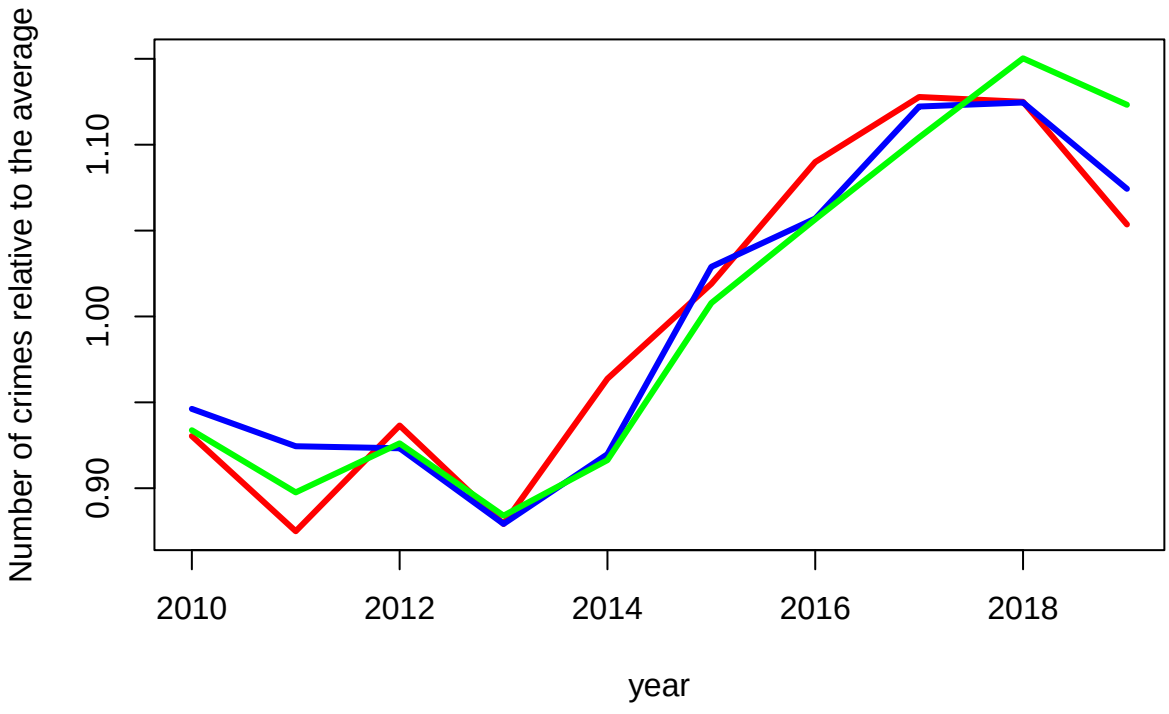
```
with(dataCrimeMS13, table(place1, place2))
```

```
         place2
place1    buffer outside     SZ
  buffer  204167       0      0
  outside      0  424617      0
  SZ           0       0  78163
```

Does crime behave differently inside the safety zone compared with the areas beyond the safety zone? let's break down the crime counts by year and plot them. We're going to divide the crime count by their average so that they are on the same scale. The area beyond the buffer is very large and it doesn't make sense to compare their counts directly.

```
# In date_occ, the first 10 characters hold the dates
dataCrimeMS13$date_occ <- ymd(substring(dataCrimeMS13$date_occ,1,10))
```

```r
# count the number of crimes by year and area
a <- aggregate(dr_no~place1+year(date_occ),
               data=dataCrimeMS13,
               length,
               drop=FALSE)
a <- reshape(a, timevar="place1", idvar="year(date_occ)", direction="wide")
names(a) <- c("year","buffer","outside","SZ")
# normalize to the average crime count over the period
a$SZ      <- a$SZ     /mean(a$SZ)
a$buffer  <- a$buffer /mean(a$buffer)
a$outside <- a$outside/mean(a$outside)
plot(SZ~year, data=a,
     type="l",
     col="red",
     lwd=3,
     ylim=range(a$buffer,a$outside,a$SZ),
     ylab="Number of crimes relative to the average")
lines(buffer~year, data=a, col="blue", lwd=3)
lines(outside~year, data=a, col="green", lwd=3)
```



Exercises

10. How many 2019 crimes occurred inside safety zones?
11. How many crimes per square mile inside safety zones? (Hint 1: use `st_area()` for area), Hint 2: use `st_intersects()` to see which fall inside `mapSZ`)
12. How many crimes per square mile outside the safety zone, but within 1 mile of a safety zone

Creating new geographic objects

Remember that the MS13 safety zone had a northern and southern component. We're going to work with just the southern component, but first we need to separate it from its northern component. We're going to show you several ways to accomplish this. The first will work directly with the `sf` objects and the second is an interactive method.

Right now, the MS13 map is stored as `MULTIPOLYGON` object.

```
is(st_geometry(mapSZms13))
```

```
[1] "sfc_MULTIPOLYGON" "sfc"              "oldClass"
```

A `MULTIPOLYGON` object is useful for managing a spatial object that involves several non-overlapping polygons, like this MS13 injunction, or the Hawaiian Islands, or the city of San Diego. We now want to break it apart into separate `POLYGON` objects using `st_cast()`.

```
a <- st_cast(mapSZms13, "POLYGON")
```

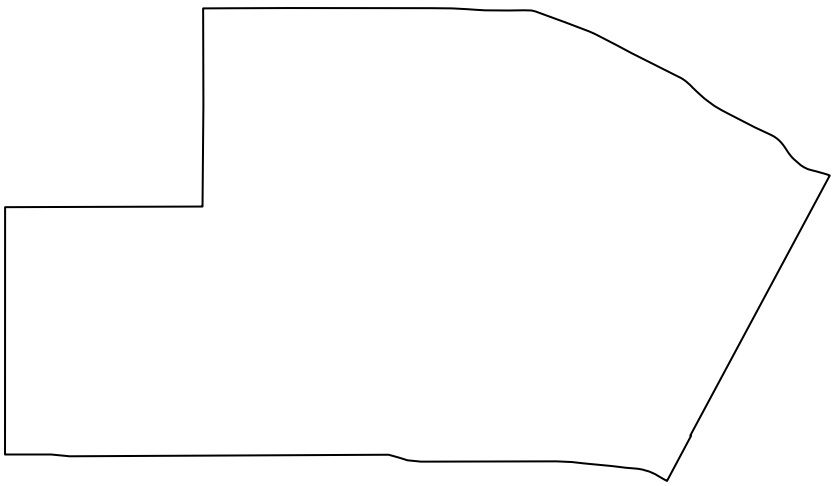
```
Warning in st_cast.sf(mapSZms13, "POLYGON"): repeating attributes for all sub-
geometries for which they may not be constant
```

```
a
```

```
Simple feature collection with 2 features and 5 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: 6463941 ymin: 1841325 xmax: 6479076 ymax: 1858250
Projected CRS: Lambert_Conformal_Conic
                      NAME  case_no                   Safety_Zn
1   Rampart/East Hollywood BC311766 Rampart/East Hollywood (MS)
1.1 Rampart/East Hollywood BC311766 Rampart/East Hollywood (MS)
           gang_name  startDate                       geometry
1   Mara Salvatrucha 2004-04-08 POLYGON ((6473357 1850297, ...
1.1 Mara Salvatrucha 2004-04-08 POLYGON ((6473614 1841583, ...
```

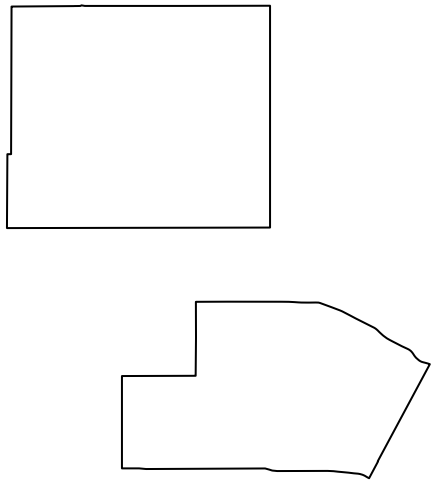
Now we can see that `a` has two distinct polygons. The second row corresponds to the southern polygon. Let's store that one separately.

```
mapSZms13s <- a[2,]
plot(st_geometry(mapSZms13s))
```

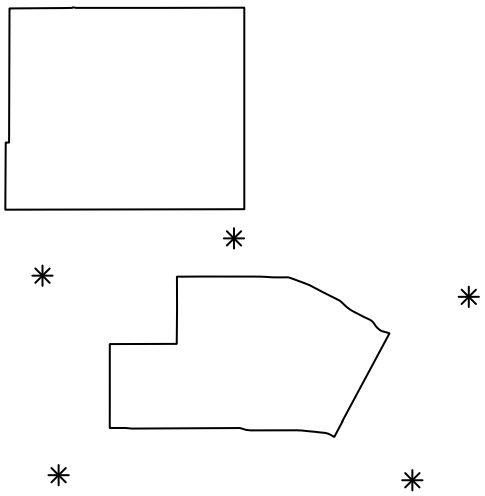


Using `st_cast()` is the most direct method. However, sometimes the shapes are more complicated and we might want to select the shape interactively. To interactively select the southern component, use the R function `locator()`. It allows you to click on an R plot and will return the coordinates of the points you've selected. Let's first get the plot of the full MS13 safety zone in the plot window.

```
plot(st_geometry(mapSZms13))
```



Next, run `boxXY <- locator()`. In the top left of the plot window you will see "Locator active (Esc to finish)". Then click several points around the southern MS13 polygon as if you are cutting out just the southern polygon. When you have finished clicking the points, press the Esc key on your keyboard. Here are the places that I clicked.



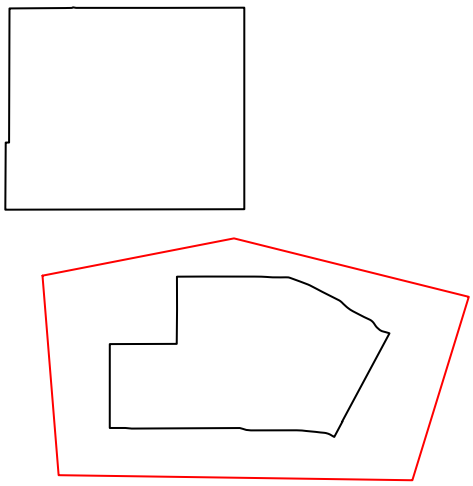
And our `boxXY` looks like this.

```
boxXY
```

```
$x
[1] 6465406 6472950 6482201 6479978 6466041

$y
[1] 1847679 1849148 1846845 1839619 1839818
```

Yours will almost certainly look different and may even have more elements. Check to make sure your box surrounds the southern safety zone.

```
# Make the end of the box reconnect back to the beginning
boxXY$x <- c(boxXY$x, boxXY$x[1])
boxXY$y <- c(boxXY$y, boxXY$y[1])
plot(st_geometry(mapSZms13),
     ylim=range(st_coordinates(st_geometry(mapSZms13))[,"Y"],
                boxXY$y))
lines(boxXY, col="red")
```



If you are not satisfied with your outline, just rerun `boxXY <- locator()` and rerun this plot to check your revised box.

We need to turn this collection of points defining our box into an `sf` object with which we can use GEOS functions. The first version we will use WKT (well known text), a way of using plain text to describe a geometric shape. This is a particularly useful method if your able to type out the specific shape that you want or need to copy a shape from another application that also uses the WKT format. You've probably already noticed a `geometry` variable in the `dataCrime` data frame that has elements that look like

```
st_as_text(st_geometry(dataCrime[1,]))
```

```
[1] "POINT (6479964 1816123)"
```

You can also make polygons using a `POLYGON` tag instead of a `POINT` tag. Here's what the first safety zone looks like in WKT format.

```
st_geometry(mapSZ)[[1]]
```

```
POLYGON ((6435396 1924413, 6435607 1924174, 6435792 1923960, 6435872 1923867, 6436158 1923545,
```

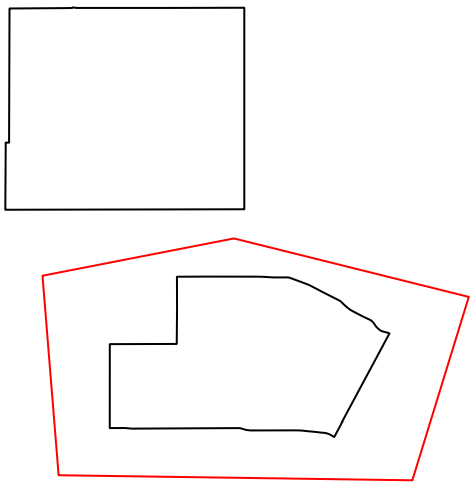
We can paste together the coordinates in `boxXY` to match this format.

```
boxTemp <- paste0("POLYGON((",
                  paste(paste(boxXY$x, boxXY$y), collapse=","),
                  "))")
boxTemp
```

```
[1] "POLYGON((6465406 1847679,6472950 1849148,6482201 1846845,6479978 1839619,6466041 1839818,6
```

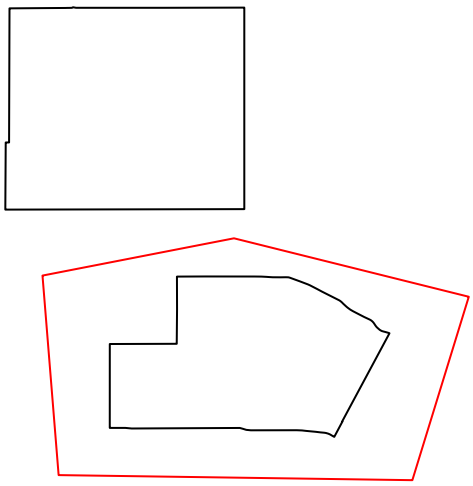
The text looks correct, so now we convert it to a simple features object, making sure to also tell R the coordinate system that we are using.

```
boxTemp <- st_as_sfc(boxTemp,
                     crs=st_crs(mapSZms13))
plot(st_geometry(mapSZms13),
     ylim=c(1839619,1858250))
plot(st_geometry(boxTemp), border="red", add=TRUE)
```



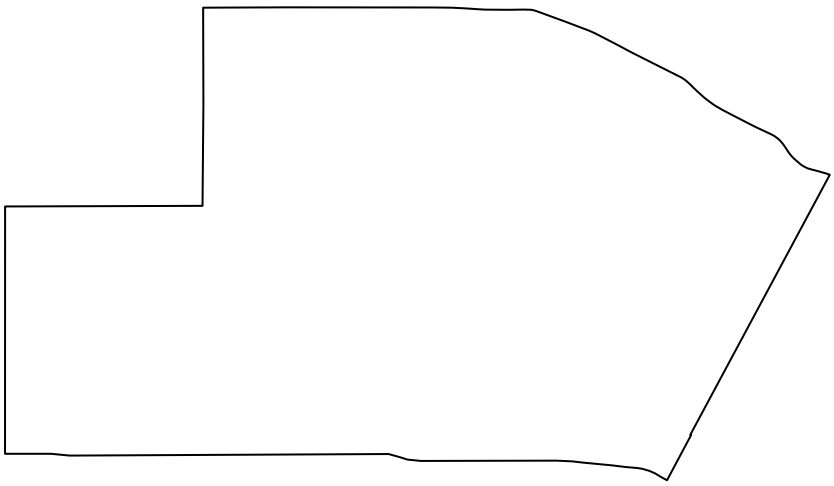
That was the WKT method. Let's try the `st_polygon()` method. `st_polygon()` takes in a matrix of coordinates and creates a simple features spatial object. Actually, it takes in a list of matrices. That way you can make objects like the northern and southern MS13 safety zones where the coordinates of each of the separate components are collected in one list.

```
boxTemp <- st_sfc(st_polygon(list(cbind(boxXY$x, boxXY$y))),
                  crs=st_crs(mapSZms13))
plot(st_geometry(mapSZms13),
     ylim=c(1839619,1858250))
plot(st_geometry(boxTemp), border="red", add=TRUE)
```



Now the only reason we did this process of creating `boxTemp` is so that we could select just the southern polygon, which is the intersection of our `boxTemp` and the original `mapSZms13`.

```
mapSZms13s <- st_intersection(st_geometry(mapSZms13), boxTemp)
plot(mapSZms13s)
```



With the southern MS13 safety zone extracted, let's explore the streets in this neighborhood.

Overlaying a street map

Let's load the street map for Los Angeles County, the county that contains the city of Los Angeles. The file we'll use here is `tl_2019_06037_roads.shp`. The naming convention says that this is a TIGER line file, from 2019, for state 06 (California), for county 037 (Los Angeles County), containing roads. Los Angeles County is large and this file has over 135,000 street segments. It can take a little while to load and project.

```
mapLAstreet <- st_read("11_shapefiles_and_data/tl_2019_06037_roads.shp")
# make sure we use the same projection as the injunction map
mapLAstreet <- st_transform(mapLAstreet, st_crs(mapSZms13s))
mapLAstreet[1:3,]
```

```
Reading layer 'tl_2019_06037_roads' from data source
  'Z:\Penn\CRIM602\notes\R4crim\11_shapefiles_and_data\tl_2019_06037_roads.shp'
  using driver 'ESRI Shapefile'
Simple feature collection with 135478 features and 4 fields
Geometry type: LINESTRING
Dimension:     XY
```

```
Bounding box:  xmin: -118.9445 ymin: 32.80628 xmax: -117.6497 ymax: 34.8233
Geodetic CRS:  NAD83
Simple feature collection with 3 features and 4 fields
Geometry type: LINESTRING
Dimension:     XY
Bounding box:  xmin: 6487582 ymin: 1835878 xmax: 6510389 ymax: 1857279
Projected CRS: Lambert_Conformal_Conic
       LINEARID             FULLNAME RTTYP MTFCC                        geometry
1 1101576755652 Golden State Fwy Rmp     M S1400 LINESTRING (6487884 1856781...
2 1101576692583       Pomona Fwy Rmp     M S1400 LINESTRING (6510389 1835878...
3 1101576663753          Soto St Rmp     M S1400 LINESTRING (6501699 1845173...
```

The file contains the geometry of each road (`geometry`), the name of the road (`FULLNAME`), and the type of road (`RTTYP` and `MTFCC`). `RTTYP` stands for "route type code" where

- M: Common (municipal) street
- C: County road
- S: State road (e.g. Highway 1, Route 66)
- I: Interstate highway (e.g. I-5, I-405, I-10)
- U: U.S. highway (U.S. 101)
- O: Other (e.g. forest roads, utility service roads)

`MTFCC` stands for "MAF/TIGER Feature Class Code". There are numerous MTFCC one for just about every geographical feature you can think of (e.g shorelines, water towers, campgrounds), but some of the common ones for our purposes here are

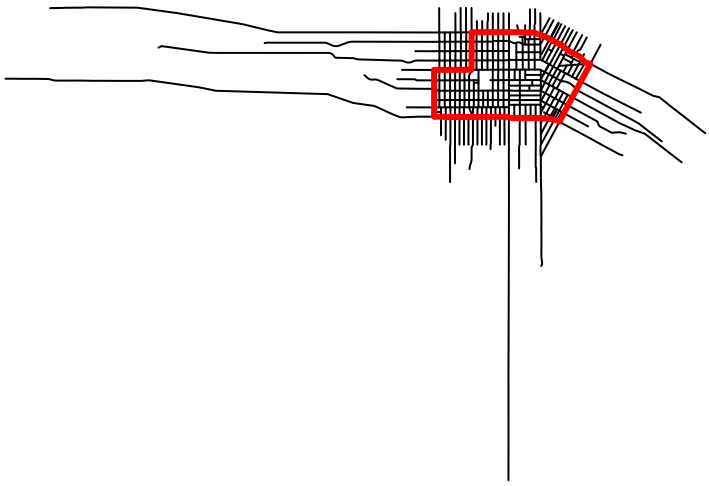
- s1100: Primary road (limited access highway)
- s1200: Secondary road (main arteries and smaller highways)
- s1400: Local neighborhood road
- s1730: Alley
- s1780: Parking lot

We don't need all the streets of Los Angeles County, so let's just get the ones that intersect without southern MS13 safety zone.

```
mapLAstreet$inSZ <- FALSE
i <- st_intersects(mapSZms13s, mapLAstreet)[[1]]
mapLAstreet$inSZ[i] <- TRUE
mapMS13street <- subset(mapLAstreet, inSZ)
```

Let's take a look at the streets.

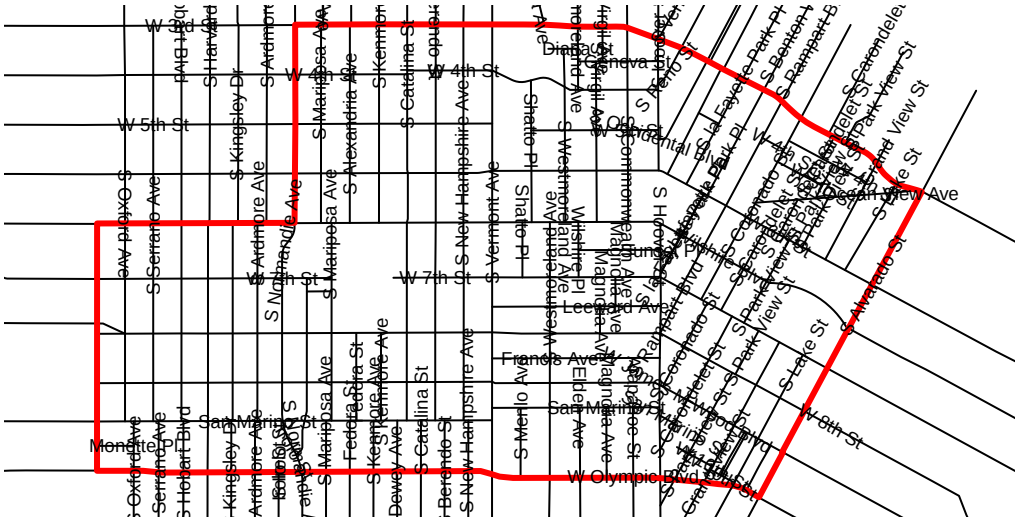
```
plot(st_geometry(mapMS13street))
plot(st_geometry(mapSZms13s), border="red", lwd=3, add=TRUE)
```



Now we have our safety zone and the streets that run through the safety zone. We would like to zoom and put some street names over the map so we can read it more like a street map. It is hard to do this perfectly, but the following function will work for our purposes. For each street segment we want to select a point on the street, near or inside the safety zone where we will put the label. Some streets run north/south, others east/west, and others diagonally. So, we need to figure out an angle for the label too.

```
# extract the coordinates for every street segment
a <- lapply(st_geometry(mapMS13street), st_coordinates)

# for each street segment get (x,y,angle)
labs <- sapply(a, function(coord)
   {
      # which parts of the street are inside MS13 safety zone
      i <- which((coord[,"X"] > st_bbox(mapSZms13s)["xmin"]) &
                 (coord[,"X"] < st_bbox(mapSZms13s)["xmax"]) &
                 (coord[,"Y"] > st_bbox(mapSZms13s)["ymin"]) &
                 (coord[,"Y"] < st_bbox(mapSZms13s)["ymax"]))
      # don't select the last one, too close to the edge
      i <- setdiff(i, nrow(coord))
      # if none are in bounding box just use the first coordinate
      if(length(i)==0) i <- 1
      # randomly choose a point on the street for the label
```

```
      i <- sample(i, size=1)
      # compute the slope  of the street, change in y/change in x
      streetSlope <- (coord[i+1,2]-coord[i,2]) / (coord[i+1,1]-coord[i,1])
      # compute the angle of the slope with the arc-tangent
      angle <- atan(streetSlope)
      # atan() returns radians, convert to degrees
      angle <- 180*angle/pi
      # round to the nearest 10
      angle <- round(angle, -1)
      # would rather not have labels that are upside down
      angle <- ifelse(angle < -90,  180+angle, angle)
      angle <- ifelse(angle >  90, -180+angle, angle)

      return(c(x=coord[i,1], y=coord[i,2], angle=angle))
   })
# transpose results and make a data frame
labs <- data.frame(t(labs))

plot(st_geometry(mapSZms13s), border="red", lwd=1)
plot(st_geometry(mapMS13street), add=TRUE)
plot(st_geometry(mapSZms13s), border="red", lwd=3, add=TRUE)

#add street names to map
for(i in 1:nrow(labs))
{
   text(labs$x[i], labs$y[i],
        mapMS13street$FULLNAME[i],
        srt=labs$angle[i],          # srt = string rotation
        cex=0.6)                    # cex = character expansion
}
```



Wilshire Blvd is a major street that runs from the Pacific Ocean to downtown Los Angeles running through the MS13 safety zone along the way. You can see it highlighted in green here.

```
plot(st_geometry(mapSZms13s), border="red", lwd=1)
plot(st_geometry(mapMS13street), add=TRUE)
plot(st_geometry(mapSZms13s), border="red", lwd=3, add=TRUE)
for(i in 1:nrow(labs))
{
   text(labs$x[i], labs$y[i],
        mapMS13street$FULLNAME[i],
        srt=labs$angle[i],          # srt = string rotation
        cex=0.6)                    # cex = character expansion
}
mapWilshire <- subset(mapMS13street, FULLNAME=="Wilshire Blvd")
plot(st_geometry(mapWilshire), col="green", lwd=3, add=TRUE)
```

We are going to count how many crimes occurred within 100 feet of Wilshire Blvd. Note that it is unlikely that any crimes will have occurred exactly on top of the line that is representing Wilshire Blvd in the map. We will work through two different methods. The first method will use a 100-foot buffer and count the crimes that land in it. The second method will compute the distance each crime is to Wilshire Blvd.

```
# create a 100-foot buffer, but only the part that is in the ms13 safety zone
mapWilbuffer <- st_intersection(st_geometry(st_buffer(mapWilshire, dist=100)),
                                st_geometry(mapSZms13s))
i <- st_intersects(mapWilbuffer, dataCrimeMS13)[[1]]
dataCrimeMS13$inWilbuf <- FALSE
dataCrimeMS13$inWilbuf[i] <- TRUE

plot(st_geometry(mapSZms13s), border="red", lwd=1)
plot(st_geometry(mapWilbuffer), add=TRUE, border="green")
plot(st_geometry(mapMS13street), add=TRUE)
plot(st_geometry(mapSZms13s), border="red", lwd=3, add=TRUE)
for(i in 1:nrow(labs))
{
   text(labs$x[i], labs$y[i],
        mapMS13street$FULLNAME[i],
        srt=labs$angle[i],          # srt = string rotation
        cex=0.6)                    # cex = character expansion
```

```
}
plot(st_geometry(subset(dataCrimeMS13, inWilbuf)),
     col="blue", add=TRUE, pch=16, cex=0.5)
```

And what are the most common crime types in this area?

```
with(dataCrimeMS13, rev(sort(table(crm_cd_desc[inWilbuf])))[1:5])
```

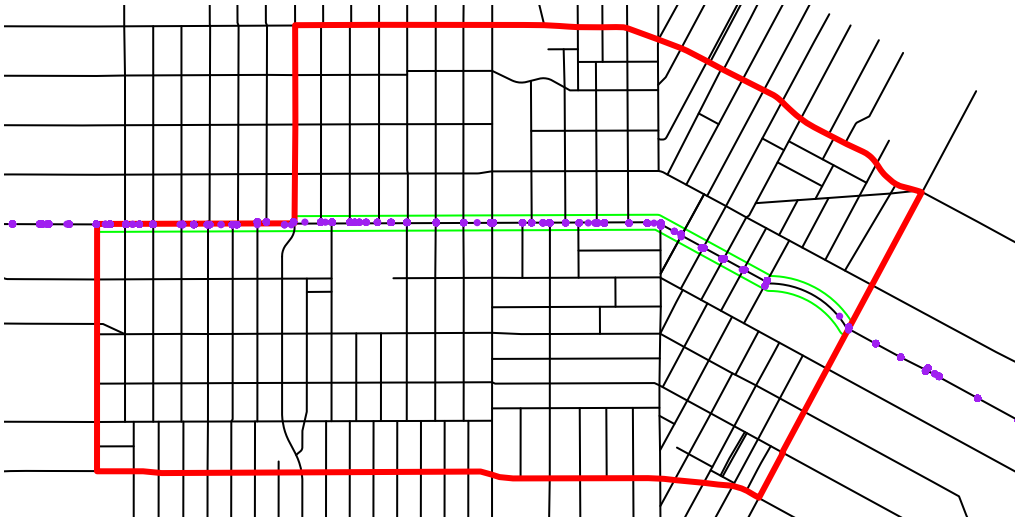
```
                        BATTERY - SIMPLE ASSAULT
                                             823
              THEFT PLAIN - PETTY ($950 & UNDER)
                                             807
                       BURGLARY FROM VEHICLE
                                             465
                                        ROBBERY
                                             428
ASSAULT WITH DEADLY WEAPON, AGGRAVATED ASSAULT
                                             330
```

In the previous method we created a 100-foot buffer and then asked which crimes landed inside the buffer. Alternatively, we can compute the distance between each crime point location and Wilshire Blvd. This second method takes a lot more computational effort and will be much slower, but we want you to be familiar with the functions that compute distances.

```
d <- st_distance(dataCrimeMS13, mapWilshire)
dim(d) # n rows, 1 column
```

```
[1] 707235      1
```

```
plot(st_geometry(mapSZms13s), border="red", lwd=1)
plot(st_geometry(mapWilbuffer), add=TRUE, border="green")
plot(st_geometry(mapMS13street), add=TRUE)
plot(st_geometry(mapSZms13s), border="red", lwd=3, add=TRUE)
plot(st_geometry(dataCrimeMS13[as.numeric(d[,1])<100,]),
     col="purple", add=TRUE, pch=16, cex=0.5)
```



Exercise

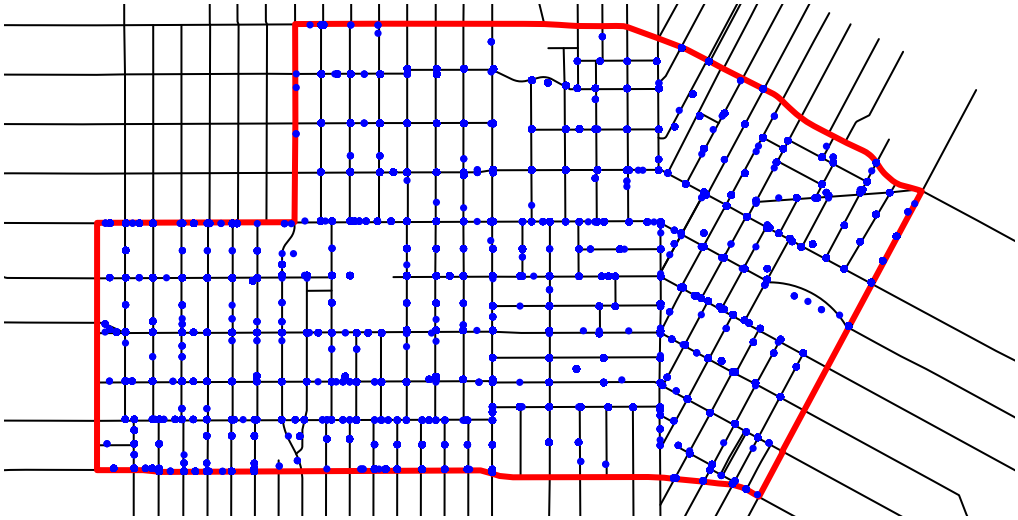
13. Are there more crimes along Wilshire Blvd or S Vermont Ave?

Find which line is closest to a point

We're going to find out which street is closest to each point. Yes, the crimes already have an address associated with them, but we'll use that to check our work.

First, let's subset our crime data so we just have crimes that fall into the MS13 southern safety zone.

```
i <- st_intersects(mapSZms13s, dataCrimeMS13)[[1]]
dataCrimeMS13s <- dataCrimeMS13[i,]
plot(st_geometry(mapSZms13s), border="red", lwd=1)
plot(st_geometry(mapMS13street), add=TRUE)
plot(st_geometry(mapSZms13s), border="red", lwd=3, add=TRUE)
plot(st_geometry(dataCrimeMS13s),
     add=TRUE, col="blue",pch=16, cex=0.5)
```



Now let's compute the distance for each point to the closest street in `mapMS13street`.

```
d <- st_distance(dataCrimeMS13s, mapMS13street)
dim(d) # row for each crime, column for each street
```

```
[1] 44728   116
```

`d` is a matrix of distances with 44728 rows and 116 columns, a distance from every crime point to every street. Now let's figure out which street is closest.

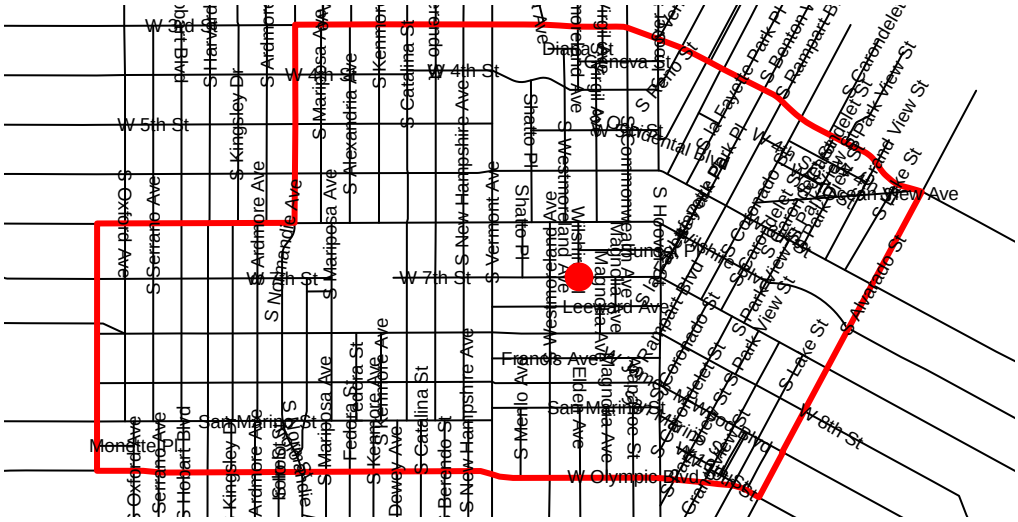
```
# for each row (crime) find out which column (street)
iClose <- apply(d, 1, which.min)

# for the first crime check that the original address is similar to closest street
dataCrimeMS13s[1,]
```

```
Simple feature collection with 1 feature and 29 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 6474497 ymin: 1844269 xmax: 6474497 ymax: 1844269
Projected CRS: Lambert_Conformal_Conic
         dr_no               date_rptd   date_occ time_occ area area_name
2161 100109513 2010-04-17T00:00:00.000 2010-04-17     1300    1   Central
     rpt_dist_no part_1_2 crm_cd                       crm_cd_desc   mocodes
2161         153        2    626 INTIMATE PARTNER - SIMPLE ASSAULT 0400 2000
     vict_age vict_sex vict_descent premis_cd premis_desc weapon_used_cd
2161       47        F            B       102    SIDEWALK            400
                                    weapon_desc status status_desc crm_cd_1
2161 STRONG-ARM (HANDS, FIST, FEET OR BODILY FORCE)     IC Invest Cont      626
     crm_cd_2 crm_cd_3 crm_cd_4 location cross_street place1 place2
2161       NA       NA       NA      7TH     WILSHIRE     SZ     SZ
                    geometry inWilbuf
2161 POINT (6474497 1844269)    FALSE
```

```
mapMS13street[iClose[1],]
```

```
Simple feature collection with 1 feature and 5 fields
Geometry type: LINESTRING
Dimension:     XY
Bounding box:  xmin: 6472016 ymin: 1838460 xmax: 6485519 ymax: 1844270
Projected CRS: Lambert_Conformal_Conic
           LINEARID FULLNAME RTTYP MTFCC                        geometry inSZ
17580 1106080860474 W 7th St     M S1400 LINESTRING (6485519 1838460... TRUE
```

```
plot(st_geometry(mapSZms13s), border="red", lwd=1)
plot(st_geometry(mapMS13street), add=TRUE)
plot(st_geometry(mapSZms13s), border="red", lwd=3, add=TRUE)
for(i in 1:nrow(labs))
{
   text(labs$x[i], labs$y[i],
        mapMS13street$FULLNAME[i],
        srt=labs$angle[i],
        cex=0.6)
}
plot(st_geometry(dataCrimeMS13s[1,]),
     add=TRUE, col="red", pch=16, cex=2)
```



Which streets have the most incidents?

```
# using distance calculation
a <- table(mapMS13street$FULLNAME[iClose])
rev(sort(a))[1:10]
```

```
          W 6th St       Wilshire Blvd             W 7th St              W 4th St
              6831                4088                 3237                 2161
          W 8th St            W 5th St W James M Wood Blvd         S Hoover St
              1908                1759                 1561                 1514
      S Catalina St       San Marino St
              1505                1342
```

```
# or,  just using the addresses
a <- gsub("^[0-9]+ ", "", dataCrimeMS13s$location)
a <- gsub(" * ", " ", a)
rev(sort(table(a)))[1:10]
```

```
a
  WILSHIRE BL      W 6TH ST      W 8TH ST S ALVARADO ST S CATALINA ST
         4275          1918          1770          1256          1177
 S BERENDO ST  S VERMONT AV        6TH ST  S KENMORE AV  S RAMPART BL
         1113          1035          1024           982           907
```

Exercise

14. There are LA Metrorail stations along Wilshire at Western Ave (farthest west), S Normandie Ave, S Vermont Ave, and Alvarado (farthest east). How many crimes occurred within 500ft of a Metrorail station?

Hint: Consider finding the stations using `st_intersection()`.

```
mapMetro <- st_intersection(subset(mapLAstreet, FULLNAME %in%
                                c("S Western Ave","S Normandie Ave",
                                  "S Vermont Ave","S Alvarado St")),
                        subset(mapLAstreet, FULLNAME=="Wilshire Blvd"))
```

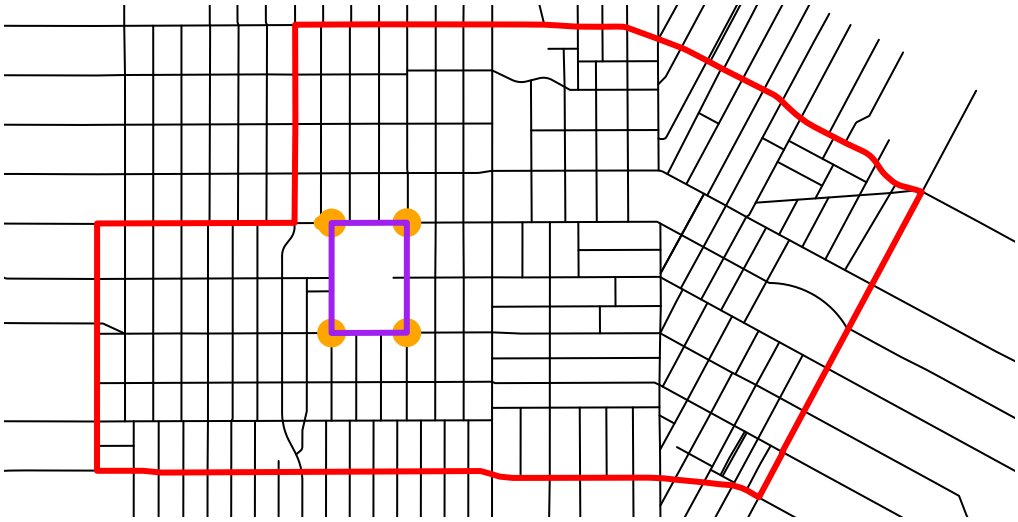
15. RFK Community Schools occupies the site between Mariposa and Catalina and W 8th St and Wilshire Blvd (former site of the Ambassador Hotel). How many crimes occurred within 500ft of the RFK School?

Hints:

```
a <- st_intersection(subset(mapLAstreet,
                                FULLNAME %in% c("S Mariposa Ave","S Catalina St")),
                        subset(mapLAstreet,
                                FULLNAME %in% c("W 8th St","Wilshire Blvd")))

plot(st_geometry(mapSZms13s), border="red", lwd=1)
plot(st_geometry(mapMS13street), add=TRUE)
plot(st_geometry(mapSZms13s), border="red", lwd=3, add=TRUE)

# plot the intersection points
plot(st_geometry(a), col="orange", add=TRUE, pch=16, cex=1)
# drop the more western Mariposa/Wilshire intersection
which.min(st_coordinates(a)[,1])
```

```
5
5
```

```
a <- a[-which.min(st_coordinates(a)[,1]),]
plot(st_geometry(a), col="orange", add=TRUE, pch=16, cex=2)

# compute the convex hull of the four points
mapRFKschool <- st_convex_hull(st_union(a))
plot(st_geometry(mapRFKschool), border="purple", add=TRUE, lwd=3)
```



Make a KML file to post to Google Maps

KML (keyhole markup language) is a standard way that Google Maps stores geographic information (Keyhole was a company that Google acquired, renaming their product Google Earth). You can convert any of the maps you make in R to KML format and post them to Google Maps.

```
a <- st_transform(mapSZms13, crs=4326)
st_write(a,
         dsn="ms13.kml",
         layer= "ms13",
         driver="KML",
         delete_dsn = TRUE)
```

```
Deleting source ‘ms13.kml’ using driver ‘KML’
Writing layer ‘ms13’ to data source ‘ms13.kml’ using driver ‘KML’
Writing 1 features with 5 fields and geometry type Multi Polygon.
```

Now you can navigate to http://www.google.com/mymaps, click import, select your `ms13.kml` file, and then it will be visible as an overlay on top of the usual Google map of Los Angeles.

Solutions to the exercises

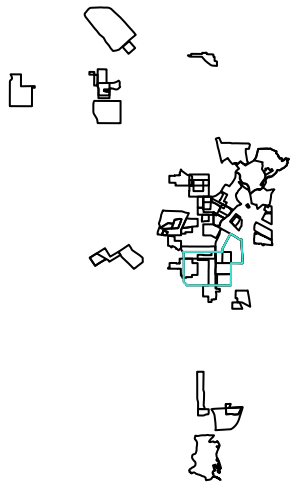
1. Find the largest and smallest safety zones (use `st_area(mapSZ)`)

```
iMin <- which.min(st_area(mapSZ))
iMax <- which.max(st_area(mapSZ))
c(iMin, iMax)
```

```
[1] 17 51
```

2. Plot all the safety zones. Color the largest in one color and the smallest in another color

```
plot(st_geometry(mapSZ))
plot(st_geometry(mapSZ[iMin,]), add=TRUE, col="salmon")
plot(st_geometry(mapSZ[iMax,]), add=TRUE, border="turquoise")
```



The smallest one is very tiny, just to the northeast of the biggest one.

3. Use `st_overlaps(mapSZ, mapSZ, sparse=FALSE)` or `print(st_overlaps(mapSZ, mapSZ, sparse=TRUE), n=Inf, max_nb=Inf)` to find two safety zones that overlap (not just touch at the edges)

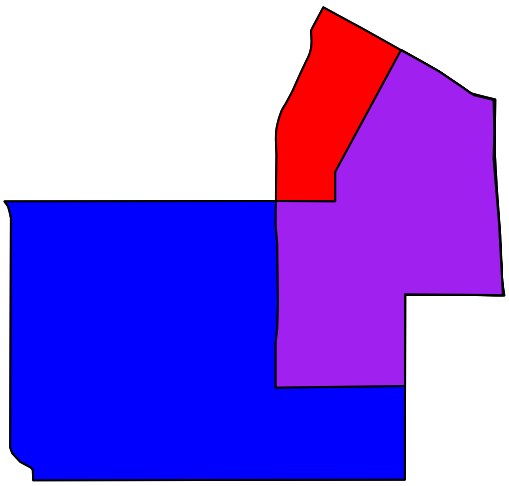
```
print(st_overlaps(mapSZ, mapSZ, sparse=TRUE), n=Inf, max_nb=Inf)
```

```
Sparse geometry binary predicate list of length 65, where the predicate
was 'overlaps'
 1: (empty)
 2: 57
 3: (empty)
 4: 57
 5: 57
 6: 7, 8
 7: 6, 13, 49
 8: 6
 9: 10, 12
 10: 9, 11, 12, 65
 11: 10, 65
 12: 9, 10
 13: 7
 14: 15, 16, 20, 22, 52, 59, 60, 61, 62
 15: 14, 61
 16: 14, 20, 22, 52, 60, 61, 62
 17: (empty)
 18: (empty)
 19: 63
 20: 14, 16, 22, 60
 21: (empty)
 22: 14, 16, 20, 60
 23: (empty)
 24: (empty)
 25: (empty)
 26: (empty)
 27: (empty)
 28: 60
 29: (empty)
 30: 33, 37, 38, 51, 53
 31: 63
 32: (empty)
 33: 30, 37, 51
 34: 46
 35: 36
 36: 35, 39
 37: 30, 33
 38: 30, 46, 51, 53
 39: 36
 40: 46, 51
 41: (empty)
 42: (empty)
 43: 54, 64
```

```
 44: (empty)
 45: (empty)
 46: 34, 38, 40, 51
 47: 56
 48: (empty)
 49: 7
 50: (empty)
 51: 30, 33, 38, 40, 46, 53
 52: 14, 16, 59, 62
 53: 30, 38, 51
 54: 43, 64
 55: (empty)
 56: 47
 57: 2, 4, 5
 58: (empty)
 59: 14, 52, 62
 60: 14, 16, 20, 22, 28
 61: 14, 15, 16
 62: 14, 16, 52, 59
 63: 19, 31
 64: 43, 54
 65: 10, 11
```

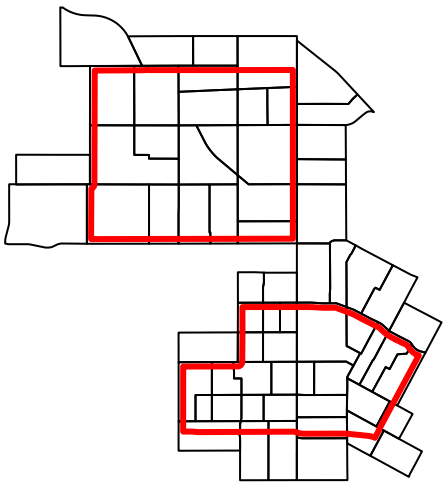
4. With the two safety zones that you found in the previous question, plot using three different colors the first safety zone, second safety zone, and their intersection (hint: use `st_intersection()`)

```
plot(st_geometry(mapSZ[c(30,51),]))
plot(st_difference(st_geometry(mapSZ[30,]),
                   st_geometry(mapSZ[51,])),
     col="red",
     add=TRUE)
plot(st_difference(st_geometry(mapSZ[51,]),
                   st_geometry(mapSZ[30,])),
     col="blue",
     add=TRUE)
plot(st_intersection(st_geometry(mapSZ[51,]),
                     st_geometry(mapSZ[30,])),
     col="purple",
     add=TRUE)
```

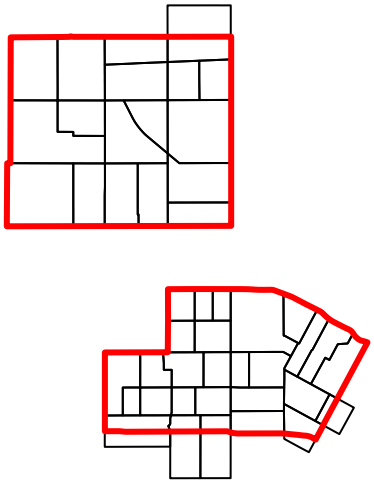


5. Census tracts that just touch the boundary of the safety zone are included. To eliminate, rather than use `st_intersects()` with `mapSZms13`, use `st_intersects()` with `st_buffer()` with a negative `dist` to select census tracts

```
plot(st_geometry(subset(mapCens, inMS13)))
plot(st_geometry(st_buffer(mapSZms13, dist = -200)),
     border="red",
     lwd=3,
     add=TRUE)
```



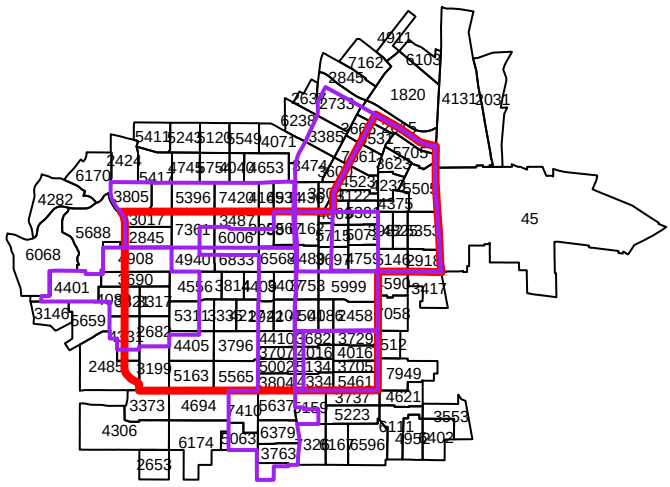
```
i <- st_intersects(st_buffer(mapSZms13, dist = -200),
                   mapCens)[[1]]
mapCens$inMS13 <- FALSE
mapCens$inMS13[i] <- TRUE
plot(st_geometry(subset(mapCens, inMS13)))
plot(st_geometry(mapSZms13), border="red", lwd=3, add=TRUE)
```



6. Create a map of all census tracts in the City of Los Angeles within 1 mile of one of the safety zones (you choose which safety zone)
7. Color each area based on a census feature (e.g. % non-white, or some other feature from the ACS data)
8. Add the polygon with your injunction zone
9. Add other injunction zones that intersect with your map

```
# find census tracts within 1 mile of SZ #51
iCens <- st_intersects(st_buffer(mapSZ[51,], dist=5280),
                  mapCens)[[1]]
plot(st_geometry(mapCens[iCens,]))

# get total population for each census tract
dataRes <- fromJSON("https://api.census.gov/data/2019/acs/acs5?get=B01001_001E&for=tract:*&in=
dataRes <- data.frame(dataRes[-1,])
names(dataRes) <- c("pop","state","county","tract")
dataRes$pop <- as.numeric(dataRes$pop)

# merge population with census tract data
i <- match(mapCens$TRACTCE, dataRes$tract)
mapCens$pop <- dataRes$pop[i]
```

```
labs <- mapCens$pop[iCens]
text(st_coordinates(st_centroid(mapCens[iCens,])), labels=labs, cex=0.5)
plot(st_geometry(mapSZ[51,]), add=TRUE, border="red", lwd=4)

# overlay the other intersecting injunctions
i <- st_intersects(mapSZ[51,], mapSZ)[[1]]
# don't include SZ #51 itself in the collection
i <- setdiff(i, 51)
plot(st_geometry(mapSZ[i,]), add=TRUE, border="purple", lwd=2)
```



10. How many 2019 crimes occurred inside safety zones?

```
i <- st_intersects(st_union(mapSZ),
                   subset(dataCrime, year(date_occ)==2019))[[1]]
nSZcrimes <- length(i)
```

11. How many crimes per square mile inside safety zones? (Hint 1: use `st_area()` for area), Hint 2: use `st_intersects()` to see which fall inside `mapSZ`)

```
library(units)
```

```
udunits database from C:/Users/greg_/OneDrive/Documents/R/win-library/4.1/units/share/udunits/u
```

```
# use set_units() to convert to square miles
#   equivalent to multiplying by 5820^2
set_units(nSZcrimes / st_area(st_union(mapSZ)), value="1/mile^2")
```

```
907.6967 [1/mile^2]
```

12. How many crimes per square mile outside the safety zone, but within 1 mile of a safety zone

```
mapBuf <- st_difference(st_buffer(st_union(mapSZ), dist=5280),
                        st_union(mapSZ))
mapBuf <- st_intersection(mapBuf, mapLA)

i <- st_intersects(mapBuf,
                   subset(dataCrime, year(date_occ)==2019))[[1]]
set_units(length(i) / st_area(mapBuf), value="1/mile^2")
```

```
473.4945 [1/mile^2]
```

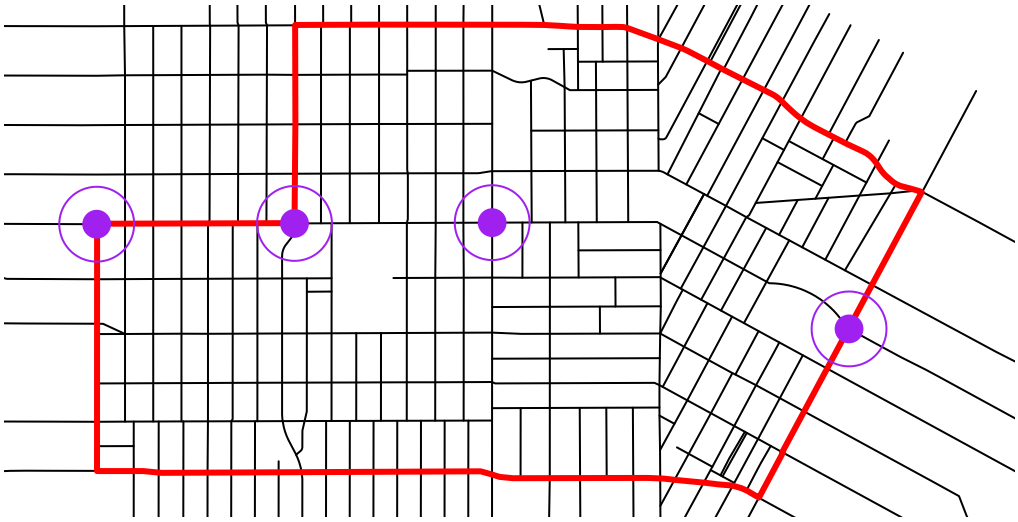
13. Are there more crimes along Wilshire Blvd or S Vermont Ave?

```
i <- st_intersects(st_intersection(st_buffer(subset(mapMS13street,
                                                    FULLNAME=="S Vermont Ave"),
                                             dist=100),
                                   mapSZms13s),
                   dataCrime)[[1]]
length(i)
i <- st_intersects(st_intersection(st_buffer(subset(mapMS13street,
                                                    FULLNAME=="Wilshire Blvd"),
                                             dist=100),
                                   mapSZms13s),
                   dataCrime)[[1]]
length(i)
```

```
[1] 3113
[1] 6059
```

14. There are LA Metrorail stations along Wilshire at Western Ave (farthest west), S Normandie Ave, S Vermont Ave, and Alvarado (farthest east). How many crimes occurred within 500ft of a Metrorail station?

```
mapMetro <- st_intersection(subset(mapLAstreet, FULLNAME %in%
                                c("S Western Ave","S Normandie Ave",
                                  "S Vermont Ave","S Alvarado St")),
                    subset(mapLAstreet, FULLNAME=="Wilshire Blvd"))
plot(st_geometry(mapSZms13s), border="red", lwd=1)
plot(st_geometry(mapMS13street), add=TRUE)
plot(st_geometry(mapSZms13s), border="red", lwd=3, add=TRUE)
plot(st_geometry(mapMetro), col="purple", add=TRUE, pch=16, cex=2)

plot(st_geometry(st_buffer(mapMetro, dist=500)),
     add=TRUE, border="purple")
```



```
i <- st_intersects(st_buffer(mapMetro, dist=500), dataCrime)
# how many crimes near each station?
sapply(i , length)
# how many crimes overall?
length(unlist(i))
```

```
[1]  963  539  839 1063
[1] 3404
```

15. RFK Community Schools occupies the site between Mariposa and Catalina and W 8th St

and Wilshire Blvd (former site of the Ambassador Hotel). How many crimes occurred within 500ft of the RFK School?

```
length(st_intersects(st_buffer(mapRFKschool, dist=500),
                     dataCrimeMS13s)[[1]])
```

```
[1] 4518
```