Calculating Zip-Level Access Metrics: Minimum Distance from Zip Centroid to Resource

Angela Li

4/7/2020

Tutorial objectives

This tutorial demonstrates how to calculate a minimum distance value from a zip code centroid to a set of resources, such as locations of methadone clinics. Each zip code will be assigned a "minimum distance access metric" as a value that indicates access to resources from that zip code. This tutorial assumes some familiarity with the R programming language, but introduces spatial concepts to those who have not previously worked with spatial data in R. Please contact Angela Li at ali6@uchicago.edu with any questions about this content.

Our inputs are:

- a CSV file with the locations of our resources ("chicago_methadone.csv"), and
- a zip code boundary file ("chicago_zips.shp").

We will calculate the minimum distance between the resources and the centroids of the zip codes, then save the results as a shapefile and as a CSV.

Our final result will be a shapefile/CSV with the minimum distance value for each zip.

Packages used

We will use the following packages in this tutorial:

- `sf`: to manipulate spatial data
- `tmap`: to visualize and create maps
- `units`: to convert units within spatial data

First, install the relevant R packages with the following commands:

```
install.packages("sf")
install.packages("tmap")
install.packages("units")
```

Then load the libraries for use. (The message you see about GEOS, GDAL, and PROJ refer to the software libraries that allow you to work with spatial data.)

```
library(sf)
```

```
## Linking to GEOS 3.7.2, GDAL 2.4.2, PROJ 5.2.0
```

```
library(tmap)
library(units)
```

```
## udunits system database from /Library/Frameworks/R.framework/Versions/3.6/Resources/library/units/sha
```

Read in resource data

We will use a CSV of methadone clinic addresses in Chicago as an example. This file represents point locations of clinics.

```
methadone_clinics <- read.csv("chicago_methadone.csv")
```

Take a look at the first few rows of your data.

```
head(methadone_clinics)
```

```
##   X                                                    Name
## 1 1               Chicago Treatment and Counseling Center, Inc.
## 2 2                     Sundace Methadone Treatment Center, LLC
## 3 3 Soft Landing Interventions/DBA Symetria Recovery of Lakeview
## 4 4                                       PDSSC - Chicago, Inc.
## 5 5                          Center for Addictive Problems, Inc.
## 6 6                                Family Guidance Centers, Inc.
##                    Address    City State   Zip Longitude Latitude
## 1 4453 North Broadway st. Chicago    IL 60640 -87.65594 41.96303
## 2 4545 North Broadway St. Chicago    IL 60640 -87.65703 41.96481
## 3    3934 N. Lincoln Ave. Chicago    IL 60613 -87.67844 41.95321
## 4     2260 N. Elston Ave. Chicago    IL 60614 -87.67412 41.92272
## 5        609 N. Wells St. Chicago    IL 60654 -87.63406 41.89273
## 6     310 W. Chicago Ave. Chicago    IL 60654 -87.63635 41.89660
```

Our data has been geocoded, which means that it has latitude and longitude as columns associated with the address in the data. If you do not have this information, see the Appendix for how to geocode your data.

Convert to a spatial data frame using the `st_as_sf()` function. The `coords` argument specifies which two columns are the X and Y for your data. We set the `crs` argument equal to 4326 because this data is in latitude and longitude (otherwise known as "unprojected", which means it is not in feet or meters).

```
meth_sf <- st_as_sf(methadone_clinics,
                    coords = c("Longitude", "Latitude"),
                    crs = 4326)

meth_sf
```

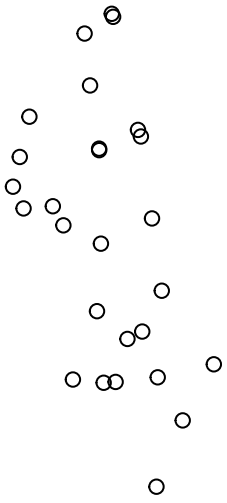
```
## Simple feature collection with 27 features and 6 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: -87.73491 ymin: 41.68699 xmax: -87.57656 ymax: 41.96481
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
## First 10 features:
##     X                                                    Name
## 1   1               Chicago Treatment and Counseling Center, Inc.
## 2   2                     Sundace Methadone Treatment Center, LLC
## 3   3 Soft Landing Interventions/DBA Symetria Recovery of Lakeview
## 4   4                                       PDSSC - Chicago, Inc.
## 5   5                          Center for Addictive Problems, Inc.
## 6   6                                Family Guidance Centers, Inc.
## 7   7                                     A Rincon Family Services
## 8   8                                                            *
## 9   9                   Healthcare Alternative Systems, Inc./NEXA
## 10 10                         Specialized Assistance Services, NFP
```

```
##                    Address    City State   Zip                    geometry
## 1  4453 North Broadway st. Chicago    IL 60640 POINT (-87.65594 41.96303)
## 2  4545 North Broadway St. Chicago    IL 60640 POINT (-87.65703 41.96481)
## 3     3934 N. Lincoln Ave. Chicago    IL 60613 POINT (-87.67844 41.95321)
## 4      2260 N. Elston Ave. Chicago    IL 60614 POINT (-87.67412 41.92272)
## 5         609 N. Wells St. Chicago    IL 60654 POINT (-87.63406 41.89273)
## 6      310 W. Chicago Ave. Chicago    IL 60654  POINT (-87.63635 41.8966)
## 7       3809 W. Grand Ave. Chicago    IL 60651 POINT (-87.72196 41.90436)
## 8      140 N. Ashland Ave. Chicago    IL 60607  POINT (-87.66694 41.8847)
## 9      210 N. Ashland Ave. Chicago    IL 60607   POINT (-87.667 41.88561)
## 10     2630 S. Wabash Ave. Chicago    IL 60616  POINT (-87.6253 41.84459)
```

Note that this is a data frame, but that it has a final column called "geometry" that stores the spatial information.

We can now plot the location of the methadone clinics with base R. We use the `st_geometry()` function to just get a single point map from the geographies.

```
plot(st_geometry(meth_sf))
```

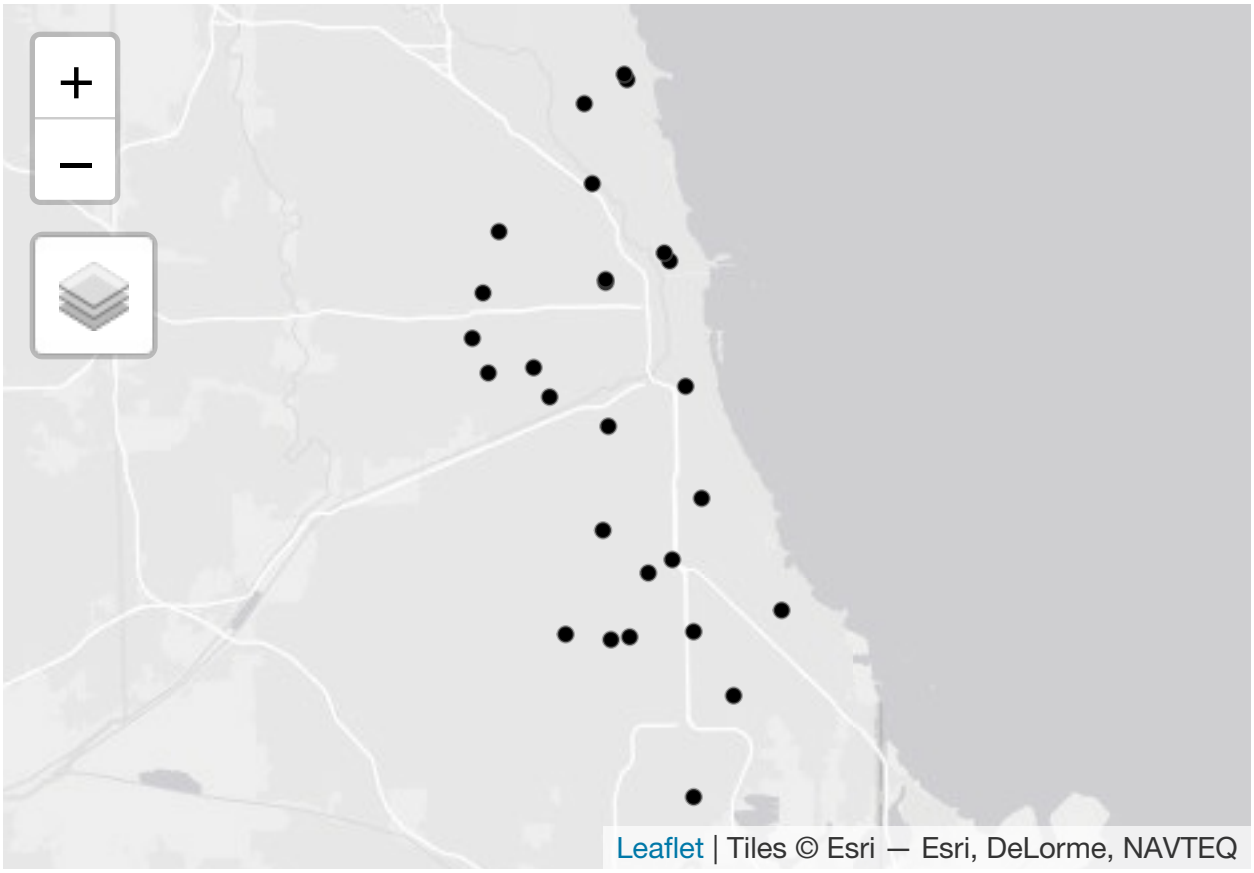


To make a slightly more interesting map, you can add an interactive basemap with tmap, using the `tmap_mode` function to change to "view" mode:

```
tmap_mode("view")
```

```
## tmap mode set to interactive viewing
```

```
tm_shape(meth_sf) +
  tm_dots()
```



Read in zip code data

If you have zip code boundary data from the Census (or other relevant site), you can load them into R with the `read_sf` command. Boundary data is commonly in the **shapefile** format, which has both a spatial (.shp, .shx, .prj) and a flat-file (.dbf) component. Shapefiles are made of four files (.shp, .shx, .prj, .dbf), all which needed to be in the same folder for the file to be read.

```
chicago_zips <- read_sf("chicago_zips.shp")
```

Note: If you do not have the zip boundary data, please see the Appendix for instructions on how to pull them directly from the Census website into R.

If we take a look at the top of the data, we can see that the zip codes have data attached to them. The last column is the "geometry" column, which stores the spatial data.

Additionally, there is a header with some spatial metadata about the data frame, including the type of geometry ("MULTIPOLYGON"), the bounding box (the square that surrounds your data), and the geographic projection (4326 is the shortcode reference for the string that starts "+proj=longlat +datum=WGS84 +no_defs"). Otherwise, this is just like your normal R tabular data frame.

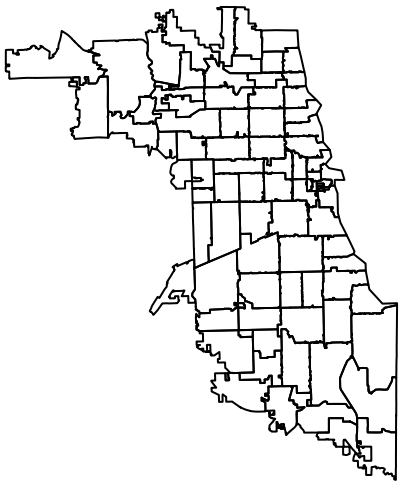
```
head(chicago_zips)
```

```
## Simple feature collection with 6 features and 9 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -88.06058 ymin: 41.73452 xmax: -87.58209 ymax: 42.04052
## epsg (SRID):    4326
```

```
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
## # A tibble: 6 x 10
##   ZCTA5CE10 GEOID10 CLASSFP10 MTFCC10 FUNCSTAT10 ALAND10 AWATER10 INTPTLAT10
##   <chr>     <chr>   <chr>     <chr>   <chr>      <chr>   <chr>    <chr>
## 1 60501     60501   B5        G6350   S          125322~ 974360   +41.78022~
## 2 60007     60007   B5        G6350   S          364933~ 917560   +42.00860~
## 3 60651     60651   B5        G6350   S          9052862 0        +41.90209~
## 4 60652     60652   B5        G6350   S          129878~ 0        +41.74793~
## 5 60653     60653   B5        G6350   S          6041418 1696670  +41.81996~
## 6 60654     60654   B5        G6350   S          1464813 113471   +41.89182~
## # ... with 2 more variables: INTPTLON10 <chr>, geometry <MULTIPOLYGON [°]>
```

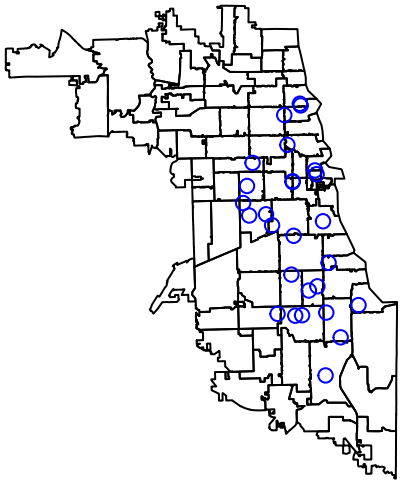
We can check that we pulled the zip code data properly by plotting it. Again, we use the `st_geometry()` function to just get the outline of the geometries.

```
plot(st_geometry(chicago_zips))
```



We can add a second layer in blue with the access locations:

```
plot(st_geometry(chicago_zips))
plot(st_geometry(meth_sf), col = "blue", add = TRUE)
```

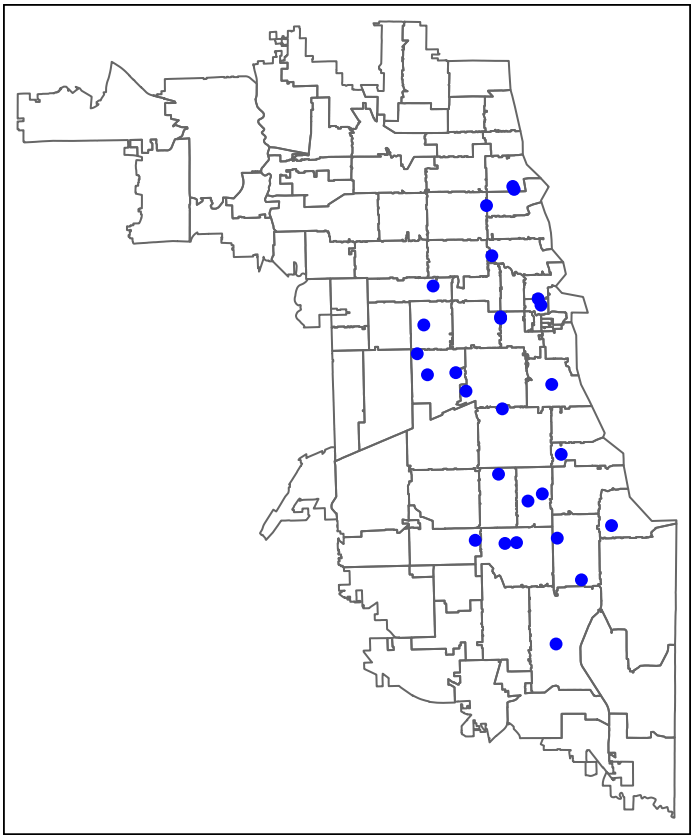


With multiple layers, it can be easier to use tmap to plot:

```
tmap_mode("plot")
```

```
## tmap mode set to plotting
```

```
tm_shape(chicago_zips) +
  tm_borders() +
tm_shape(meth_sf) +
  tm_dots(col = "blue", size = 0.2)
```

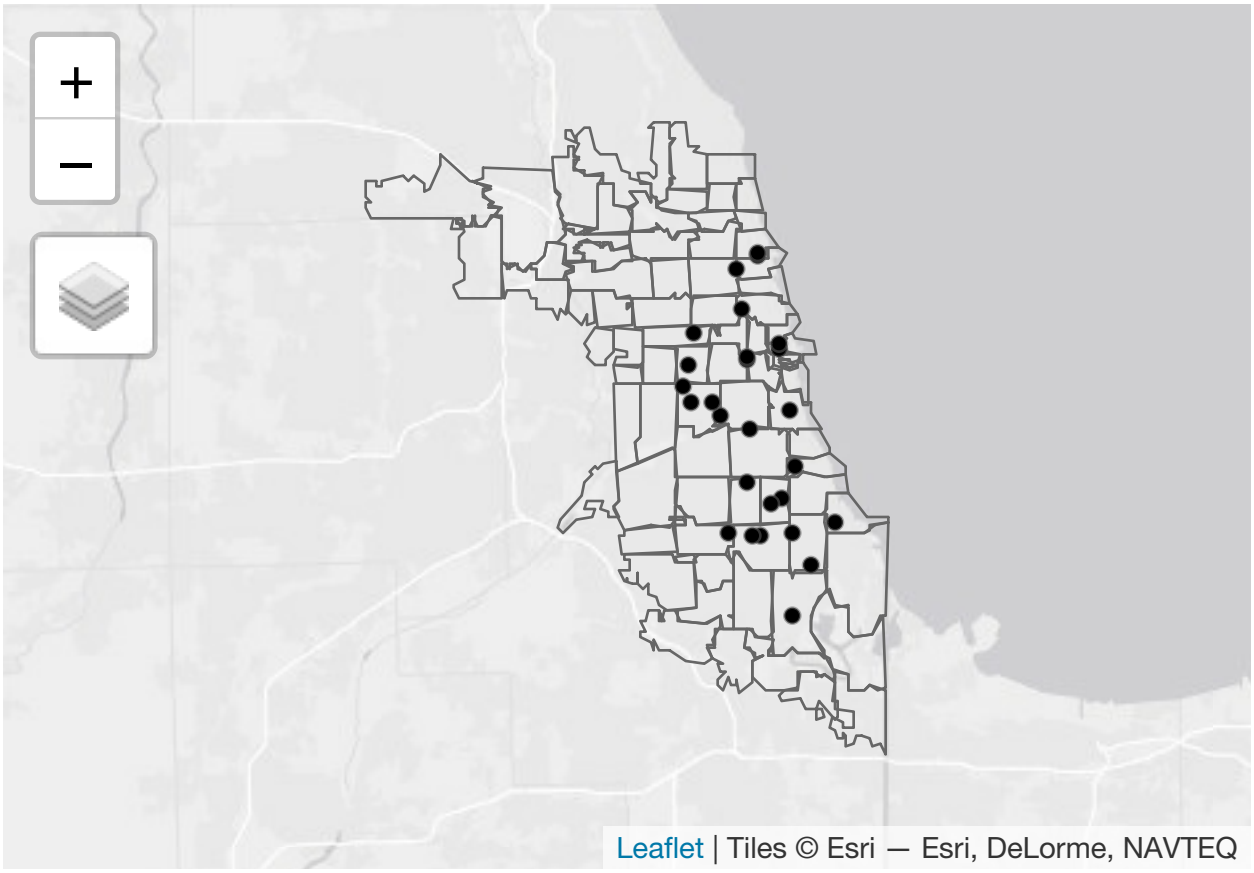


Again, we can create an interactive map with tmap:

```
tmap_mode("view")
```

```
## tmap mode set to interactive viewing
```

```
tm_shape(chicago_zips) +
  tm_borders() +
tm_shape(meth_sf) +
  tm_dots()
```



Calculate centroids of zip code boundaries

Now, we will calculate the centroids of the zip code boundaries.

We will first need to project our data, which means change it from latitude and longitude to meaningful units, like ft or meters, so we can calculate distance properly. We'll use the Illinois State Plane projection, with an EPSG code of 3435.

Aside: To find the most appropriate projection for your data, do a Google Search for which projection works well - for state level data, each state has a State Plane projection with a specific code, known as the EPSG. I use epsg.io to check projections - here's the New York State Plane page.

Use the `st_transform` function to change the projection of the data. Notice how the values in `geometry` go from being relatively small (unprojected, lat/long) to very large (projected, in US feet).

```
chicago_zips <- st_transform(chicago_zips, 3435)

chicago_zips
```

```
## Simple feature collection with 85 features and 9 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 1058388 ymin: 1791133 xmax: 1205317 ymax: 1966816
## epsg (SRID):    3435
## proj4string:    +proj=tmerc +lat_0=36.66666666666666 +lon_0=-88.33333333333333 +k=0.9999749999999999
## # A tibble: 85 x 10
```

```
##    ZCTA5CE10 GEOID10 CLASSFP10 MTFCC10 FUNCSTAT10 ALAND10 AWATER10 INTPTLAT10
##    <chr>     <chr>   <chr>     <chr>   <chr>      <chr>   <chr>    <chr>
##  1 60501     60501   B5        G6350   S          125322~ 974360   +41.78022~
##  2 60007     60007   B5        G6350   S          364933~ 917560   +42.00860~
##  3 60651     60651   B5        G6350   S          9052862 0        +41.90209~
##  4 60652     60652   B5        G6350   S          129878~ 0        +41.74793~
##  5 60653     60653   B5        G6350   S          6041418 1696670  +41.81996~
##  6 60654     60654   B5        G6350   S          1464813 113471   +41.89182~
##  7 60655     60655   B5        G6350   S          114080~ 0        +41.69477~
##  8 60656     60656   B5        G6350   S          8465226 0        +41.97428~
##  9 60657     60657   B5        G6350   S          5888324 2025836  +41.94029~
## 10 60659     60659   B5        G6350   S          5251086 2818     +41.99148~
## # ... with 75 more rows, and 2 more variables: INTPTLON10 <chr>,
## #   geometry <MULTIPOLYGON [US_survey_foot]>
```

Then, we will calculate the centroids:

```
chicago_centroids <- st_centroid(chicago_zips)
```

```
## Warning in st_centroid.sf(chicago_zips): st_centroid assumes attributes are
## constant over geometries of x
```

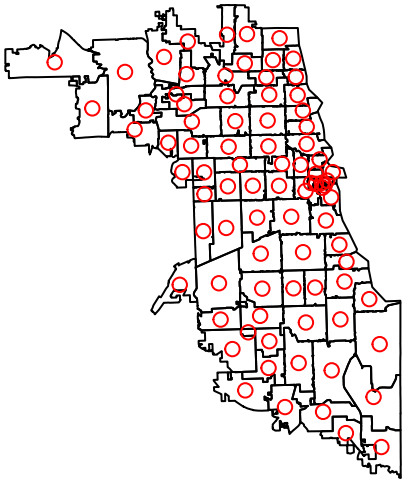
```
chicago_centroids
```

```
## Simple feature collection with 85 features and 9 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: 1076716 ymin: 1802621 xmax: 1198093 ymax: 1956017
## epsg (SRID):    3435
## proj4string:    +proj=tmerc +lat_0=36.66666666666666 +lon_0=-88.33333333333333 +k=0.9999749999999999
## # A tibble: 85 x 10
##    ZCTA5CE10 GEOID10 CLASSFP10 MTFCC10 FUNCSTAT10 ALAND10 AWATER10 INTPTLAT10
##    <chr>     <chr>   <chr>     <chr>   <chr>      <chr>   <chr>    <chr>
##  1 60501     60501   B5        G6350   S          125322~ 974360   +41.78022~
##  2 60007     60007   B5        G6350   S          364933~ 917560   +42.00860~
##  3 60651     60651   B5        G6350   S          9052862 0        +41.90209~
##  4 60652     60652   B5        G6350   S          129878~ 0        +41.74793~
##  5 60653     60653   B5        G6350   S          6041418 1696670  +41.81996~
##  6 60654     60654   B5        G6350   S          1464813 113471   +41.89182~
##  7 60655     60655   B5        G6350   S          114080~ 0        +41.69477~
##  8 60656     60656   B5        G6350   S          8465226 0        +41.97428~
##  9 60657     60657   B5        G6350   S          5888324 2025836  +41.94029~
## 10 60659     60659   B5        G6350   S          5251086 2818     +41.99148~
## # ... with 75 more rows, and 2 more variables: INTPTLON10 <chr>,
## #   geometry <POINT [US_survey_foot]>
```

For each zip code, this will calculate the centroid, and the output will be a point dataset.

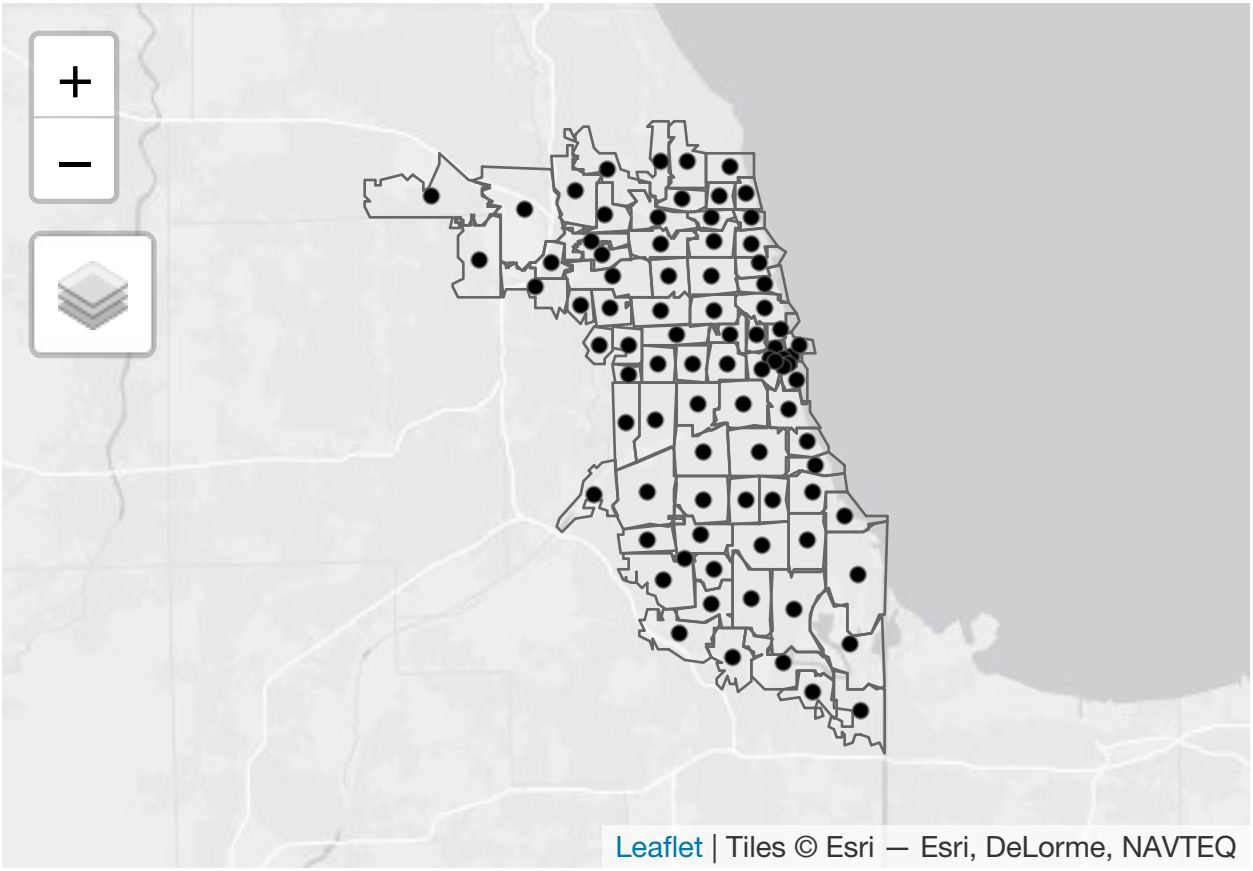
Plot to double check that everything is ok. The `st_geometry()` function will once again just return the outline:

```
plot(st_geometry(chicago_zips))
plot(st_geometry(chicago_centroids), add = TRUE, col = "red")
```



Once again, we can create an interactive map:

```
tm_shape(chicago_zips) +
  tm_borders() +
tm_shape(chicago_centroids) +
  tm_dots()
```



Ensure that centroid and resource projections match

If we immediately try to calculate the distance between the zip centroids and the locations of the resources using the `st_distance` function, we'll get an error:

```
st_distance(chicago_centroids, meth_sf, by_element = TRUE)
```

```
Error in st_distance(chicago_centroids, meth_sf, by_element = TRUE) : st_crs(x) == st_crs(y) is not TRU
```

Why is there an error? Because the projection of the centroids and the resource locations don't match up. Let's project the resource locations so that they match the projection of the centroids.

First, use the `st_crs` function to check that the coordinate reference system (or projection) is the same. They're not, so we have to fix it.

```
st_crs(chicago_centroids)
```

```
## Coordinate Reference System:
##   EPSG: 3435
##   proj4string: "+proj=tmerc +lat_0=36.66666666666666 +lon_0=-88.33333333333333 +k=0.9999749999999999
```

```
st_crs(meth_sf)
```

```
## Coordinate Reference System:
##   EPSG: 4326
##   proj4string: "+proj=longlat +datum=WGS84 +no_defs"
```

We'll take the CRS from the zip code centroids data, and use it as input to `st_transform` applied to the methadone clinics data.

```
new_crs <- st_crs(chicago_centroids)
new_crs
```

```
## Coordinate Reference System:
##   EPSG: 3435
##   proj4string: "+proj=tmerc +lat_0=36.66666666666666 +lon_0=-88.33333333333333 +k=0.9999749999999999
```

```
meth_sf <- st_transform(meth_sf, new_crs)
```

If we check the CRS again, we now see that they match. **Mismatched projections are a commonly made mistake in geospatial data processing.**

```
st_crs(chicago_centroids)
```

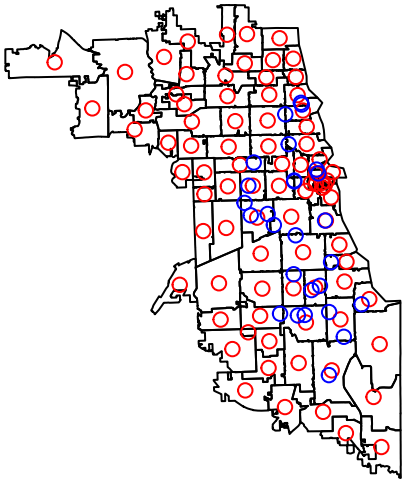
```
## Coordinate Reference System:
##   EPSG: 3435
##   proj4string: "+proj=tmerc +lat_0=36.66666666666666 +lon_0=-88.33333333333333 +k=0.9999749999999999
```

```
st_crs(meth_sf)
```

```
## Coordinate Reference System:
##   EPSG: 3435
##   proj4string: "+proj=tmerc +lat_0=36.66666666666666 +lon_0=-88.33333333333333 +k=0.9999749999999999
```

Now we have the zip boundaries, the centroids of the zips, and the resource locations, as shown below. Next, we will calculate the distance to the nearest resource from each zip code centroid.

```
plot(st_geometry(chicago_zips))
plot(st_geometry(chicago_centroids), col = "red", add = TRUE)
plot(st_geometry(meth_sf), col = "blue", add = TRUE)
```



Calculate distance from centroid to nearest resource

First, we'll identify the resource that is the closest to a zip centroid using the `st_nearest_feature` function. (It will return the index of the object that is nearest, so we will subset the resources by the index to get the nearest object.)

```
nearest_clinic_indexes <- st_nearest_feature(chicago_centroids, meth_sf)

nearest_clinic <- meth_sf[nearest_clinic_indexes,]

nearest_clinic
```

```
## Simple feature collection with 85 features and 6 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: 1147259 ymin: 1829334 xmax: 1190725 ymax: 1930492
## epsg (SRID):    3435
## proj4string:    +proj=tmerc +lat_0=36.66666666666666 +lon_0=-88.33333333333333 +k=0.9999749999999999
## First 10 features:
##       X                                                    Name
## 16   16                          Katherine Boone Robinson Foundation
## 7     7                                    A Rincon Family Services
## 7.1   7                                    A Rincon Family Services
## 26   26                             New Hope Community Service Center
## 15   15         HRDI- Grand Boulevard Professional Counseling Center
## 5     5                          Center for Addictive Problems, Inc.
## 26.1 26                             New Hope Community Service Center
## 7.2   7                                    A Rincon Family Services
## 1     1               Chicago Treatment and Counseling Center, Inc.
## 3     3 Soft Landing Interventions/DBA Symetria Recovery of Lakeview
##                     Address    City State   Zip                geometry
## 16        4100 W. Ogden Ave. Chicago    IL 60623 POINT (1149563 1888684)
## 7         3809 W. Grand Ave. Chicago    IL 60651 POINT (1150678 1908331)
## 7.1       3809 W. Grand Ave. Chicago    IL 60651 POINT (1150678 1908331)
## 26          2559 W. 79th St. Chicago    IL 60652 POINT (1160443 1852136)
## 15           340 E. 51st St. Chicago    IL 60615 POINT (1179400 1871296)
## 5           609 N. Wells St. Chicago    IL 60654 POINT (1174640 1904278)
```

```
## 26.1        2559 W. 79th St. Chicago    IL 60652 POINT (1160443 1852136)
## 7.2       3809 W. Grand Ave. Chicago    IL 60651 POINT (1150678 1908331)
## 1    4453 North Broadway st. Chicago    IL 60640 POINT (1168480 1929847)
## 3       3934 N. Lincoln Ave. Chicago    IL 60613 POINT (1162389 1926221)
```

Then, we will calculate the distance between the nearest resource and the zip code centroid with the `st_distance` function. As shown above, make sure both of your datasets are projected, and in the same projection, before you run `st_distance`.

```
min_dists <- st_distance(chicago_centroids, nearest_clinic, by_element = TRUE)

min_dists
```

```
## Units: [US_survey_foot]
##  [1] 36899.7187 82794.9499  5210.0088  7446.1648  7192.4268   885.6142
##  [7] 20584.4913 38314.4490  8469.9351 15479.7403  9796.8522  4469.3071
## [13] 33683.4980 24082.8186 24169.2397 45189.1792 31267.0776 10254.9649
## [19] 10958.9389 13821.3363 49825.9391 32430.5115 36620.8289 22036.8980
## [25] 13688.1510 22177.9153 63240.9022  4249.2975  3766.8707  5131.5781
## [31]  5548.8181  8889.3859  3988.0292  5492.6866  7091.5663  6849.3251
## [37]  3958.0982 45915.8759 32569.9607 44521.9752 58458.5465  5406.8794
## [43]  5887.8101  2278.0342  6660.2051  5735.8249   304.8631 13604.9478
## [49]  6942.3909  4993.9155  2841.4986  1679.0098  7651.3608  2529.0080
## [55]  1667.5299  9406.8277 16622.9728  2042.7828 11421.7437 22480.9662
## [61] 12104.7974  7613.8768 39613.5103 11724.1443 18463.5889 27529.9719
## [67]  5232.8529  8774.4241 25352.7928 18954.0196 26416.7824  7550.1810
## [73]  3455.8152 10997.6485  3097.7944 16812.6822  6171.7070 25247.7440
## [79] 17149.4029 15235.8434 25019.9566 18574.8897 20179.9325 33065.4125
## [85] 22450.6644
```

This is in US feet. To change to a more meaningful unit, such as miles, we can use the `set_units()` function:

```
min_dists_mi <- set_units(min_dists, "mi")

min_dists_mi
```

```
## Units: [mi]
##  [1]  6.98859707 15.68089308  0.98674606  1.41026130  1.36220476  0.16773030
##  [7]  3.89858569  7.25653895  1.60415759  2.93177485  1.85546815  0.84646137
## [13]  6.37946314  4.56114901  4.57751667  8.55857378  5.92180684  1.94223208
## [19]  2.07556077  2.61768256  9.43674977  6.14215462  6.93577693  4.17366328
## [25]  2.59245803  4.20037114 11.97746755  0.80479280  0.71342392  0.97189174
## [31]  1.05091463  1.68359919  0.75531008  1.04028363  1.34310236  1.29722327
## [37]  0.74964130  8.69620601  6.16856550  8.43220914 11.07171655  1.02403224
## [43]  1.11511778  0.43144674  1.26140501  1.08633265  0.05773933  2.57669981
## [49]  1.31484938  0.94581922  0.53816370  0.31799492  1.44912426  0.47897974
## [55]  0.31582069  1.78159972  3.14829660  0.38689146  2.16321335  4.25776726
## [61]  2.29257984  1.44202501  7.50257377  2.22048632  3.49689882  5.21402025
## [67]  0.99107260  1.66182607  4.80167491  3.58978362  5.00318849  1.42996139
## [73]  0.65451176  2.08289214  0.58670466  3.18422648  1.16888624  4.78177925
## [79]  3.24799947  2.88558217  4.73863762  3.51797856  3.82196456  6.26240125
## [85]  4.25202828
```

We then rejoin the minimum distances to the zip code data, by column binding `min_dists_mi` to the original `chicago_zips` data.

```
min_dist_sf <- cbind(chicago_zips, min_dists_mi)
min_dist_sf
```

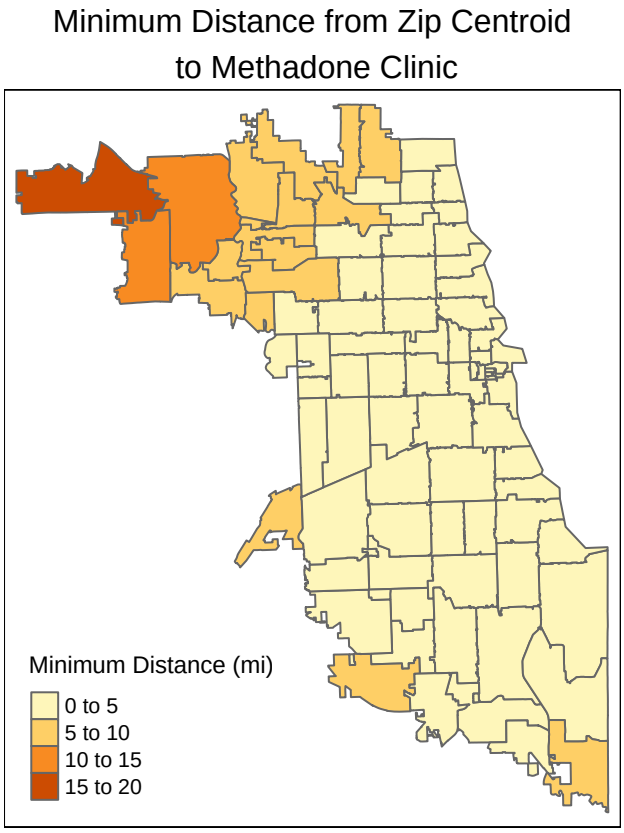
```
## Simple feature collection with 85 features and 10 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 1058388 ymin: 1791133 xmax: 1205317 ymax: 1966816
## epsg (SRID):    3435
## proj4string:    +proj=tmerc +lat_0=36.66666666666666 +lon_0=-88.33333333333333 +k=0.9999749999999999
## First 10 features:
##    ZCTA5CE10 GEOID10 CLASSFP10 MTFCC10 FUNCSTAT10  ALAND10 AWATER10  INTPTLAT10
## 1      60501   60501        B5   G6350          S 12532295   974360 +41.7802209
## 2      60007   60007        B5   G6350          S 36493383   917560 +42.0086000
## 3      60651   60651        B5   G6350          S  9052862        0 +41.9020934
## 4      60652   60652        B5   G6350          S 12987857        0 +41.7479319
## 5      60653   60653        B5   G6350          S  6041418  1696670 +41.8199645
## 6      60654   60654        B5   G6350          S  1464813   113471 +41.8918225
## 7      60655   60655        B5   G6350          S 11408010        0 +41.6947762
## 8      60656   60656        B5   G6350          S  8465226        0 +41.9742800
## 9      60657   60657        B5   G6350          S  5888324  2025836 +41.9402931
## 10     60659   60659        B5   G6350          S  5251086     2818 +41.9914885
##      INTPTLON10    min_dists_mi                       geometry
## 1  -087.8232440  6.9885971 [mi] MULTIPOLYGON (((1112613 185...
## 2  -087.9973398 15.6808931 [mi] MULTIPOLYGON (((1058389 194...
## 3  -087.7408565  0.9867461 [mi] MULTIPOLYGON (((1136069 190...
## 4  -087.7147951  1.4102613 [mi] MULTIPOLYGON (((1145542 185...
## 5  -087.6059654  1.3622048 [mi] MULTIPOLYGON (((1177007 187...
## 6  -087.6383036  0.1677303 [mi] MULTIPOLYGON (((1170904 190...
## 7  -087.7037764  3.8985857 [mi] MULTIPOLYGON (((1146378 183...
## 8  -087.8271283  7.2565390 [mi] MULTIPOLYGON (((1110359 193...
## 9  -087.6468569  1.6041576 [mi] MULTIPOLYGON (((1162394 192...
## 10 -087.7039859  2.9317749 [mi] MULTIPOLYGON (((1148555 194...
```

We can now visualize the zip-level access to methadone clinics using our new access metric, using the `tmap` package.

```
tmap_mode("plot")
```

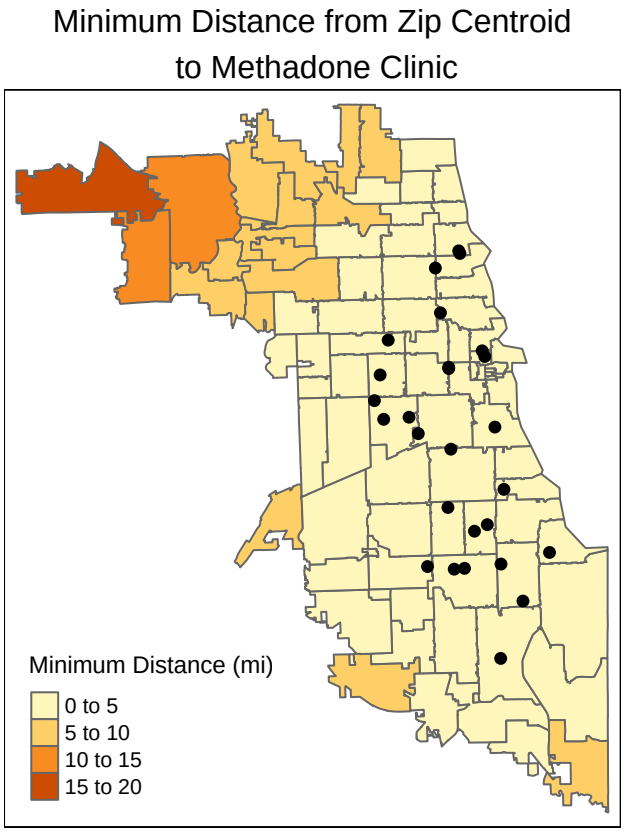
```
## tmap mode set to plotting
```

```
tm_shape(min_dist_sf) +
  tm_polygons("min_dists_mi",
              title = "Minimum Distance (mi)") +
  tm_layout(main.title = "Minimum Distance from Zip Centroid\n to Methadone Clinic",
            main.title.position = "center",
            main.title.size = 1)
```



Access by zip code can also be combined with locations of resources:

```
tm_shape(min_dist_sf) +
  tm_polygons("min_dists_mi",
              title = "Minimum Distance (mi)") +
  tm_shape(meth_sf) +
  tm_dots(size = 0.2) +
  tm_layout(main.title = "Minimum Distance from Zip Centroid\n to Methadone Clinic",
            main.title.position = "center",
            main.title.size = 1)
```



Save as zip-code level dataset

To save our final result to a CSV, use the `layer_options = "GEOMETRY=AS_XY"` command. Note that this option only works when you are working with point data.

```
write_sf(min_dist_sf, "min_dist.csv", layer_options = "GEOMETRY=AS_XY")
```

We can also write out this data to a shapefile format:

```
write_sf(min_dist_sf, "min_dists_sf.shp")
```

Appendix

Geocode addresses

If you have the addresses for your resources, but no latitude and longitude associated, you will need to geocode your resources.

To do so, there are a number of options in R. I like the OpenCage geocoder in the `tmaptools` packaage. This caps you at around 2000 addresses at once, and may require some data manipulation, so you may wish to use a proprietary software instead (i.e. Esri Geocoder, which is included in Esri software like ArcGIS).

Here's an example of geocoding a single address:

```
# install.packages(tmaptools)
library(tmaptools)
```

```
geocode_OSM("4545 North Broadway St., Chicago, IL",
            as.data.frame = TRUE)
```

```
##                                 query      lat       lon  lat_min  lat_max
## 1 4545 North Broadway St., Chicago, IL 43.04382 -87.90796 43.04366 43.04397
##     lon_min  lon_max
## 1 -87.90821 -87.9077
```

To apply the function to multiple addresses, you can try the following:

First ensure that you have a *character vector* of full addresses:

```
full_addresses <- paste(meth_sf$Address, meth_sf$City, meth_sf$State, meth_sf$Zip)

full_addresses
```

```
##  [1] "4453 North Broadway st. Chicago IL 60640"
##  [2] "4545 North Broadway St. Chicago IL 60640"
##  [3] "3934 N. Lincoln Ave. Chicago IL 60613"
##  [4] "2260 N. Elston Ave. Chicago IL 60614"
##  [5] "609 N. Wells St. Chicago IL 60654"
##  [6] "310 W. Chicago Ave. Chicago IL 60654"
##  [7] "3809 W. Grand Ave. Chicago IL 60651"
##  [8] "140 N. Ashland Ave. Chicago IL 60607"
##  [9] "210 N. Ashland Ave. Chicago IL 60607"
## [10] "2630 S. Wabash Ave. Chicago IL 60616"
## [11] "4132 W. Madison St. Chicago IL 60624"
## [12] "3520 S. Ashland Ave. Chicago IL 60609"
## [13] "2800 S. California Ave. Chicago IL 60608"
## [14] "3113 W. Cermak Rd. Chicago IL 60623"
## [15] "340 E. 51st St. Chicago IL 60615"
## [16] "4100 W. Ogden Ave. Chicago IL 60623"
## [17] "1330 S. Kostner Ave. Chicago IL 60623"
## [18] "5701 S. Wood St. Chicago IL 60636"
## [19] "326 W. 64th st. Chicago IL 60621"
## [20] "1950 E 75th St. Chicago IL 60649"
## [21] "6614 S Halsted St. Chicago IL 60621"
## [22] "110 E. 79th St. Chicago IL 60619"
## [23] "8000 S. Racine Ave. Chicago IL 60620"
## [24] "936 E. 93rd St. Chicago IL 60619"
## [25] "8014 S. Ashland Ave. Chicago IL 60620"
## [26] "2559 W. 79th St. Chicago IL 60652"
## [27] "33 E 114th St. Chicago IL 60628"
```

```
class(full_addresses)
```

```
## [1] "character"
```

Then geocode the addresses. This can now be transformed into an `sf` object and be used in the rest of analysis.

```
geocode_OSM(full_addresses, as.data.frame = TRUE)
```

```
## Warning in FUN(X[[i]], ...): No results found for "4453 North Broadway st.
## Chicago IL 60640".
```

```
## Warning in FUN(X[[i]], ...): No results found for "4545 North Broadway St.
## Chicago IL 60640".
```

```
##                                      query      lat       lon  lat_min
## 1     3934 N. Lincoln Ave. Chicago IL 60613 41.95327 -87.67844 41.95317
## 2      2260 N. Elston Ave. Chicago IL 60614 41.92263 -87.67425 41.92252
## 3         609 N. Wells St. Chicago IL 60654 41.89271 -87.63379 41.89264
## 4      310 W. Chicago Ave. Chicago IL 60654 41.89679 -87.63640 41.89670
## 5       3809 W. Grand Ave. Chicago IL 60651 41.90411 -87.72202 41.90394
## 6      140 N. Ashland Ave. Chicago IL 60607 41.88474 -87.66725 41.88434
## 7      210 N. Ashland Ave. Chicago IL 60607 41.88606 -87.66711 41.88601
## 8      2630 S. Wabash Ave. Chicago IL 60616 41.84456 -87.62575 41.84446
## 9      4132 W. Madison St. Chicago IL 60624 41.88106 -87.69873 41.88103
## 10    3520 S. Ashland Ave. Chicago IL 60609 41.82977 -87.66586 41.82973
## 11 2800 S. California Ave. Chicago IL 60608 41.84058 -87.69523 41.84053
## 12      3113 W. Cermak Rd. Chicago IL 60623 41.85160 -87.70347 41.85135
## 13         340 E. 51st St. Chicago IL 60615 41.80236 -87.61762 41.80217
## 14      4100 W. Ogden Ave. Chicago IL 60623 41.86128 -87.69588 41.86105
## 15    1330 S. Kostner Ave. Chicago IL 60623 41.86331 -87.73523 41.86317
## 16        5701 S. Wood St. Chicago IL 60636 41.79005 -87.66861 41.78988
## 17         326 W. 64th st. Chicago IL 60621 41.77859 -87.63306 41.77734
## 18         1950 E 75th St. Chicago IL 60649 41.75912 -87.57656 41.75896
## 19      6614 S Halsted St. Chicago IL 60621 41.77378 -87.64501 41.77371
## 20         110 E. 79th St. Chicago IL 60619 41.75125 -87.62078 41.75111
## 21     8000 S. Racine Ave. Chicago IL 60620 41.74848 -87.65400 41.74838
## 22         936 E. 93rd St. Chicago IL 60619 41.72593 -87.60110 41.72585
## 23    8014 S. Ashland Ave. Chicago IL 60620 41.76450 -87.66384 41.76332
## 24        2559 W. 79th St. Chicago IL 60652 41.74978 -87.68772 41.74966
## 25          33 E 114th St. Chicago IL 60628 41.68714 -87.62184 41.68709
##     lat_max   lon_min   lon_max
## 1  41.95336 -87.67861 -87.67825
## 2  41.92276 -87.67446 -87.67411
## 3  41.89277 -87.63396 -87.63356
## 4  41.89698 -87.63670 -87.63602
## 5  41.90430 -87.72220 -87.72183
## 6  41.88516 -87.66754 -87.66707
## 7  41.88611 -87.66716 -87.66706
## 8  41.84468 -87.62608 -87.62542
## 9  41.88107 -87.70084 -87.69665
## 10 41.82981 -87.66599 -87.66574
## 11 41.84063 -87.69528 -87.69518
## 12 41.85169 -87.70356 -87.70337
## 13 41.80252 -87.61786 -87.61738
## 14 41.86133 -87.69655 -87.69575
## 15 41.86345 -87.73545 -87.73500
## 16 41.79026 -87.66920 -87.66830
## 17 41.77980 -87.63486 -87.63186
## 18 41.75928 -87.57663 -87.57649
## 19 41.77386 -87.64524 -87.64478
## 20 41.75139 -87.62084 -87.62073
## 21 41.74858 -87.65418 -87.65381
## 22 41.72600 -87.60114 -87.60105
## 23 41.76492 -87.66386 -87.66380
## 24 41.74990 -87.68786 -87.68758
## 25 41.68719 -87.62189 -87.62179
```

Pull zip code boundaries directly from Census

If you do not have zip code boundaries on hand, you can use an R package called `tigris` to pull them directly from the Census website.

```
# install.packages(tigris)
library(tigris)
```

```
## To enable
## caching of data, set `options(tigris_use_cache = TRUE)` in your R script or .Rprofile.
```

```
##
## Attaching package: 'tigris'
```

```
## The following object is masked from 'package:graphics':
##
##     plot
```

The `zctas()` function will pull the last published year of zip code boundaries, which defaults to 2018. Additional functions to pull states, counties, tracts, blocks, and more can be found in the `tigris` documentation. You can change the year with the `year` argument. **Note: this will take a while to run.**

```
il_zips <- zctas(state = "IL", year = 2017)
```

Once you pull the boundaries, you can plot your data and calculate the centroids.