Interactive Maps for Dummies:

Lecture 2: Draw Maps

SHI Shuang

The Univeristy of Hong Kong

Visiting Research Fellow at Brown University

Jul. 23, 2020

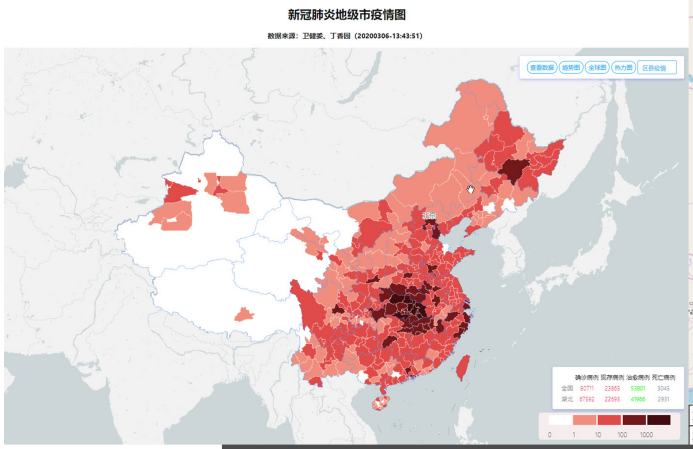
Prepared for HKBU Zoom Lectures

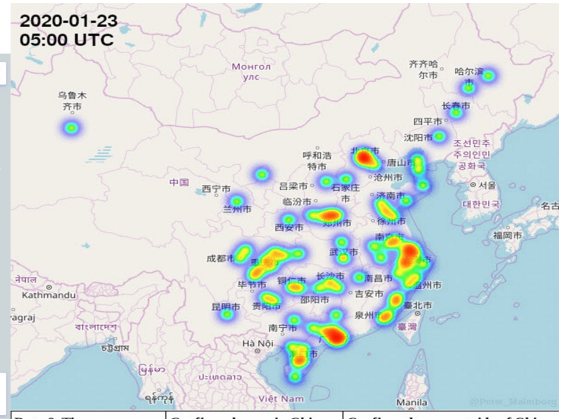
Today

- Recap: shape file and projection
- Use local data
- Web mapping

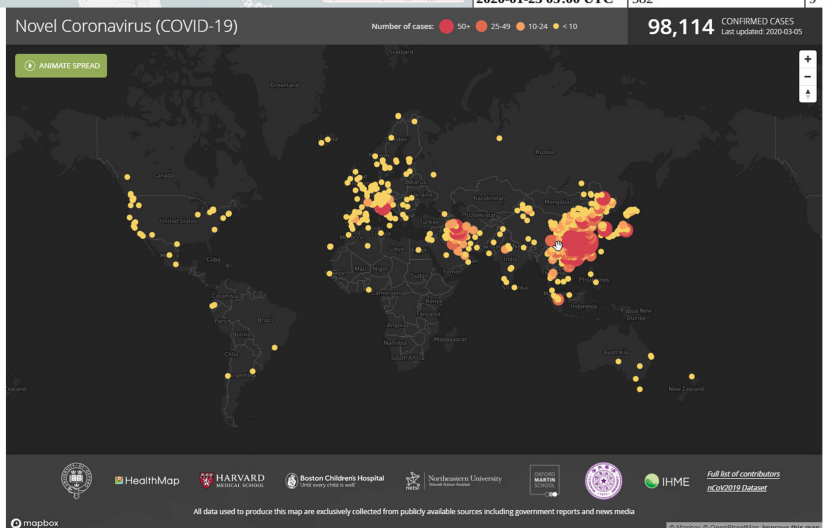
Draw Map

- Choropleth maps show geographical regions colored, shaded, or graded according to some variable
- *R* is not as powerful as ArcGIS with GIS data, but *ggplot2* can draw map

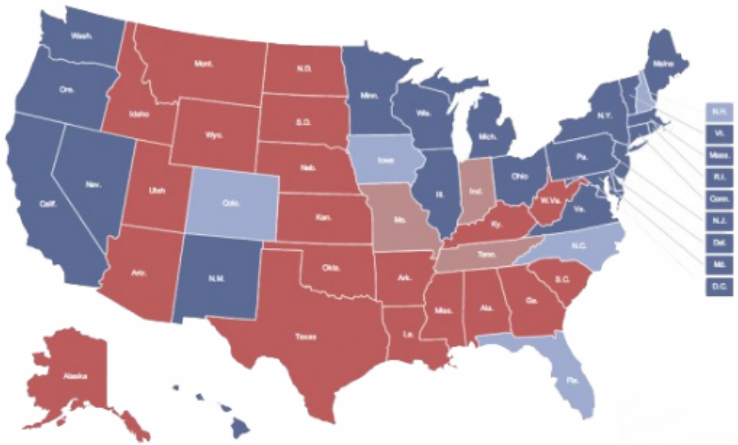


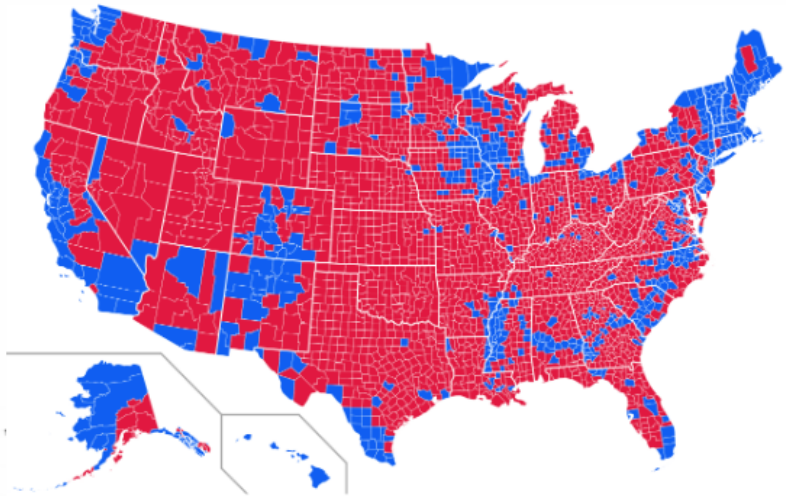


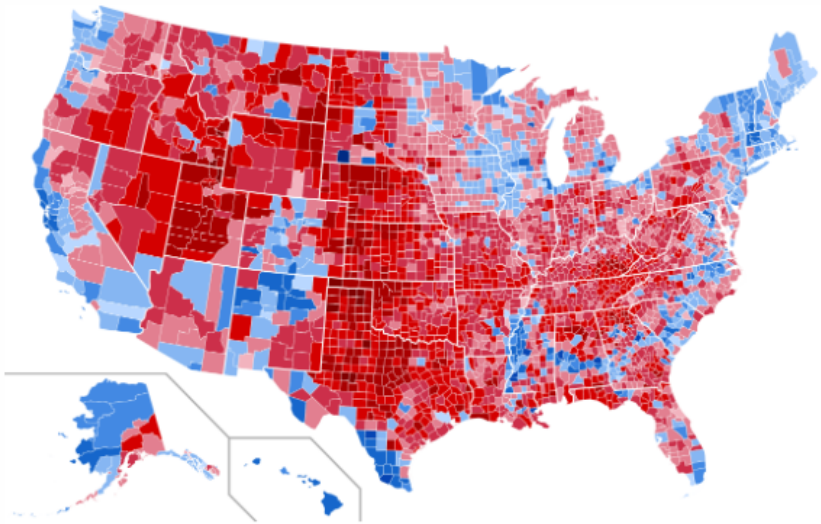
Date & Time	Confirmed cases in China	Confirmed cases outside of China
2020-01-23 05:00 UTC	582	9

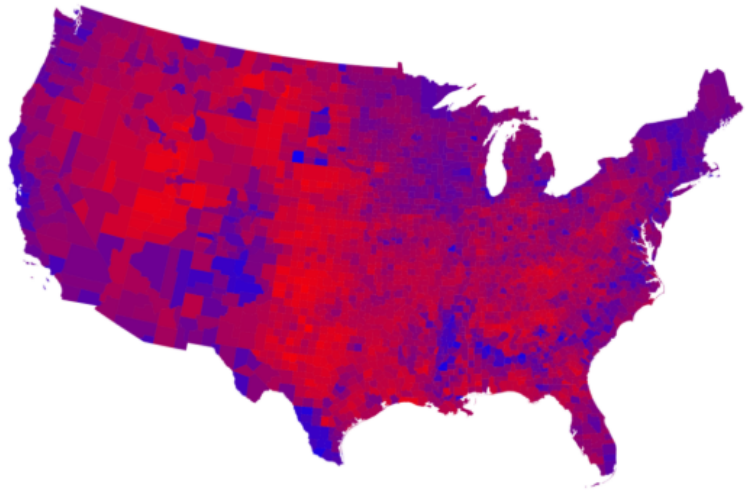


Read a Choropleth Map









Two Types of Spatial Data

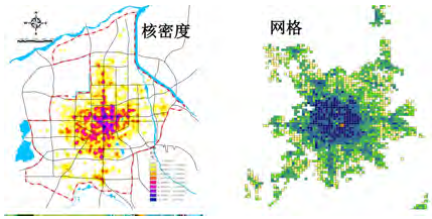
- Vector vs. Raster

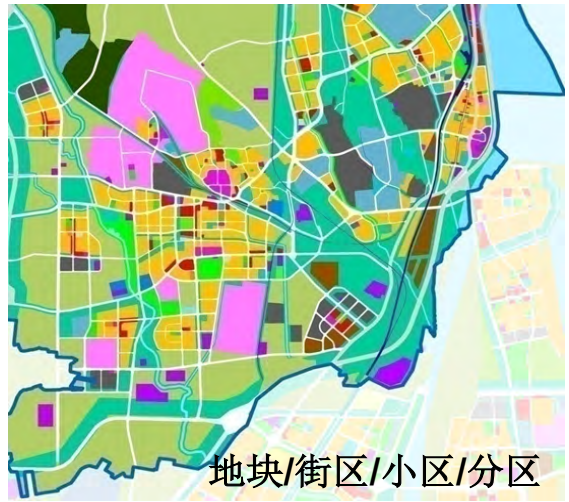
Vector Data

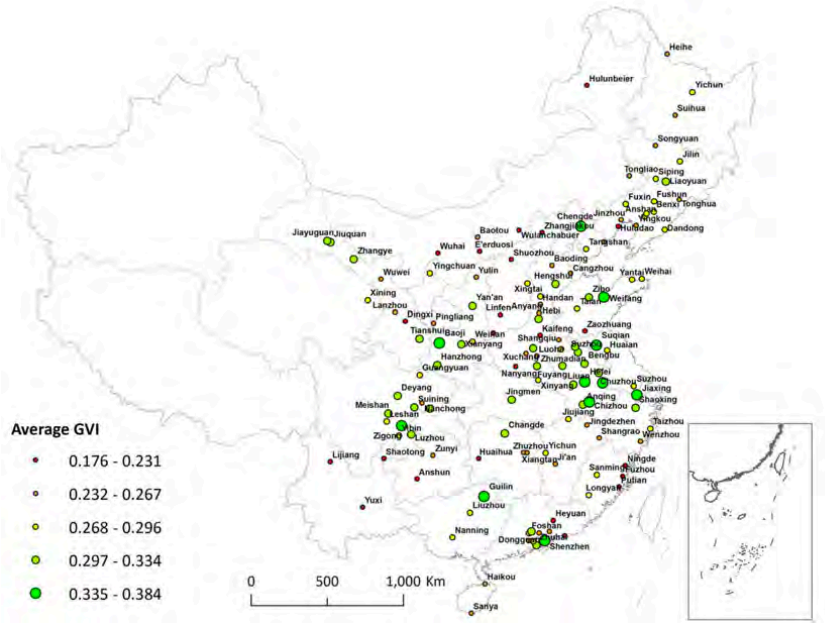
- Vector data basically encoding points, lines, and polygons in geographic space
- Shapefile is a file format specifically for geographic vector data
  - File extension is .shp
  - Other files accompany the .shp file with extensions .dbf and .prj
    - .dbf contains attribute format of shapefile
    - .prg contains information about the projection of the coordinates

Examples of Vector Data

从几何形状角度：点、线和面







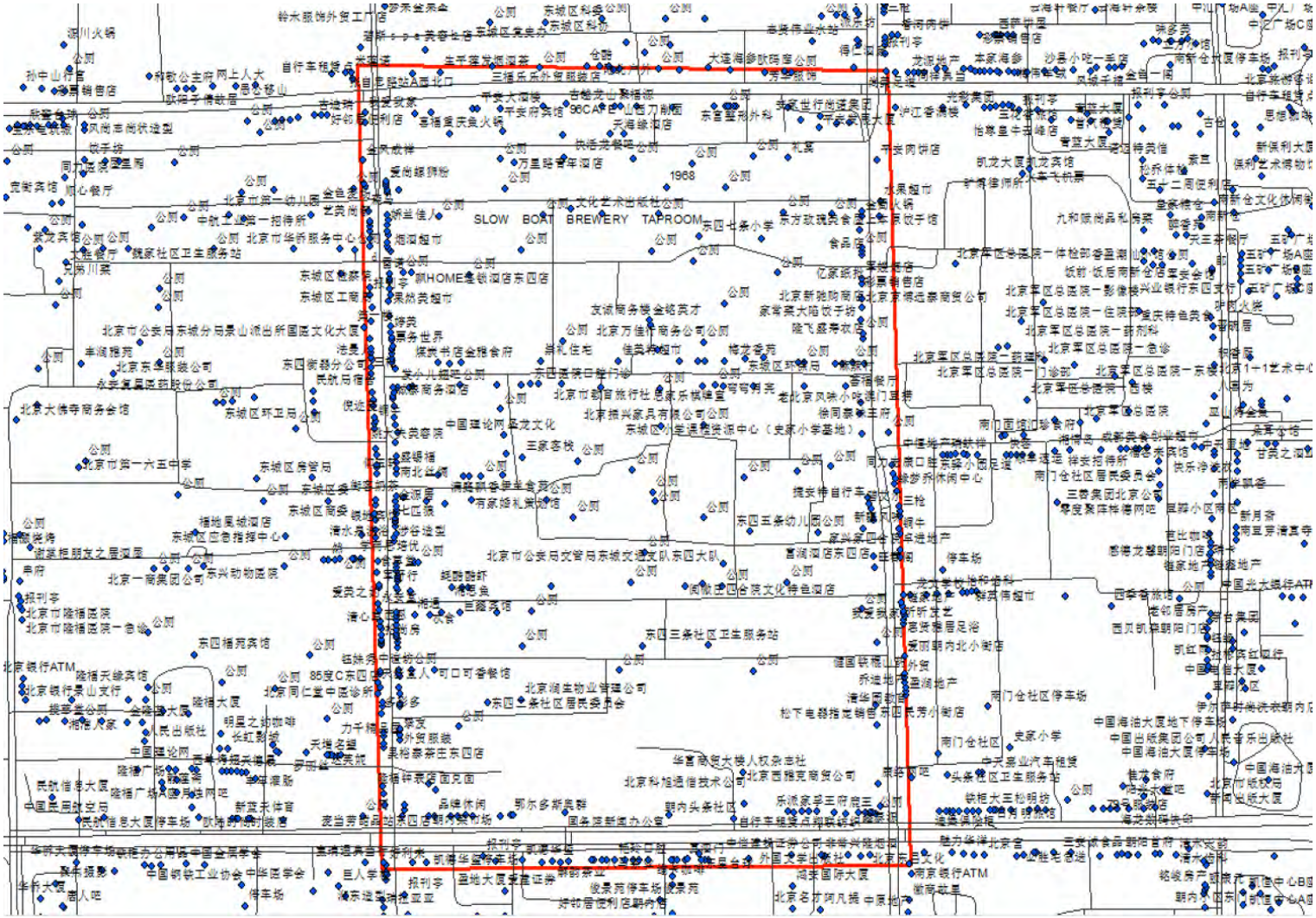
- 空间新数据多以点和线的形式存在
- 分析、统计与可视化多体现在点、线和面三个形式

- https://www.beijingcitylab.com/

Vector Data

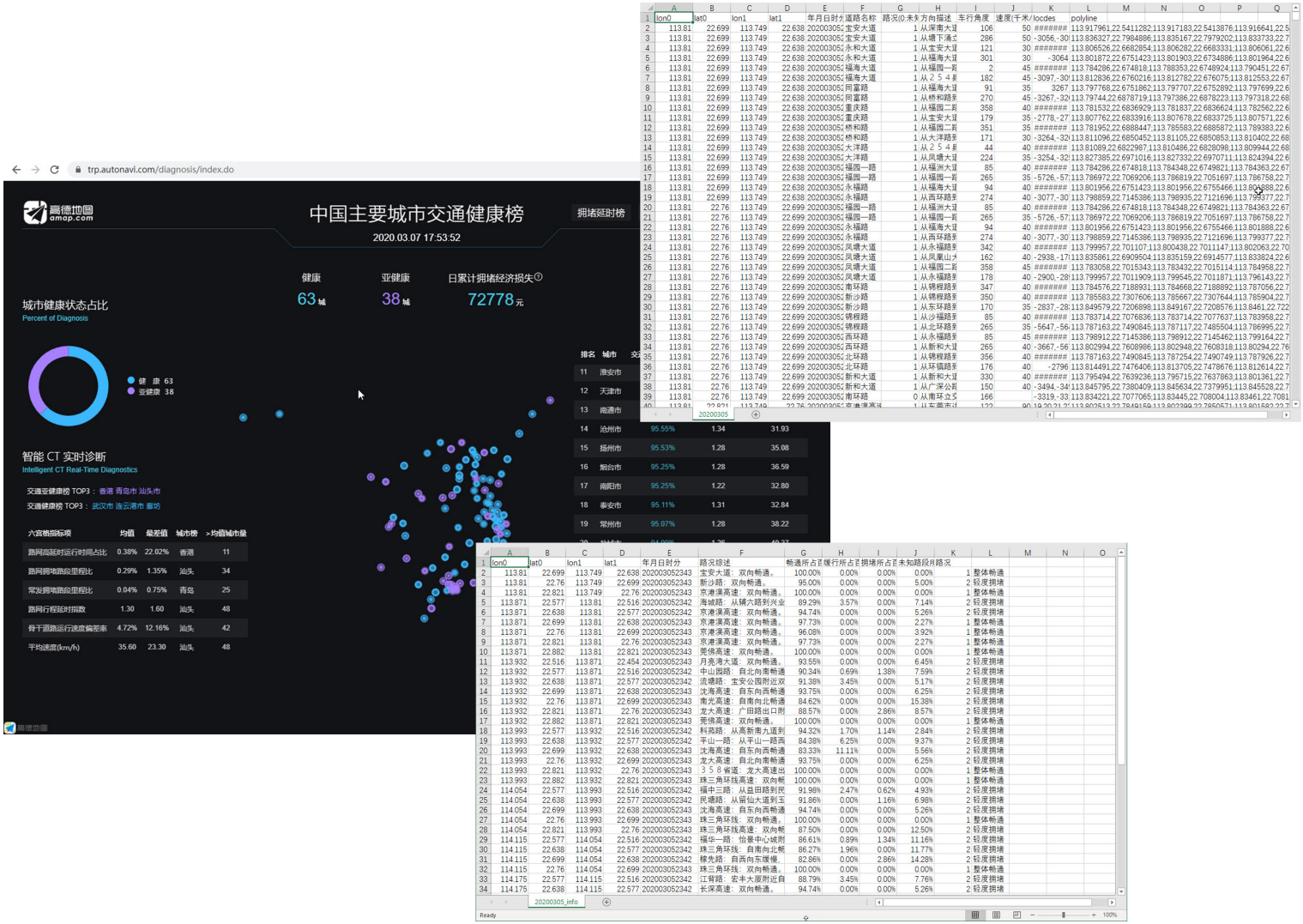
- Vector data basically encoding points, lines, and polygons in geographic space
- Shapefile is a file format specifically for geographic vector data
  - File extension is .shp
  - Other files accompany the .shp file with extensions .dbf and .prj
    - .dbf contains attribute format of shapefile
    - .prg contains information about the projection of the coordinates

Point of Interest data



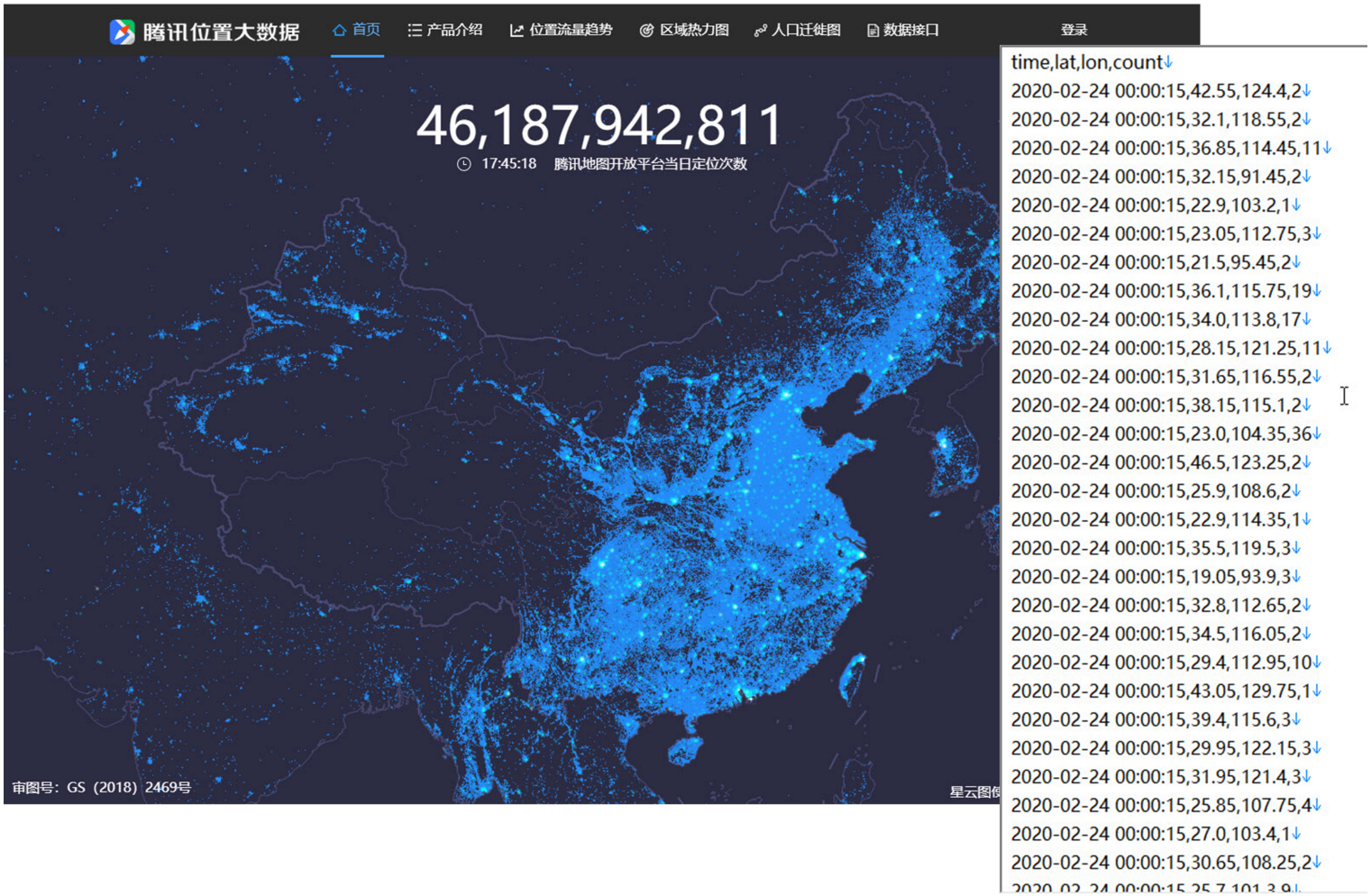
- We could scrape this data from Baidu map API

Other Vector Data Resource



- Use the Gaode transportation data API to measure population flow

Other Vector Data Resource



- Use Tecent location data (using smartphone location service information) to measure real-time population density

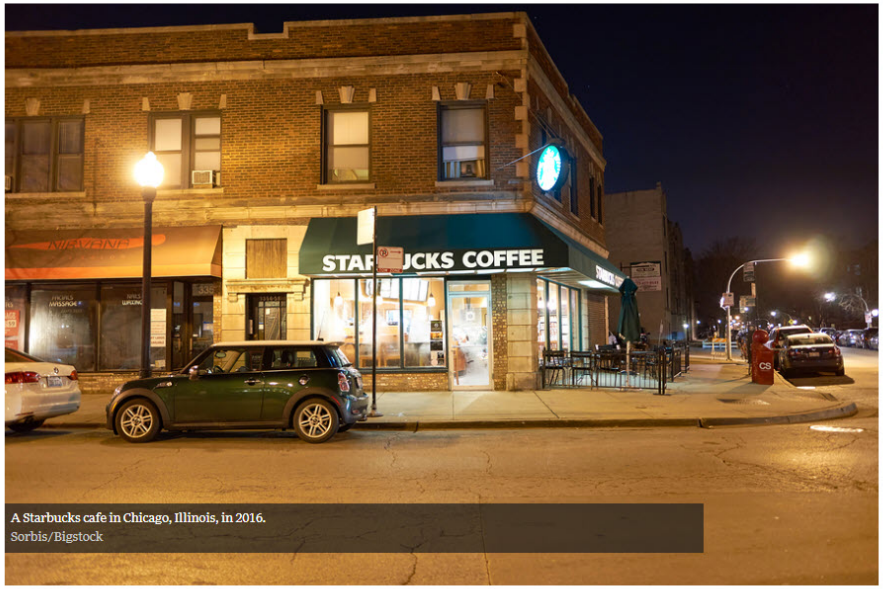
Application of POI and Dianping Data

June 11, 2018

There goes the neighborhood

Online platforms like Yelp can help economists learn about how gentrification happens in real time.

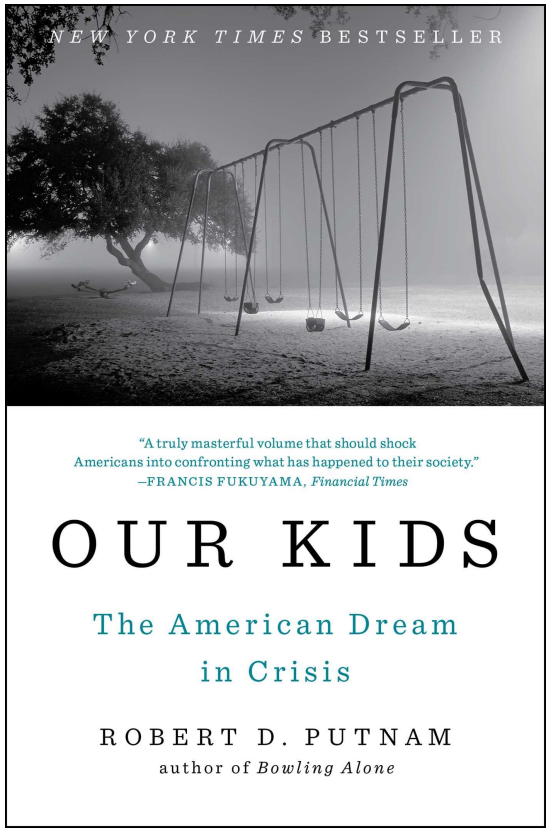
By Chris Fleisher



A Starbucks cafe in Chicago, Illinois, in 2016.
Sorbis/Bigstock

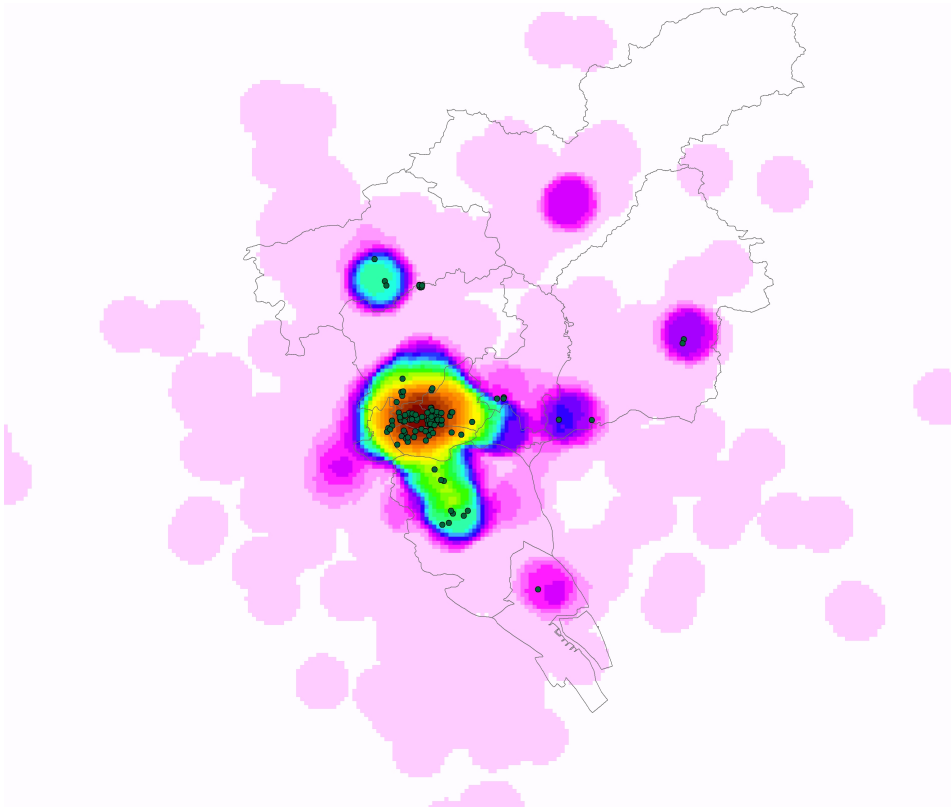
Last November, a Denver coffee shop caused a public uproar with a sidewalk sign.

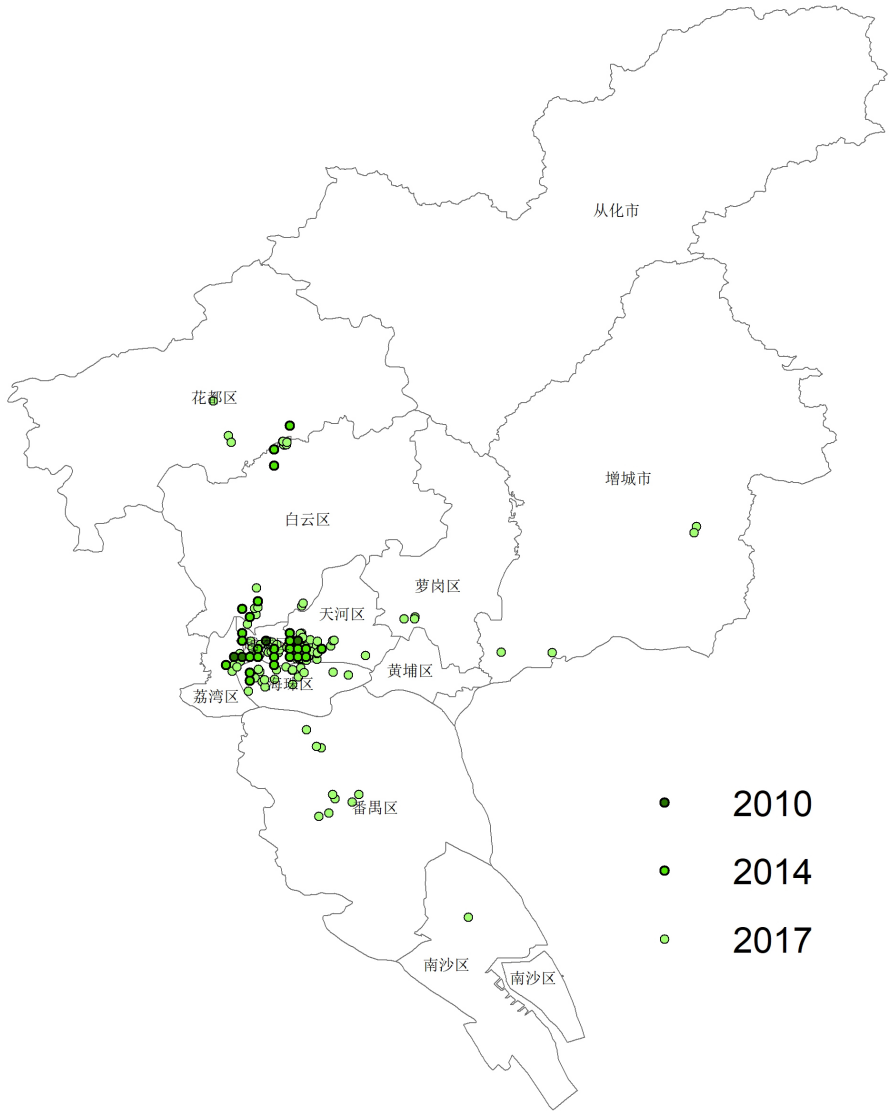
"Happily Gentrifying the Neighborhood Since 2014," the sign at Ink! Coffee said, a joke that struck a raw nerve in a rapidly changing area of the city.



- Panel data from POI or Dianping is particularly useful to measure the changing landscape of urban life
- Glaeser et al. (2017, 2018) use the location of Starbucks (from Yelp.com) to measure gentrification

Starbucks and House Prices in Guangzhou (2017)

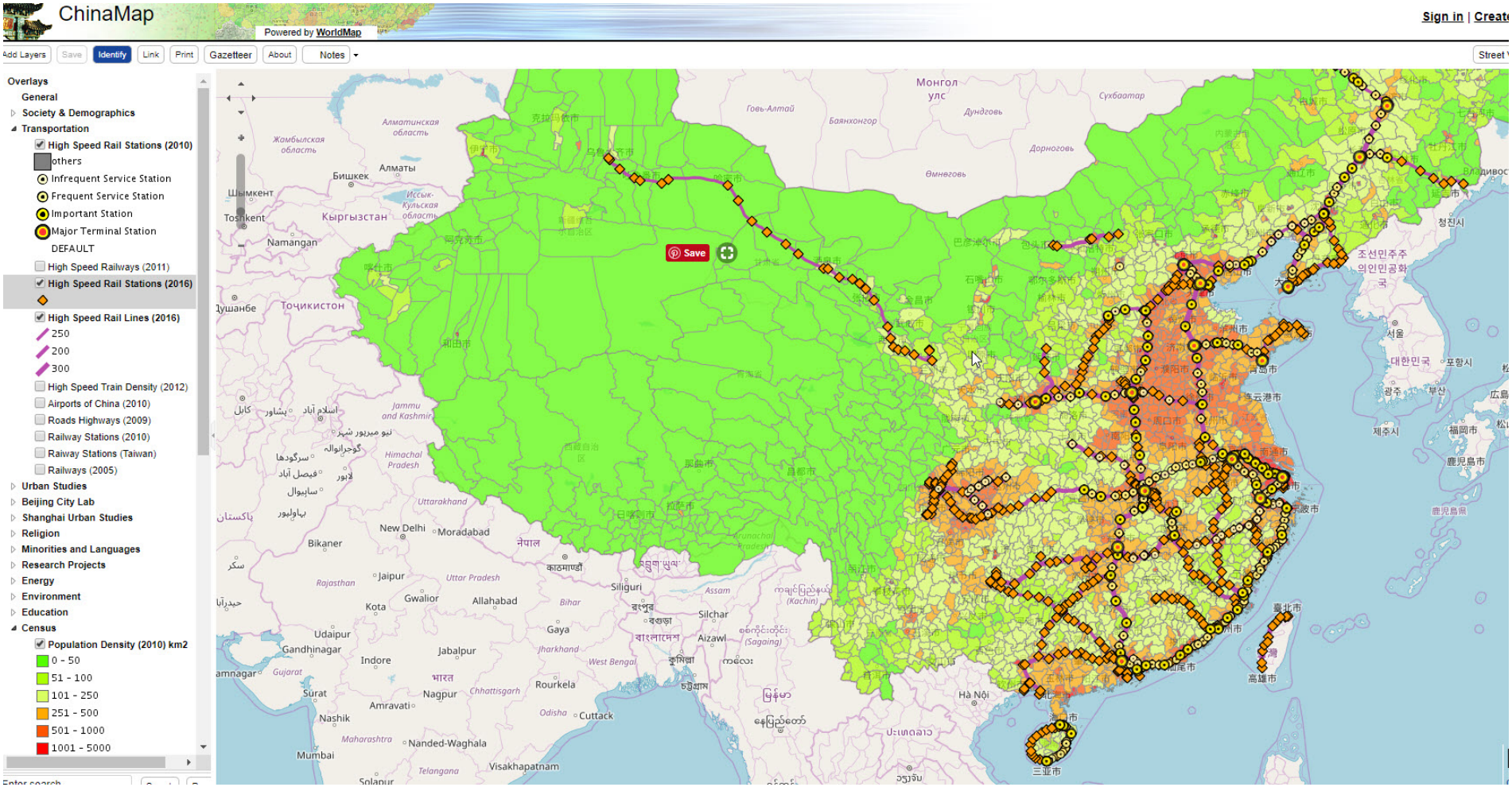




Other Spatial Data Resource

- FAO Geonetwork
- Geocommons
- Earth Explorer
- Devecondata
- Harvard China GIS

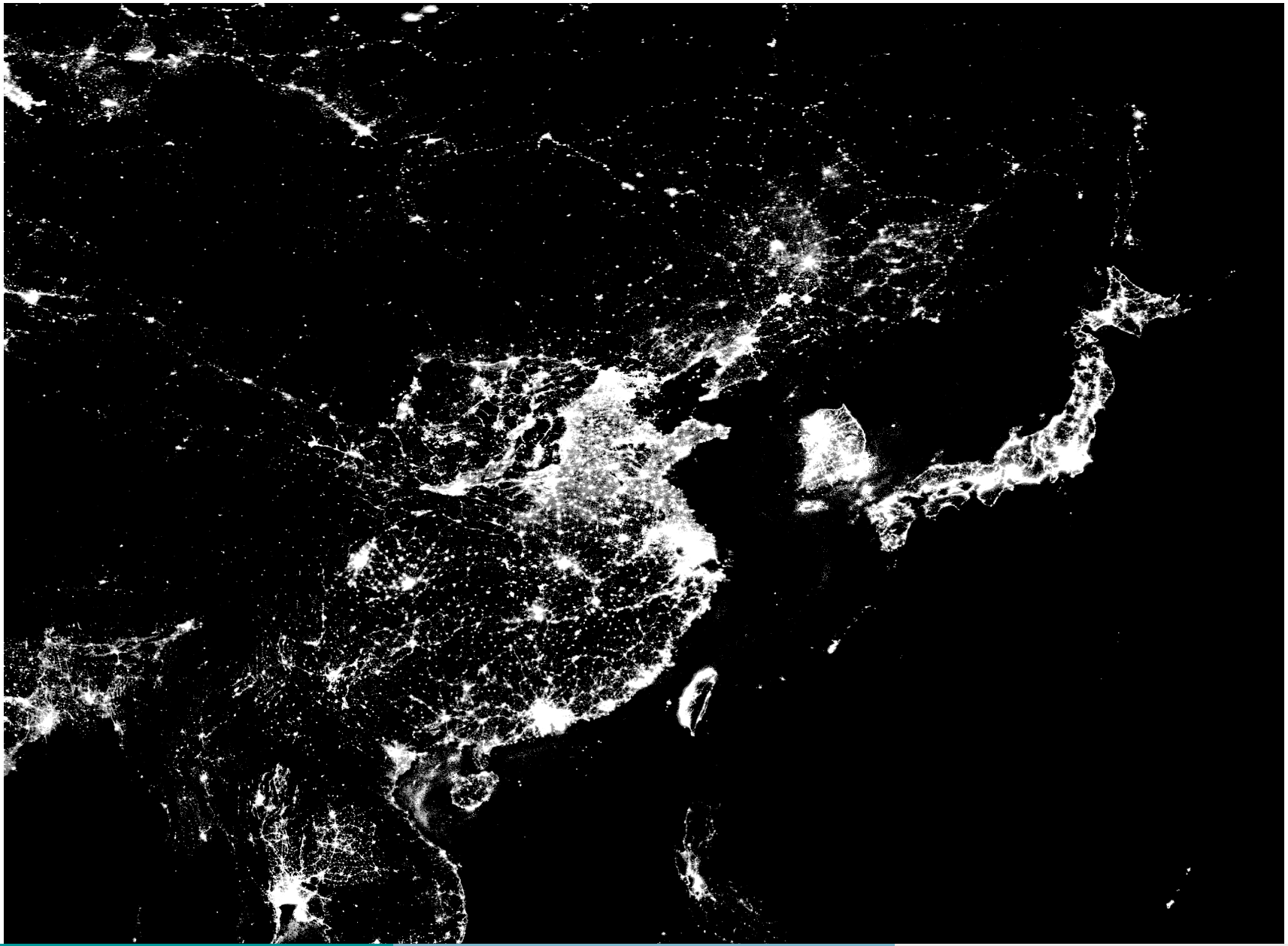
Other Vector Data Resource



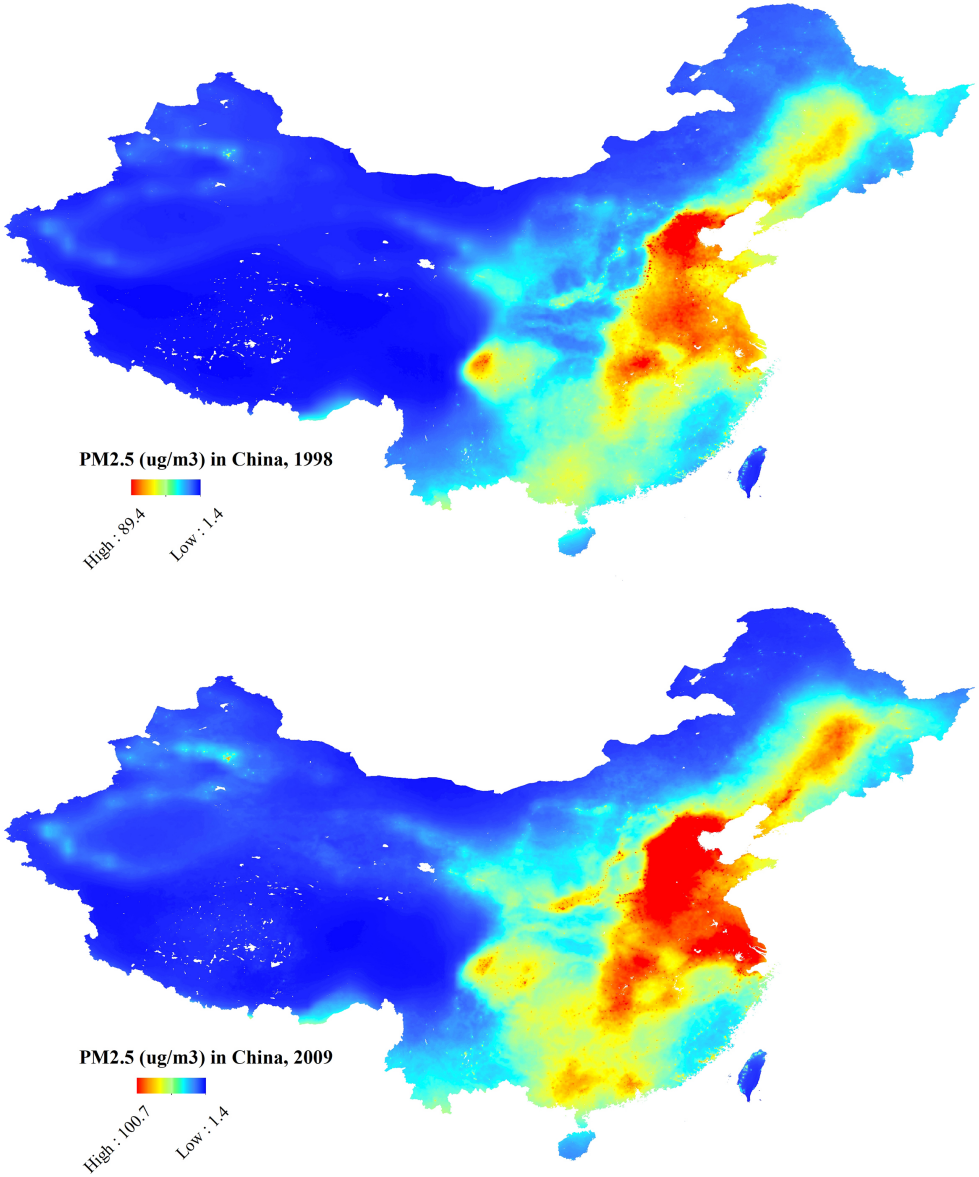
Raster Data

- Files format include: ESRI grid, TIFF, can be transferred from ASCII or NetCDF
- Examples of raster data
  - Elevation
  - GAZE Agricultural Suitability Data
  - DMSP-OLS Nighttime Light Data
  - Weather and pollution data from remote sensing pictures
    - Globe Summary Of Days
    - Terrestrial Air Temperature: 1900-2010 Gridded Monthly Time Series (v 3.01)
    - Global Annual Average PM2.5 Grids from MODIS and MISR Aerosol

Nighttime Light Data

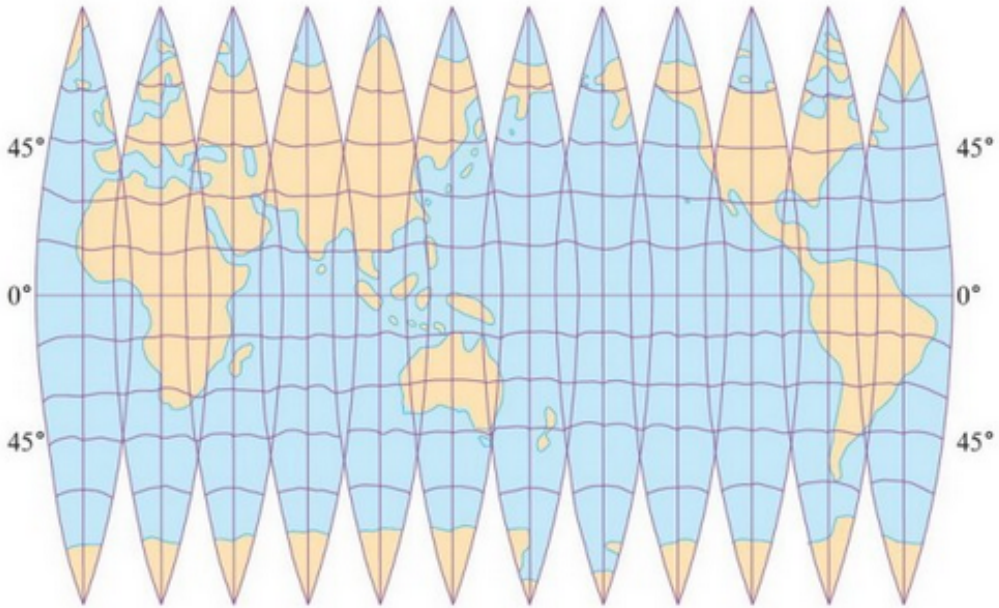


Pollution Data



Map Projections

- Define the spatial relationship between locations on earth and their relative locations on a flat map
- Are mathematical expressions
- Cause the distortion of one or more map properties (scale, distance, direction, shape)
- Each projection distorts size of area, distance in different way

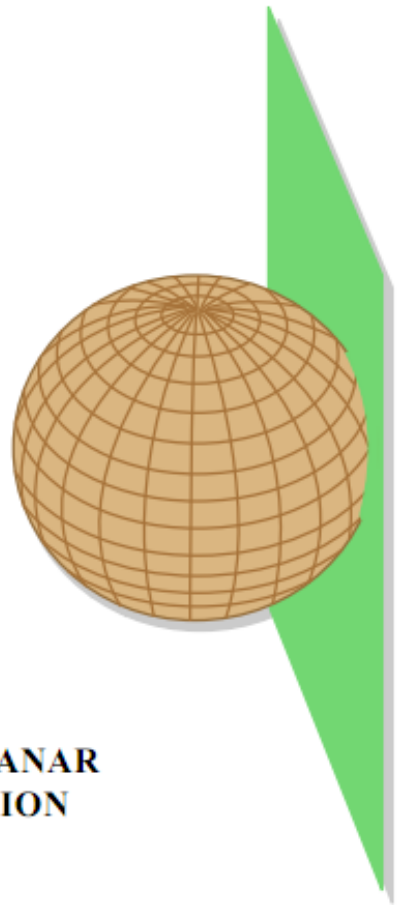


Classification System

- Planar
- Cylindrical
- Conic

Planar Surface

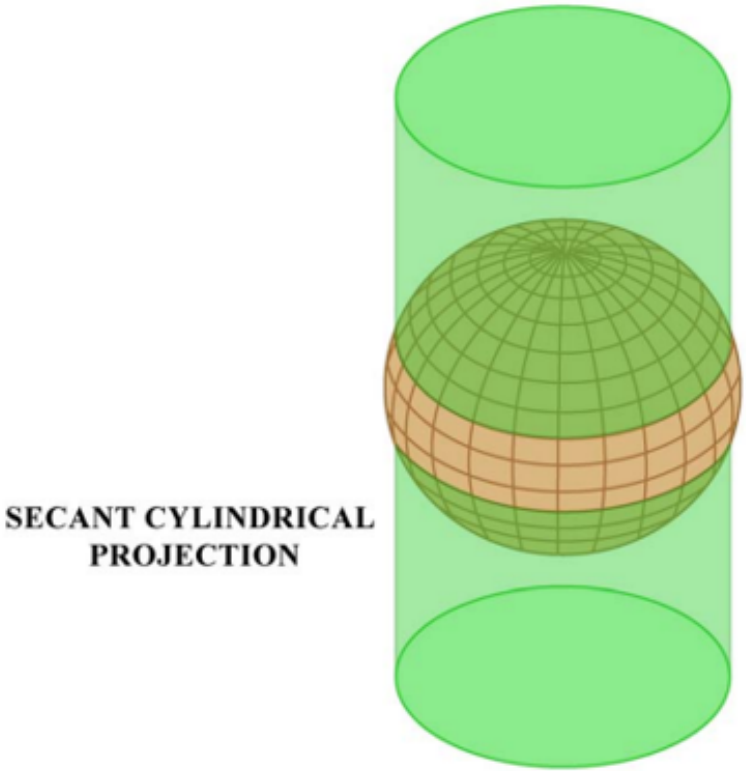
- Earth intersects the plane on a small circle. All points on circle have no scale distortion.



SECANT PLANAR
PROJECTION

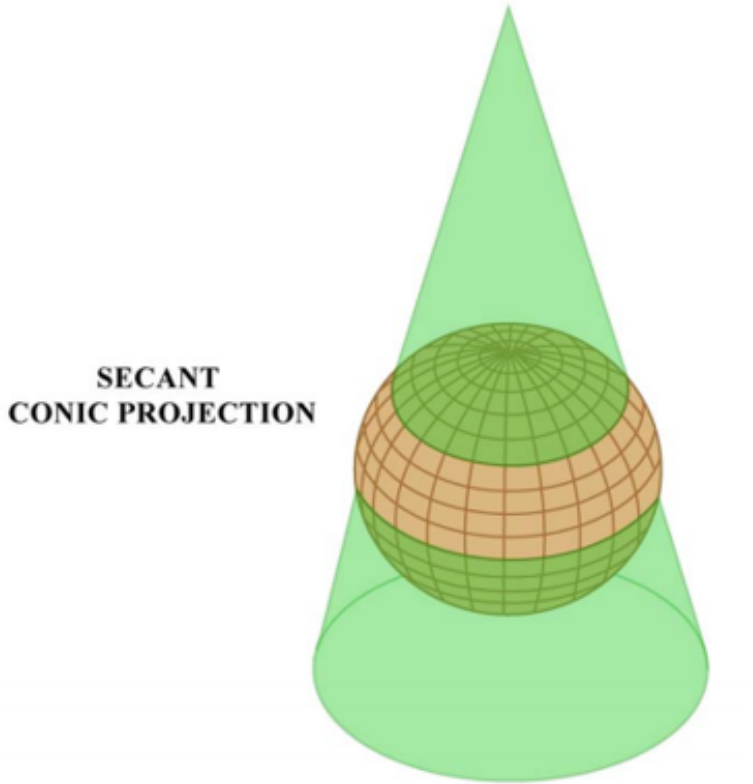
Cylindrical Surface

- Earth intersects the cylinder on two smallcircles. All points along both circles have no scale distortion.



Conic Surface

- Earth intersects the cone at two circles. All points along both circles have no scale distortion.



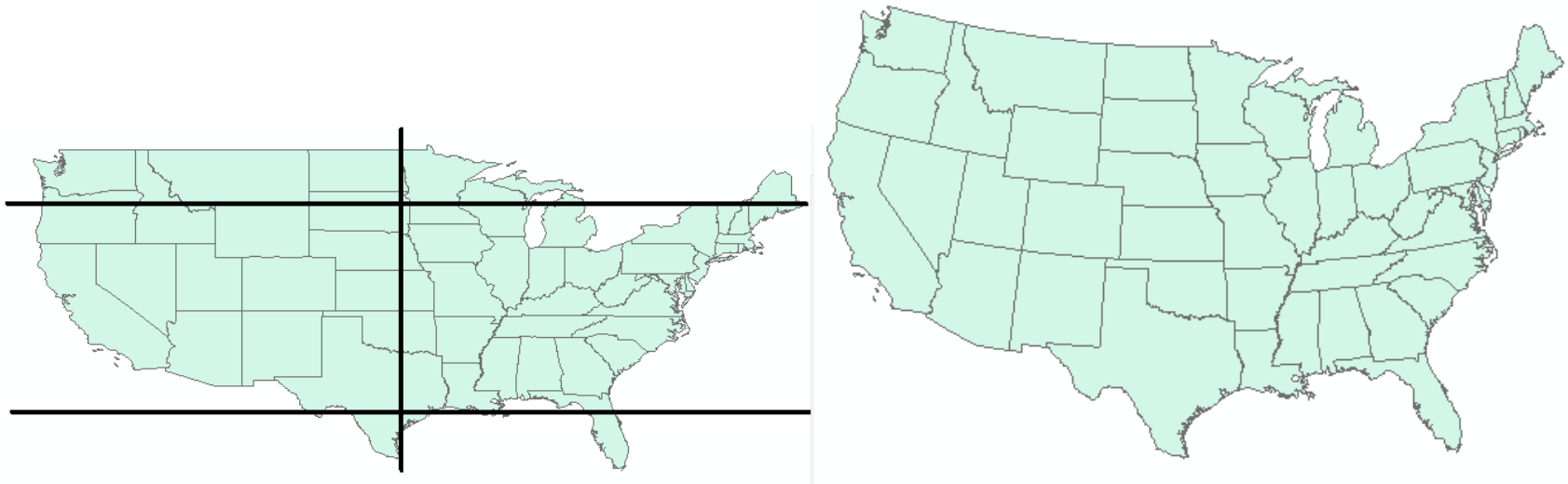
Scale Distortion

- Scale near intersections with surface are accurate
- Scale between intersections is too small
- Scale outside of intersections is too large and gets excessively large the further one goes beyond the intersections

Why project data?

- Data often comes in geographic, or spherical coordinates (latitude and longitude) and can't be used for area calculations in most GIS software applications
- Some projections work better for different parts of the globe giving more accurate calculations

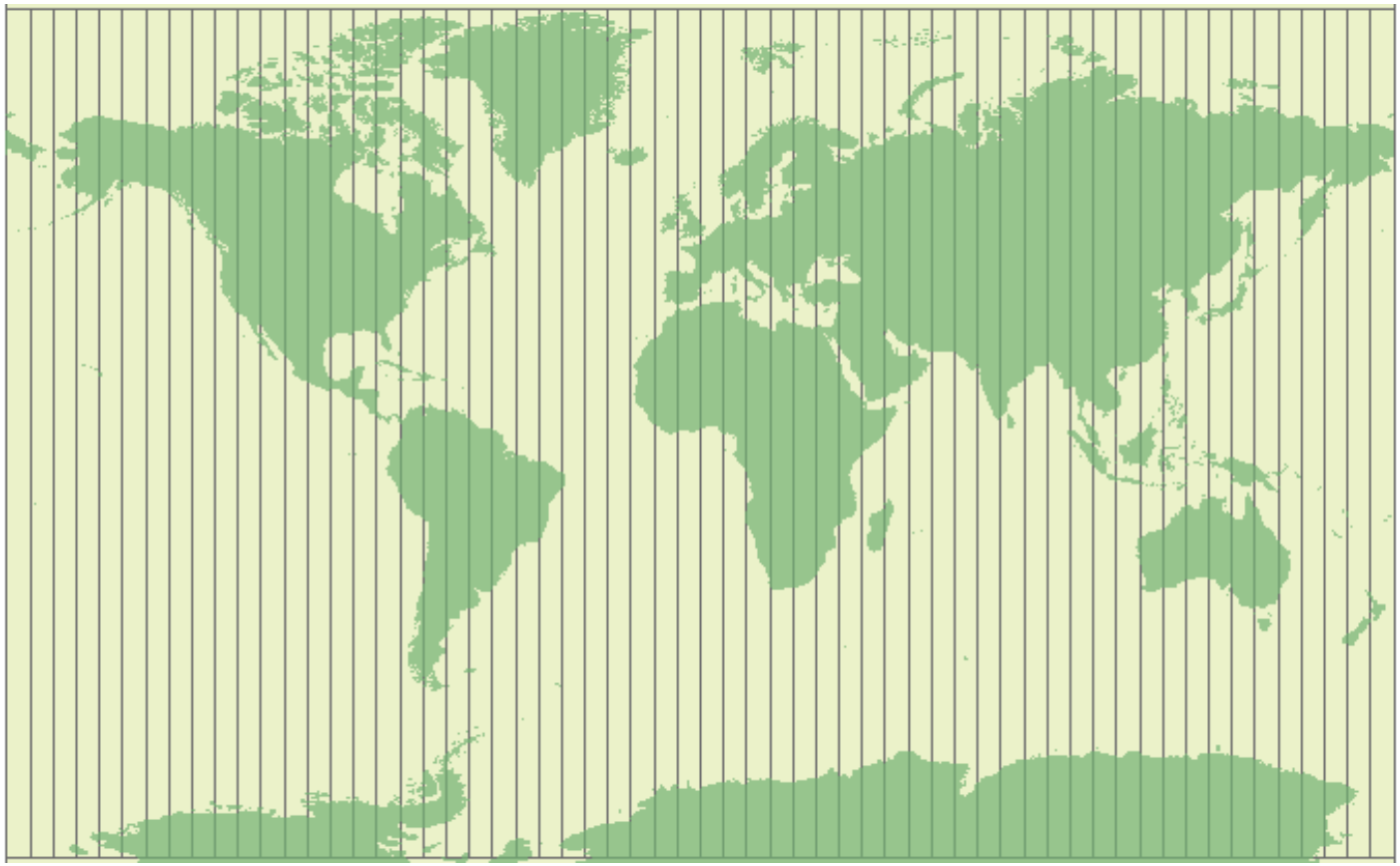
Conic projection for US



How to Choose Projections

- Generally, follow the lead of people who make maps of the area you are interested in. Look at maps!
- UTM is commonly used and is a good choice when the east-west width of area does not exceed 6 degrees
- The most commomly used—World Geodetic System: WGS84 (EPSG:4326)

UTM projection



The **sf** package

- There are two general packages to provide classes and methods for spatial data in R: the **sp** and **sf**
- Since **sf** is a more updated package with some nice properties and compatibility with **ggplot2** etc., we will focus on **sf**
  - **sf** implements a formal standard called *"Simple Features"* that specifies a storage and access model of spatial geometries (point, line, polygon)
  - It is *simple* in the sense that it consists of points connected by straight line pieces, and does not intersect itself
  - Such standard has been widely adopted by different spatial database (e.g. PostGIS) and more recent standards (e.g. GeoJSON)

The **sf** package (Cont'd)

- In **sf** spatial objects are stored as a simple data frame with a special column that contains the information for the geometry coordinates
- That column is a list with the same length as the number of rows in the data frame
- Normally before drawing map, we need a shapefile as the boundary map, you can either:
  - Use the maps loaded with package
  - Bring in external data in the form of shapefiles
  - Create a spatial object from a lat/lon table
- Then you can add attributes to the shapefiles by attribut join or spatial join

Boundary map

- First step: drawing boundaries
  - Using **ne_countries()** from the **rnaturalearth** package to get the polygons of world country boundaries

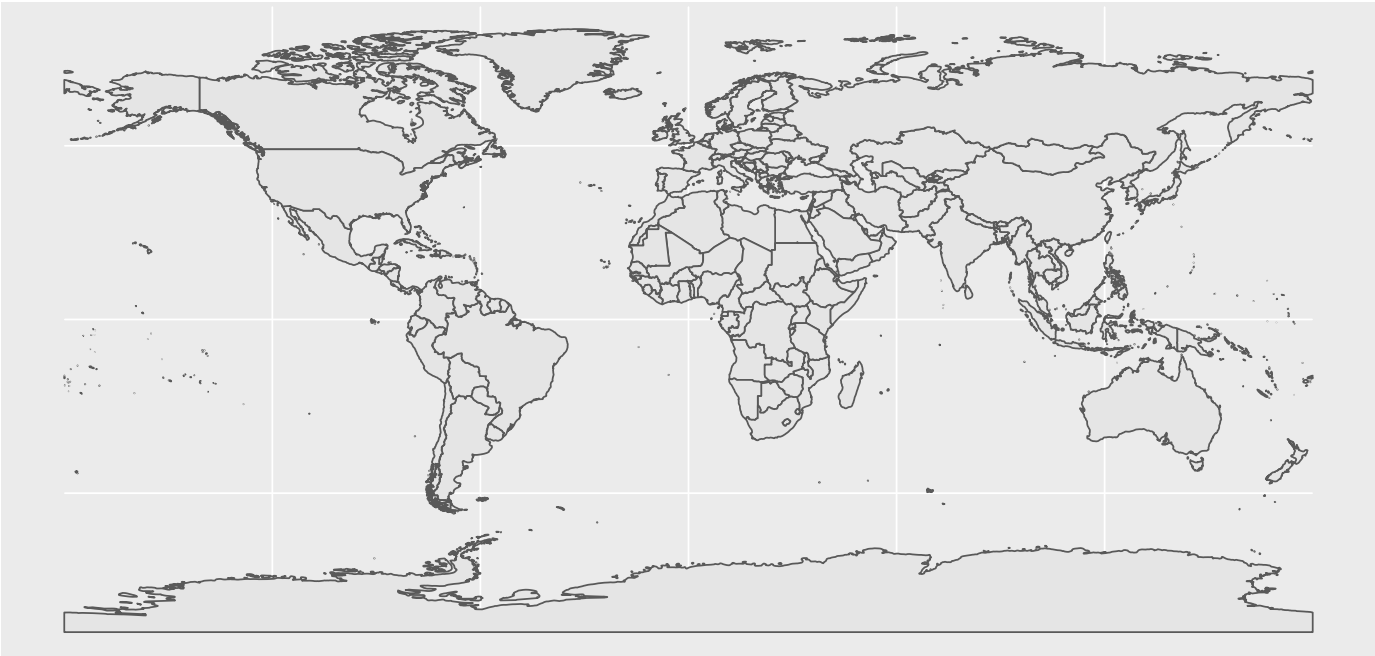
```
category   cultural
                              type scale110 scale50 scale10
1                        countries     TRUE    TRUE    TRUE
2                        map_units     TRUE    TRUE    TRUE
3                     map_subunits    FALSE    TRUE    TRUE
4                      sovereignty     TRUE    TRUE    TRUE
5                   tiny_countries     TRUE    TRUE    TRUE
6                           states    FALSE    TRUE    TRUE
7                 populated_places     TRUE    TRUE    TRUE
8               boundary_lines_land     TRUE    TRUE    TRUE
9                pacific_groupings     TRUE    TRUE    TRUE
10         breakaway_disputed_areas    FALSE    TRUE    TRUE
11    boundary_lines_disputed_areas    FALSE    TRUE    TRUE
12 boundary_lines_maritime_indicator    FALSE    TRUE    TRUE
13                        airports    FALSE    TRUE    TRUE
14                           ports    FALSE    TRUE    TRUE
15                      urban_areas    FALSE    TRUE    TRUE
16                           roads    FALSE   FALSE    TRUE
17                        railroads    FALSE   FALSE    TRUE
```

```
category   physical
                              type scale110 scale50 scale10
1                        coastline     TRUE    TRUE    TRUE
2                             land     TRUE    TRUE    TRUE
3                            ocean     TRUE    TRUE    TRUE
4          rivers_lake_centerlines     TRUE    TRUE    TRUE
5                            lakes     TRUE    TRUE    TRUE
6                  glaciated_areas     TRUE    TRUE    TRUE
7    antarctic_ice_shelves_polys     TRUE    TRUE    TRUE
8                geographic_lines     TRUE    TRUE    TRUE
9                    graticules_1     TRUE    TRUE    TRUE
10                   graticules_5     TRUE    TRUE    TRUE
11                  graticules_10     TRUE    TRUE    TRUE
12                  graticules_15     TRUE    TRUE    TRUE
13                  graticules_20     TRUE    TRUE    TRUE
14                  graticules_30     TRUE    TRUE    TRUE
15             wgs84_bounding_box     TRUE    TRUE    TRUE
16                         playas    FALSE    TRUE    TRUE
17                 minor_islands    FALSE   FALSE    TRUE
18                         reefs    FALSE   FALSE    TRUE
```

Boundary Map (Cont'd)

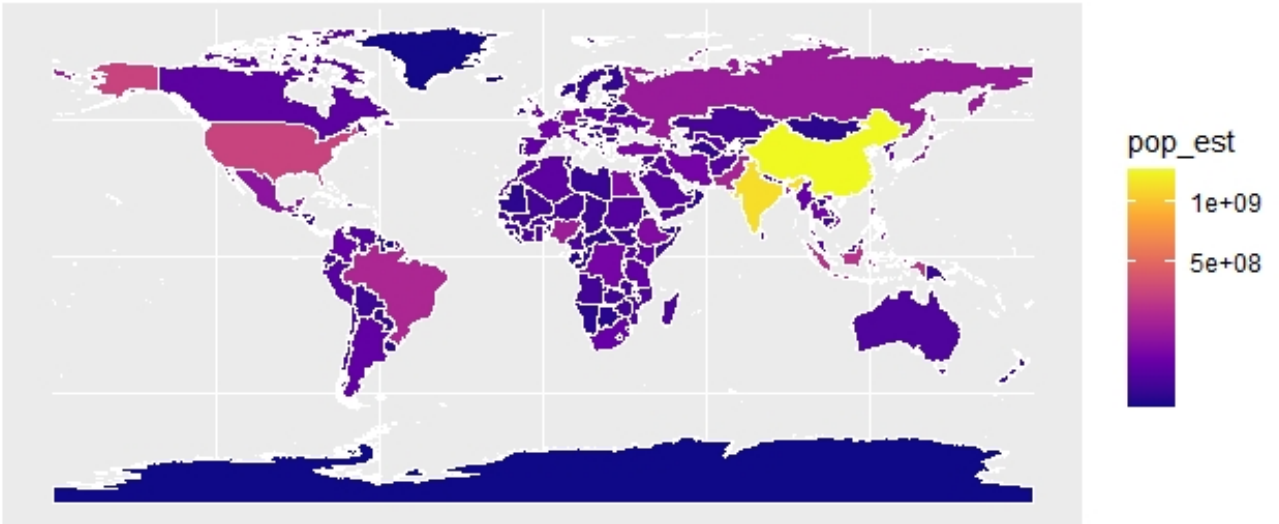
```
# Load data from sf library
world <- ne_countries(scale = "medium", returnclass = "sf")
class(world)

# Data and basic plot (ggplot and geom_sf)
ggplot(data = world) +
  geom_sf()
```



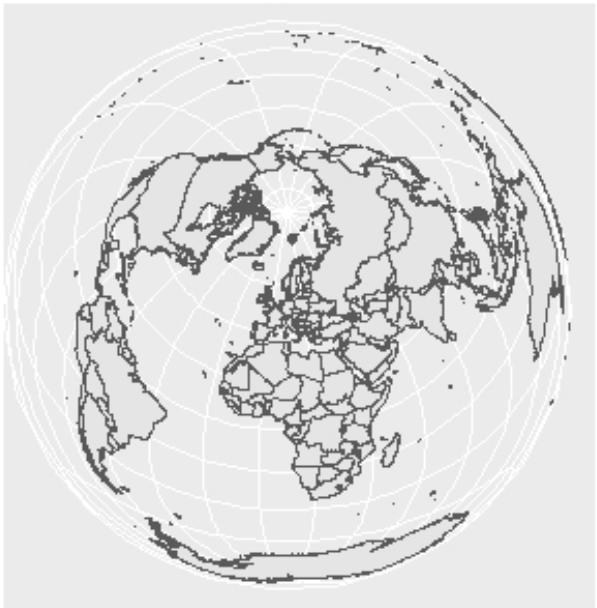
Add Title, subtitle, and axis labels, Change Color

```
29 # Add Title, subtitle, and axis labels (ggtitle, xlab, ylab)
30 ggplot(data = world) +
31   geom_sf() +
32   xlab("Longitude") + ylab("Latitude") +
33   ggtitle("World map", subtitle = paste0("(", length(unique(world$name)), "
   countries)"))
34
35 # Map color (geom_sf)
36 ggplot(data = world) +
37   geom_sf(color = "black", fill = "lightgreen")
38
39 ggplot(data = world) +
40   geom_sf(aes(fill = pop_est), color="white", size=0.5) +
41   scale_fill_viridis_c(option = "plasma", trans = "sqrt")
```



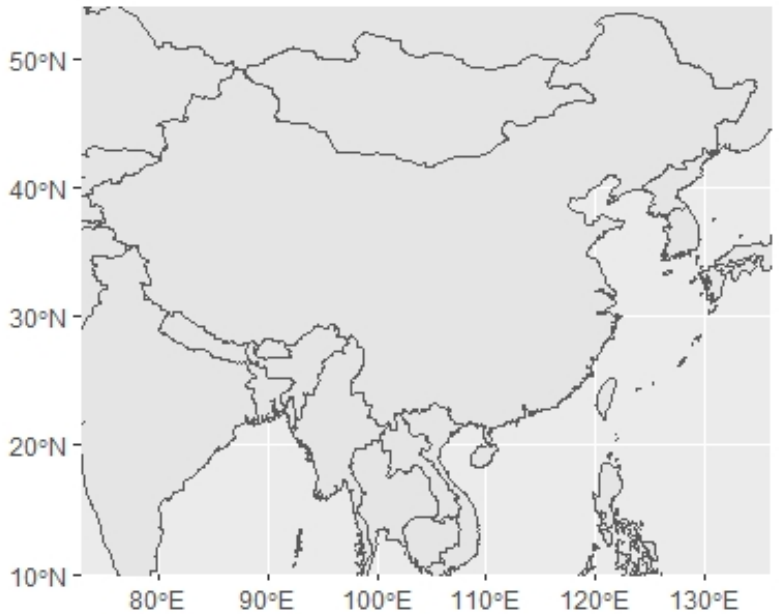
Change Projection and Extent using **coord_sf()**

```
43  # Projection and extent (coord_sf)
44  ggplot(data = world) +
45    geom_sf() +
46    coord_sf(crs = "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
    +ellps=GRS80 +units=m +no_defs ")
47
48  ggplot(data = world) +
49    geom_sf() +
50    coord_sf(crs = "+init=epsg:3035")
51
52  ggplot(data = world) +
53    geom_sf() +
54    coord_sf(crs = st_crs(3035))
```



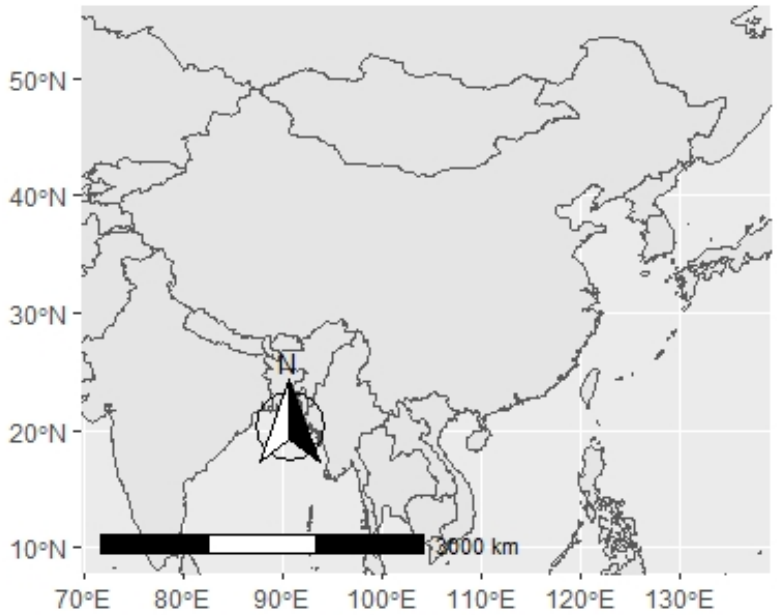
Restrict to China

```
56  # Restrict to China
57  ggplot(data = world) +
58    geom_sf() +
59    coord_sf(xlim = c(73, 136), ylim = c(10, 54), expand = FALSE)
```



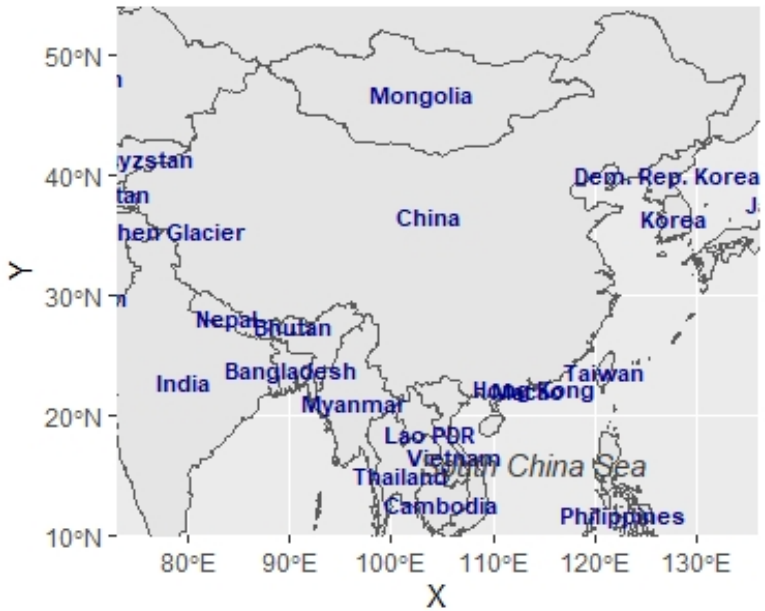
Add Scale bar and North arrow using **ggspatial**

```
61 # Scale bar and North arrow (package ggspatial)
62 library("ggspatial")
63 ggplot(data = world) +
64   geom_sf() +
65   annotation_scale(location = "bl", width_hint = 0.5) +
66   annotation_north_arrow(location = "bl", which_north = "true",
67                          pad_x = unit(0.75, "in"), pad_y = unit(0.5, "in"),
68                          style = north_arrow_fancy_orienteering) +
69   coord_sf(xlim = c(73, 136), ylim = c(10, 54))
```



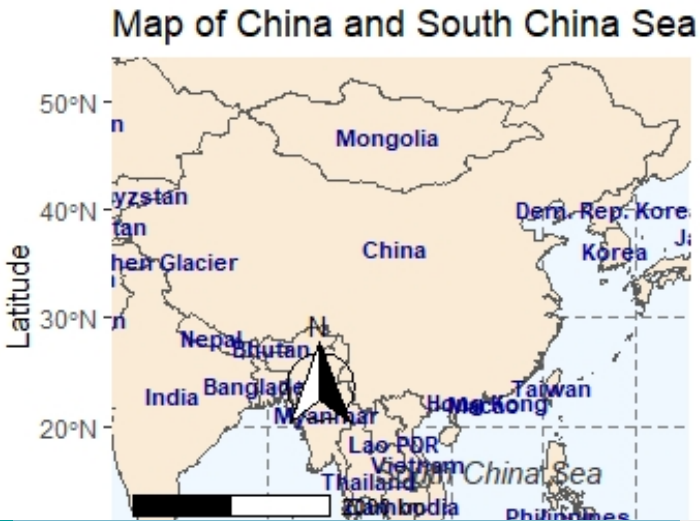
Add Country Names and Other using **geom_text** and **annotate**

```
71 # Country names and other names (geom_text and annotate)
72 library("sf")
73 world_points<- st_centroid(world)
74 world_points <- cbind(world, st_coordinates(st_centroid(world$geometry)))
75
76 ggplot(data = world) +
77   geom_sf() +
78   geom_text(data= world_points,aes(x=X, y=Y, label=name),
79             color = "darkblue", check_overlap = FALSE, fontface = "bold",
   size=3) +
80   annotate(geom = "text", x = 114, y = 16, label = "South China Sea",
81            fontface = "italic", color = "grey22", size = 4) +
82   coord_sf(xlim = c(73, 136), ylim = c(10, 54), expand = FALSE)
```



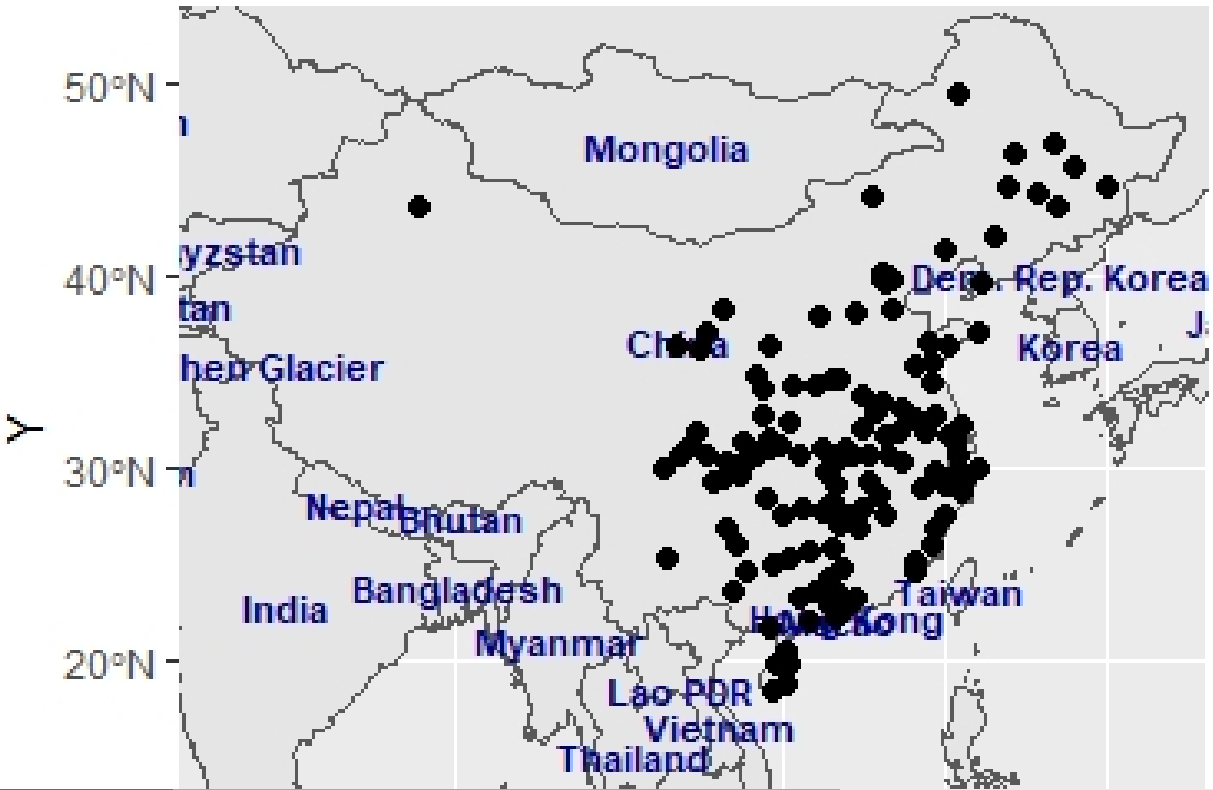
More Decoration and Save Map

```
85  # Final map
86  ggplot(data = world) +
87    geom_sf(fill= "antiquewhite") +
88    geom_text(data= world_points,aes(x=X, y=Y, label=name),
89              color = "darkblue", check_overlap = FALSE, fontface = "bold",
    size=3) +
90    annotate(geom = "text", x = 114, y = 16, label = "South China Sea",
91             fontface = "italic", color = "grey22", size = 4) +
92    annotation_scale(location = "bl", width_hint = 0.5) +
93    annotation_north_arrow(location = "bl", which_north = "true", pad_x =
    unit(0.75, "in"), pad_y = unit(0.5, "in"), style = north_arrow_fancy_orient
    eering) +
94    coord_sf(xlim = c(73, 136), ylim = c(10, 54), expand = FALSE)+
95    xlab("Longitude") +
96    ylab("Latitude") +
97    ggtitle("Map of China and South China Sea") +
98    theme(panel.grid.major = element_line(color = gray(.5), linetype =
    "dashed", size = 0.5), panel.background = element_rect(fill = "aliceblue"))
99
100 # Saving the map with ggsave
101 ggsave("china_map.pdf")
102 ggsave("china_map.png", width = 6, height = 6, dpi = "screen")
```



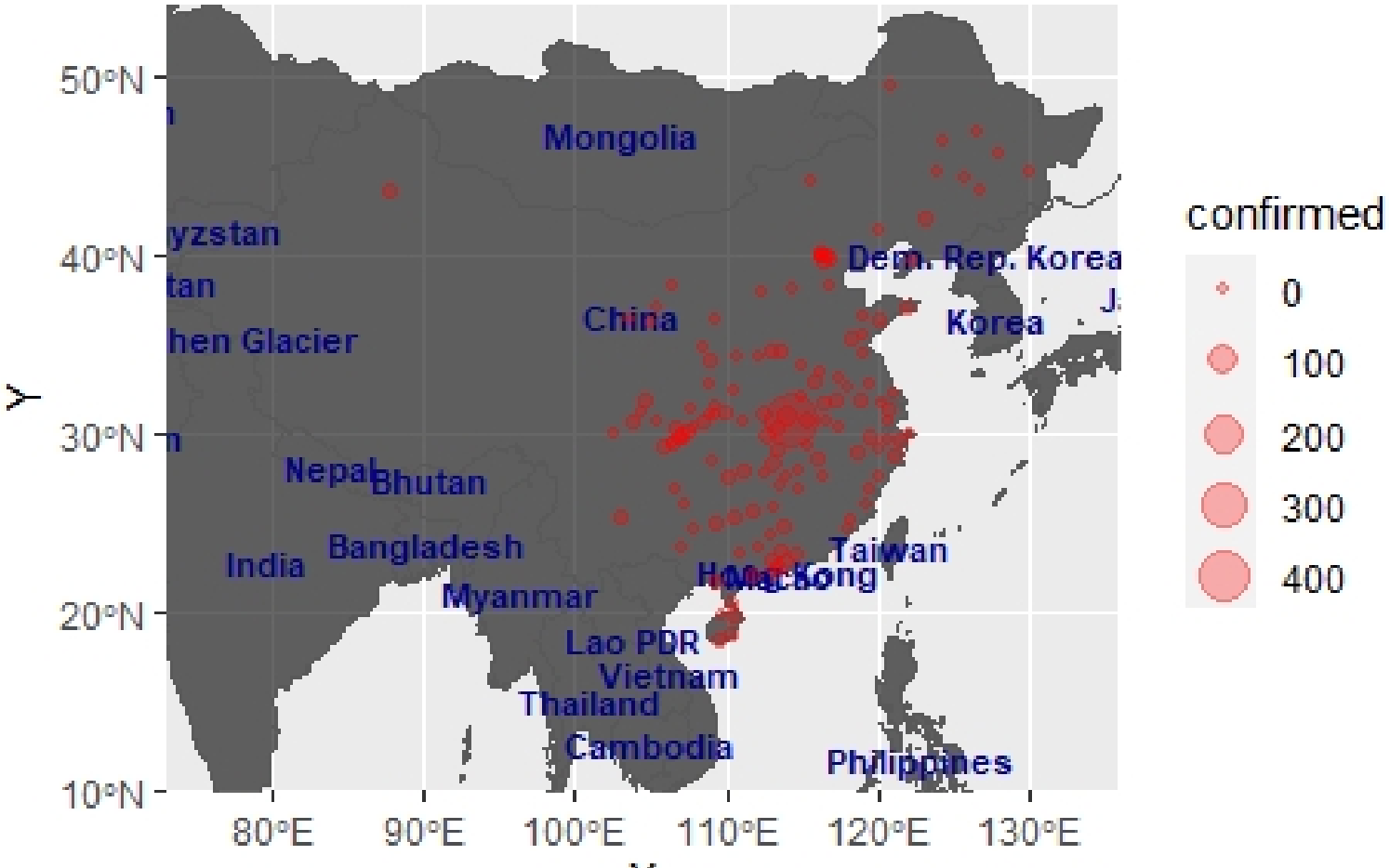
Location Map

```
109 # Field sites (point data)
110 library(foreign)
111 coronavirus  <- read.dta("coronavirus_china_data.dta")
112 sites <- subset(coronavirus, date=="2020-01-23" & !is.na(coronavirus$x))
113
114 ggplot(data = world) +
115   geom_sf() +
116   geom_text(data= world_points,aes(x=X, y=Y, label=name),
117             color = "darkblue", check_overlap = FALSE, fontface = "bold",
    size=3) +
118   geom_point(data=sites, aes(x=x, y=y), size=2) +
119   coord_sf(xlim = c(73, 136), ylim = c(10, 54), expand = FALSE)
```



More Decoration on Location Map

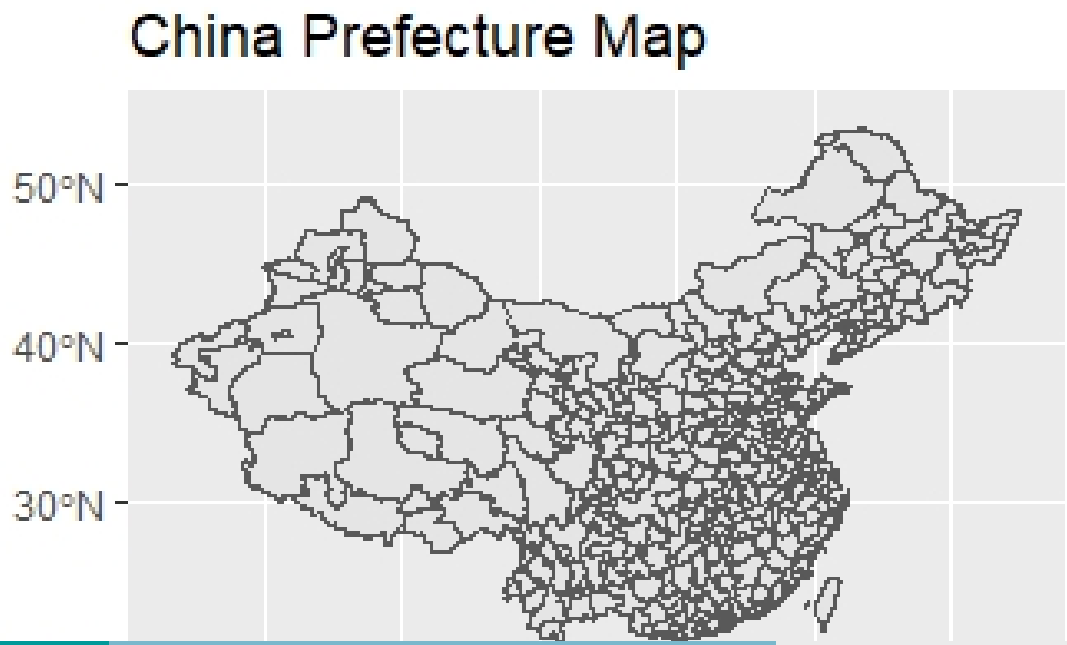
```
ggplot(data = world) +
  geom_sf(fill="black", alpha=0.6) +
  geom_text(data= world_points,aes(x=X, y=Y, label=name),
            color = "darkblue", check_overlap = FALSE, fontface = "bold",
size=3) +
  geom_point(data=sites, aes(x=x, y=y, size=confirmed), color="red", alpha
=0.3) +
  coord_sf(xlim = c(73, 136), ylim = c(10, 54), expand = FALSE)
```



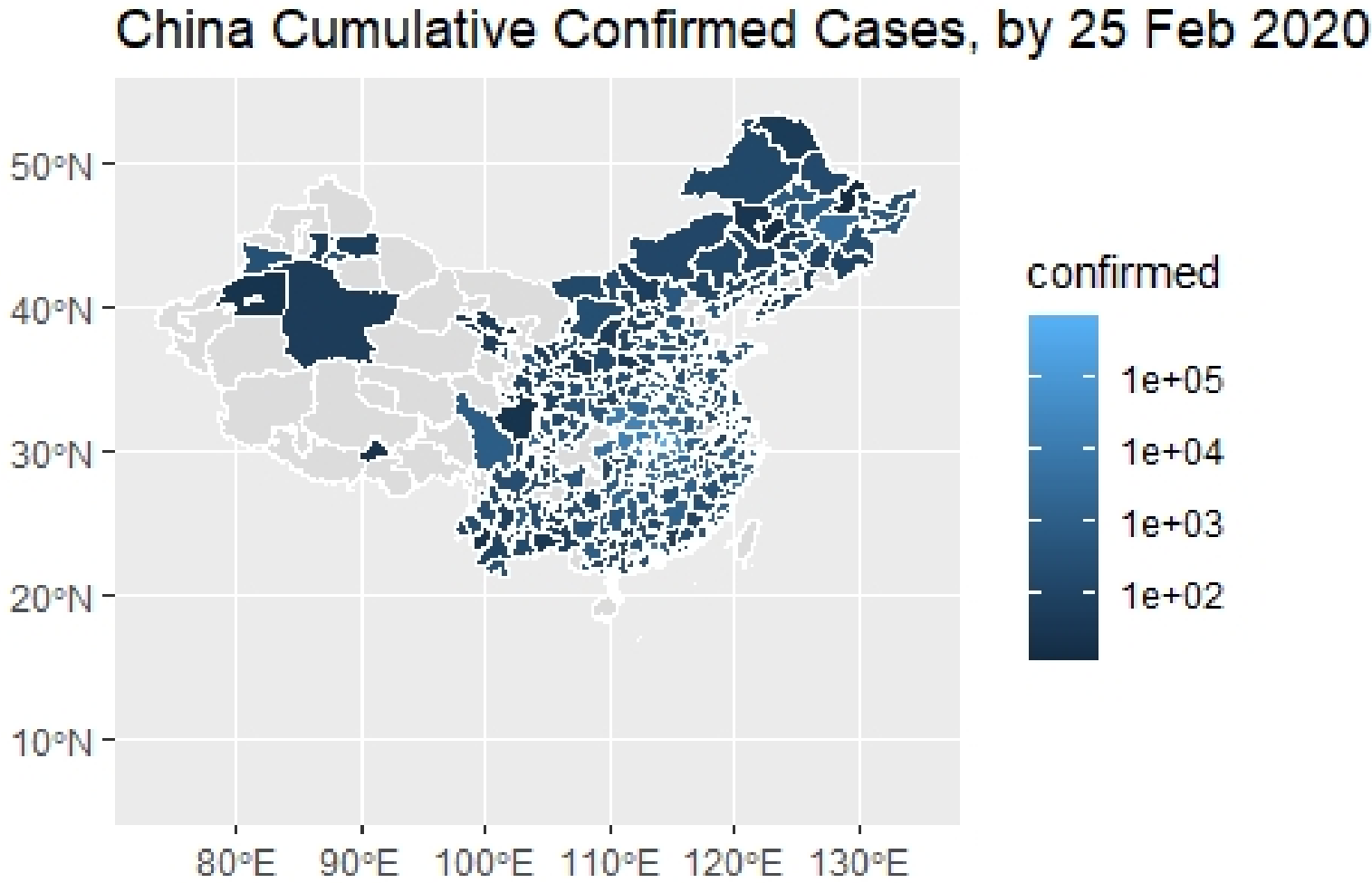
Import a Local Shapefile

- Plenty of sources online to download administartive boundary map, e.g.
  - https://www.diva-gis.org/gdata
  - https://docs.gmt-china.org/latest/dataset/gadm/index.html
  - 中国国家基础地理信息数据)

```
141 unzip("city2007.zip")
142
143 chinacity <- st_read("city2007\\city_region.shp")
144 chinacity
145
146 ggplot() +
147   geom_sf(data=chinacity) +
148   ggtitle("China Prefecture Map") +
149   coord_sf()
```



- An Execise to create a prefecture map on cumulative confirmed cases in China



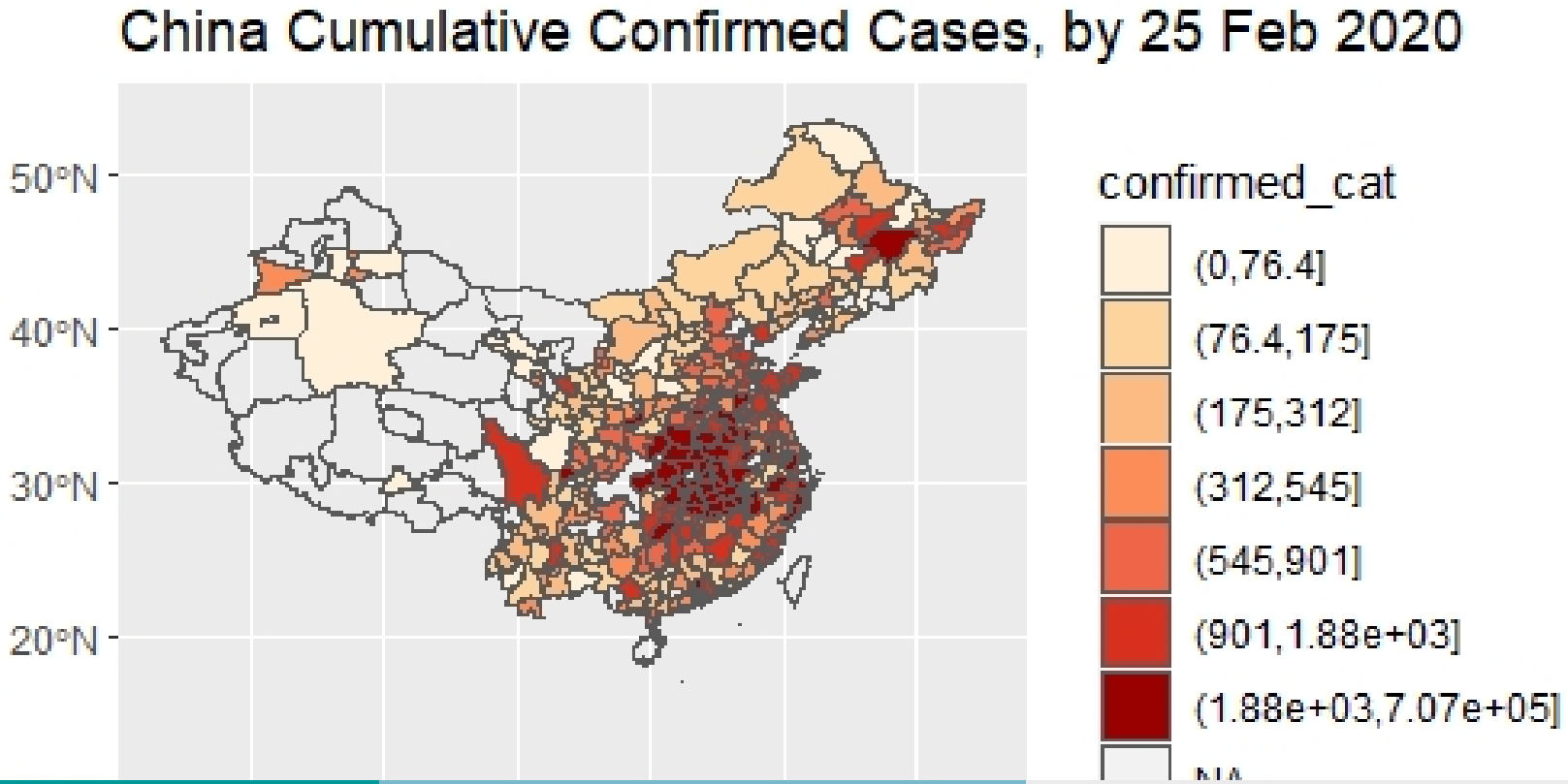
Import a Local Shapefile

- The confirmed case is a continuous variable and is plotted by **ggplot** as such
- A ”true” choropleth map we need to change the continuous vairable into a categoriacal one based on e.g. quantile in two steps:
  - Determine the quantile breaks
  - Add a categorical variable to the object which assigns each continious vaule to a bracket

```
220 # If we wanted to plot our map as a 'true' choropleth map we need to
    convert our continouse variable into a categoriacal one, according to
    whichever brackets we want to use
221 # This requires two steps:
222   ## Determine the quantile breaks;
223   ## Add a categorical variable to the object which assigns each continious
    vaule to a bracket
224 library(dplyr)
225 library(classInt)
226 breaks_qt <- classIntervals(c(min(chinacity2$confirmed) - .00001, chinacity
    2$confirmed), n = 7, style = "quantile")
227 breaks_qt
228
229 # We use cut to divice homic_rate into intervals and code them according to
    which interval they are in
230 # Lastly, we can use scale_fill_brewer() and add our color palette
231 chinacity2 <- mutate(chinacity2, confirmed_cat = cut(confirmed, breaks_qt$b
    rks))
232
233 ggplot(chinacity2) +
234   geom_sf(aes(fill=confirmed_cat)) +
235   scale_fill_brewer(palette = "OrRd") +
```

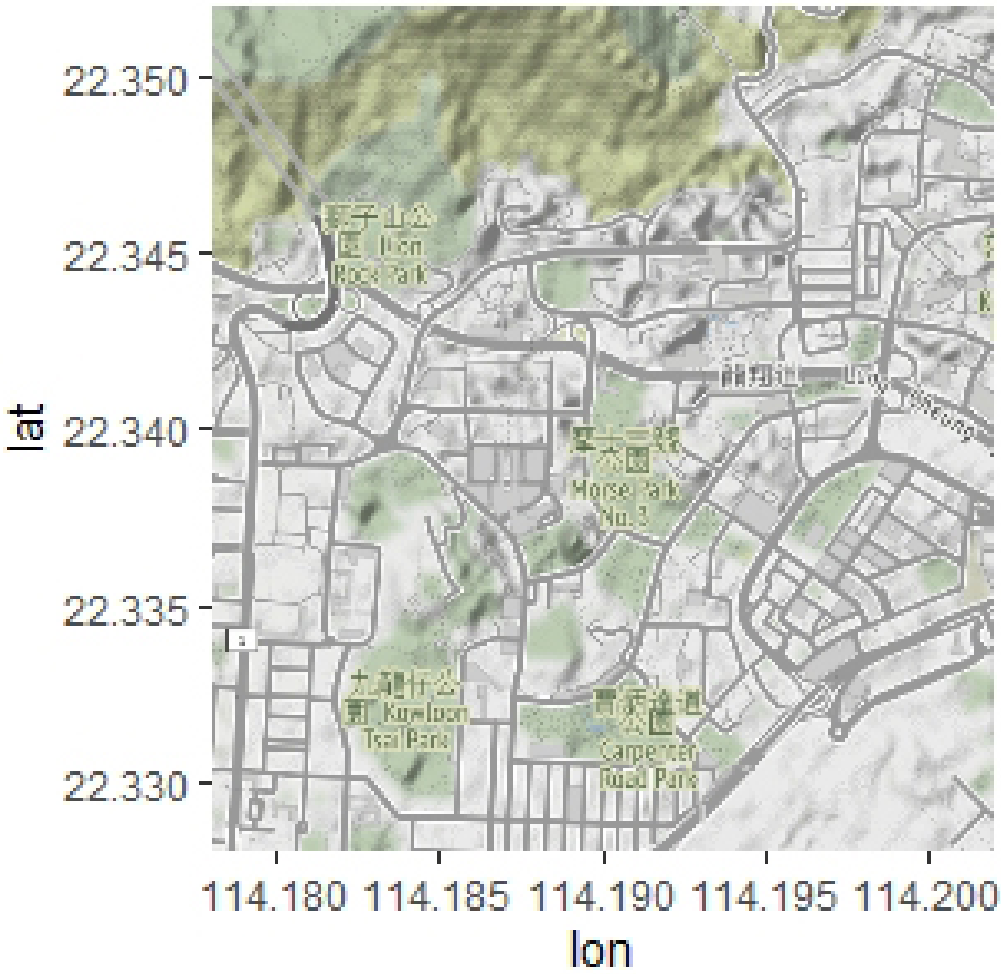
Import a Local Shapefile (Cont'd)

- The confirmed case is a continuous variable and is plotted by **ggplot** as such
- A "true" choropleth map we need to change the continuous vairable into a categoriacal one based on e.g. quantile in two steps:
  - Determine the quantile breaks
  - Add a categorical variable to the object which assigns each continious vaule to a bracket



Adding Basemaps using **ggmap**

- We can download basemaps (in the forms of satellite, roadmap, or terrain etc.) from google, OSM and other using **ggmap**

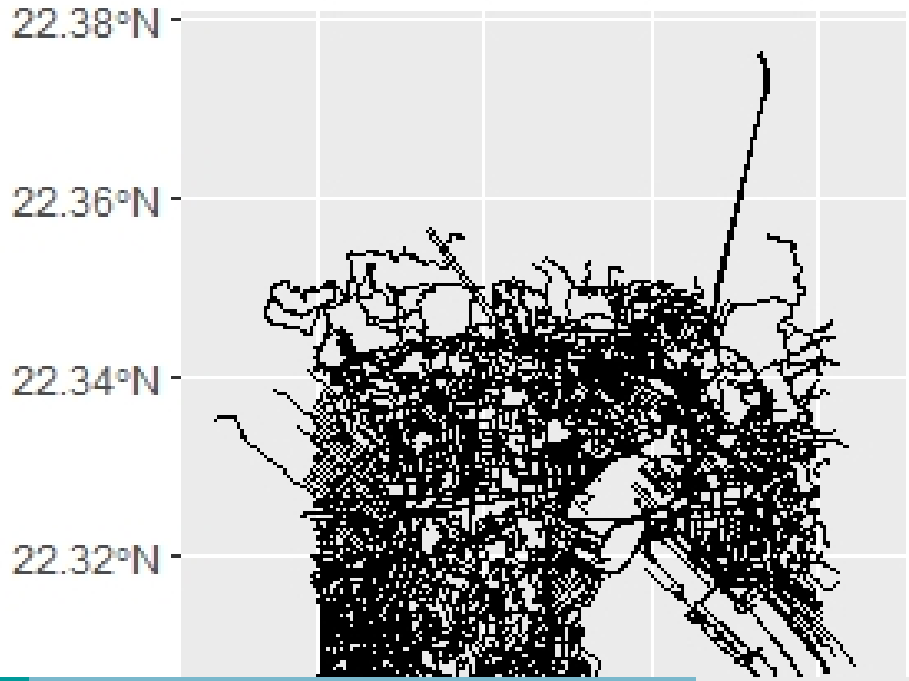


- If we want to use the map service from google, we need to apply for an API
- Then using **register_google()** function to enter the API key

Get some Road Map using **osmdata**

```
279 # get road map from open street project
280 install.packages("osmdata")
281 library(osm
282
283 kowloon <- opq(bbox =  c(114.16, 22.35, 114.22, 22.30)) %>%
284   add_osm_feature(key = 'highway') %>%
285   osmdata_sf() %>%
286   osm_poly2line()
287
288 kowloon_center <- kowloon$osm_lines %>%
289   select(highway)
290
291 kowloon_center
292
293 ggplot(data = kowloon_center) + geom_sf()
```

opq(bbox = NULL, timeout = 25, memsize)

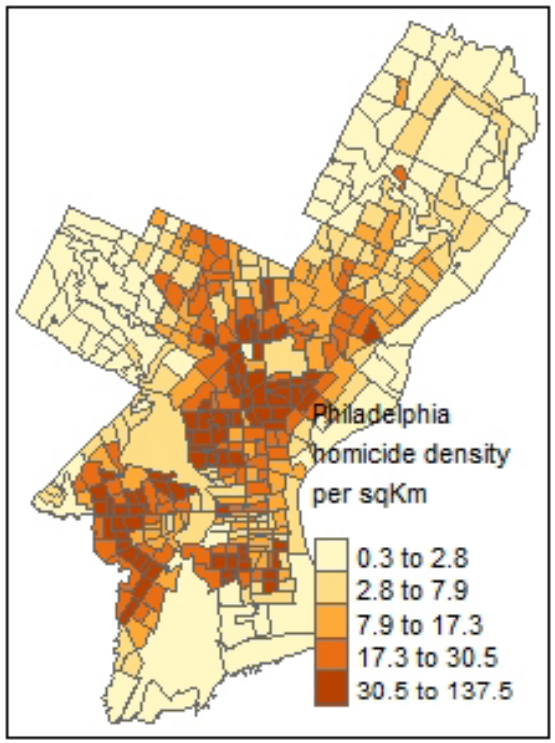


Creating a spatial object from a lat/lon table

```
302 # download and prepare data
303 download.file("http://bit.ly/R-spatial-data", "R-spatial-data.zip")
304 unzip("R-spatial-data.zip", exdir = "data")
305
306 # read in the csv file
307 philly_homicides_df <- read.csv("data/philly_homicides.csv")
308 str(philly_homicides_df) # we have longitude/latitude information inside
309
310 # create an sf object from a data frame using the st_as_sf() function
311 philly_homicides_sf <- st_as_sf(philly_homicides_df, coords = c("POINT_X",
    "POINT_Y"))
312 str(philly_homicides_sf)
313
314 # To make it a complete geographical object we assign the WGS84 projection,
    which has the EPSG code 4326:
315 st_crs(philly_homicides_sf)
316
317 st_crs(philly_homicides_sf) <- 4326 # we can use EPSG as numeric here
318 st_crs(philly_homicides_sf)
319
320 # save the object as shapefile on our hard drive for later use
321 st_write(philly_homicides_sf, "data/PhillyHomicides", driver = "ESRI
    Shapefile")
322 # to force the save:
323 st_write(philly_homicides_sf, "data/PhillyHomicides", driver = "ESRI
    Shapefile", delete_layer = TRUE)
```

Choropleth with **tmap**

- **tmap** is specifically designed to make creation of thematic maps more convenient
- It borrows from the **ggplot** syntax and takes care of a lot of the styling and aesthetics
  - **tm_shape()** where we provide the **sf** object
  - **tm_polygons()** where we set the attribute variable to map, the break style, and titles

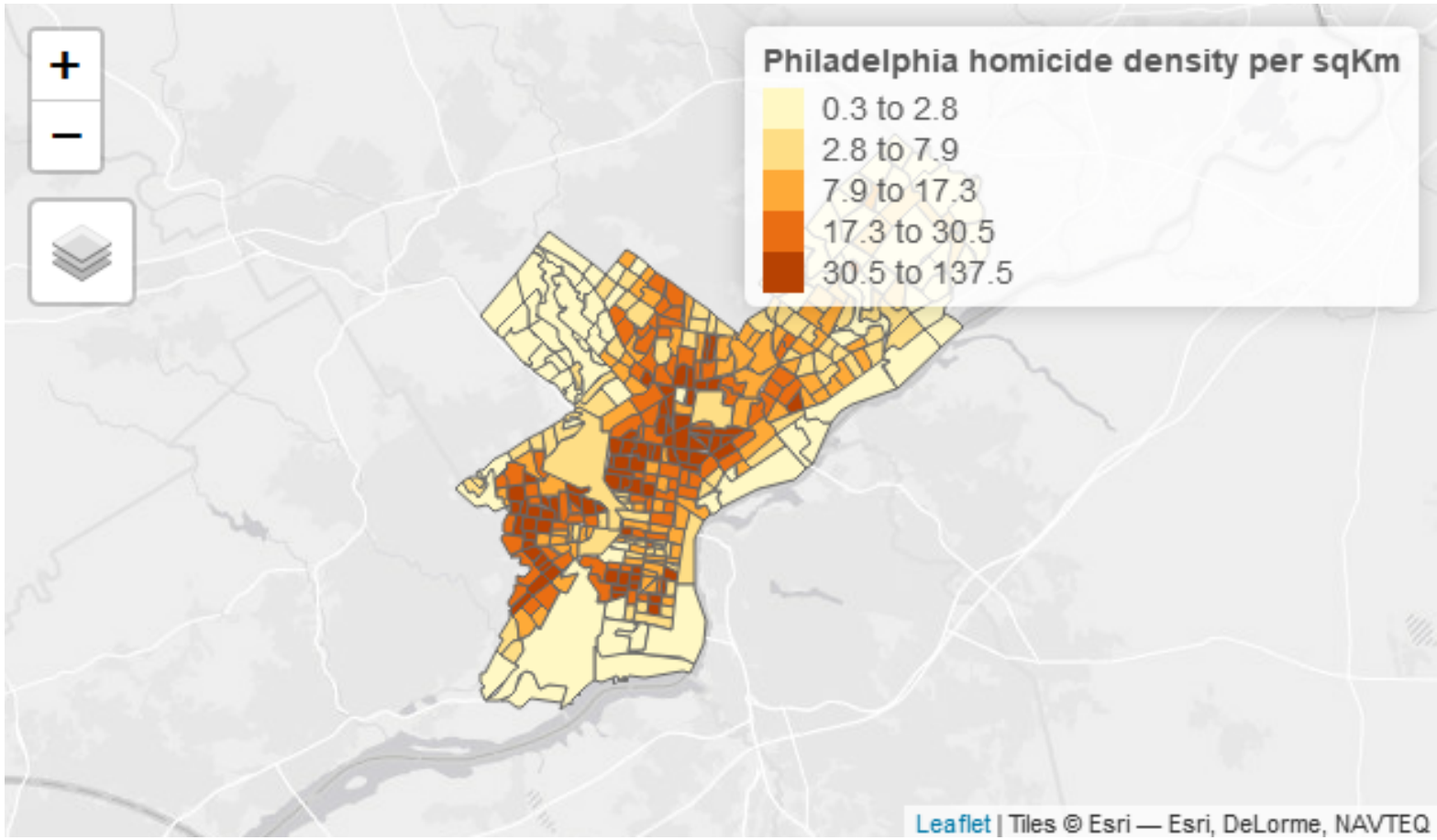



```
334 tm_shape(philly_crimes_sf) +
335   tm_polygons("homic_rate",
336               style="quantile",
337               title="Philadelphia \nhomicide density
```

Choropleth with **tmap** (Cont'd)

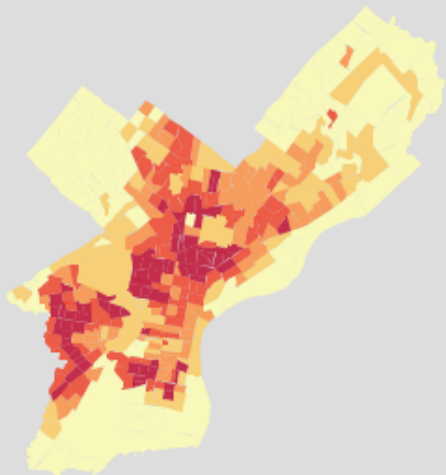
- Change from "plot" to the cool "view" mode for an interactive map, and clal the last plot

```
339 tmap_mode("plot")
340 tmap_last()
```



Web Mapping with **leaflet**

- **addPolygons()** is used to map the homicide density in philly, and
  - remove stroke (polygon borders) using **stroke = FALSE**
  - generate the fill color using leaflet's **colorQuantile()** function
  - set a fillColor for each polygon based on homic_rate and make it look nice by adjusting fillOpacity and smoothFactor (how much to simplify the polyline on each zoom level)
  - add a popup with the homic_rate values



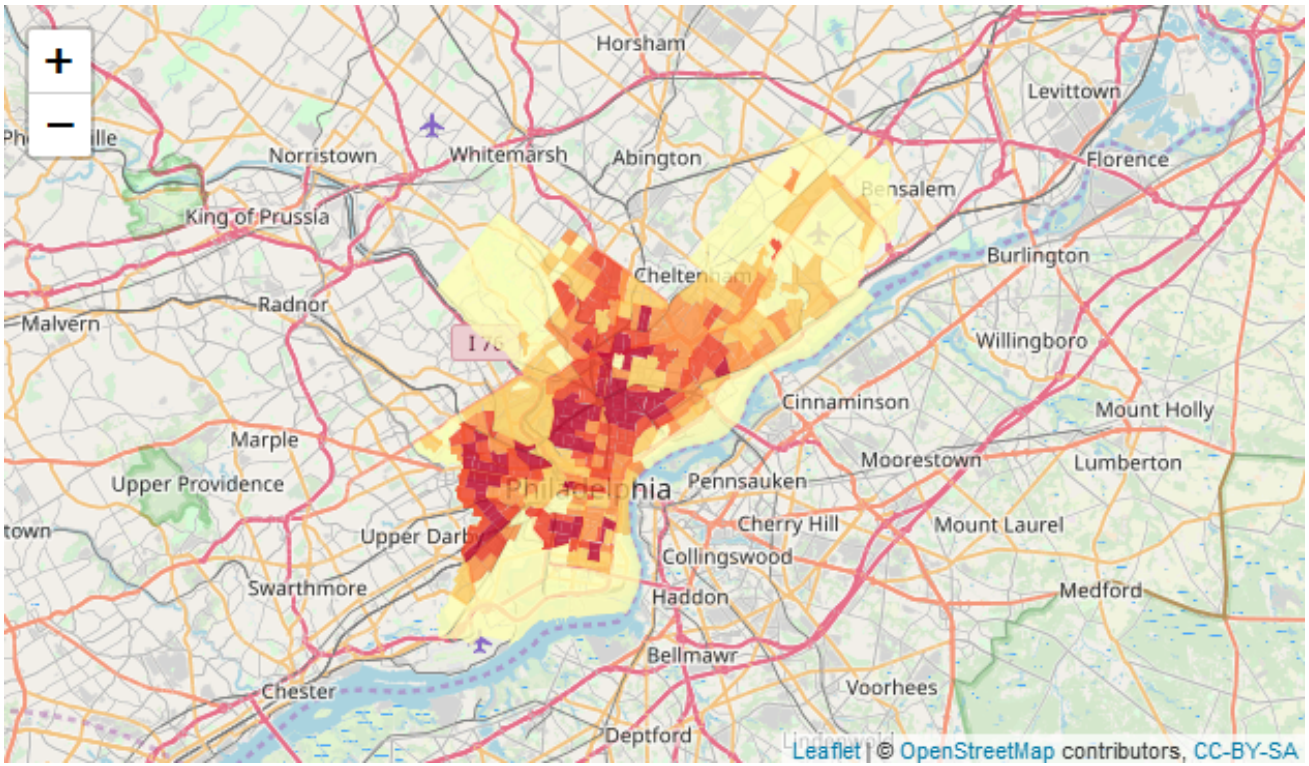
```
358  # set a fillColor for each polygon based on homic_ra
     nice by adjusting fillOpacity and smoothFactor (how
     polyline on each zoom level)
359  pal_fun <- colorQuantile("YlOrRd", NULL, n = 5)
360
361  # add a popup with the homic_rate values. We will cr
     strings, that we then supply to addPolygons()
362  p_popup <- paste0("<strong>Homicide Density: </stron
     _rate)
363
364  leaflet(philly_WGS84) %>%
365    addPolygons(
366      stroke = FALSE, # remove polygon borders
367      fillColor = ~pal_fun(homic_rate), # set fill col
     above and value
368      fillOpacity = 0.8, smoothFactor = 0.5, # make it
369      popup = p_popup)  # add popup
```

Leaflet

Web Mapping with **leaflet** (Cont'd)

- Add a basemap, which defaults to OSM, with **addTiles()**

```
372 # Add a basemap, which defaults to OSM, with addTiles()
373 leaflet(philly_WGS84) %>%
374   addPolygons(
375     stroke = FALSE,
376     fillColor = ~pal_fun(homic_rate),
377     fillOpacity = 0.8, smoothFactor = 0.5,
378     popup = p_popup) %>%
379   addTiles()
```



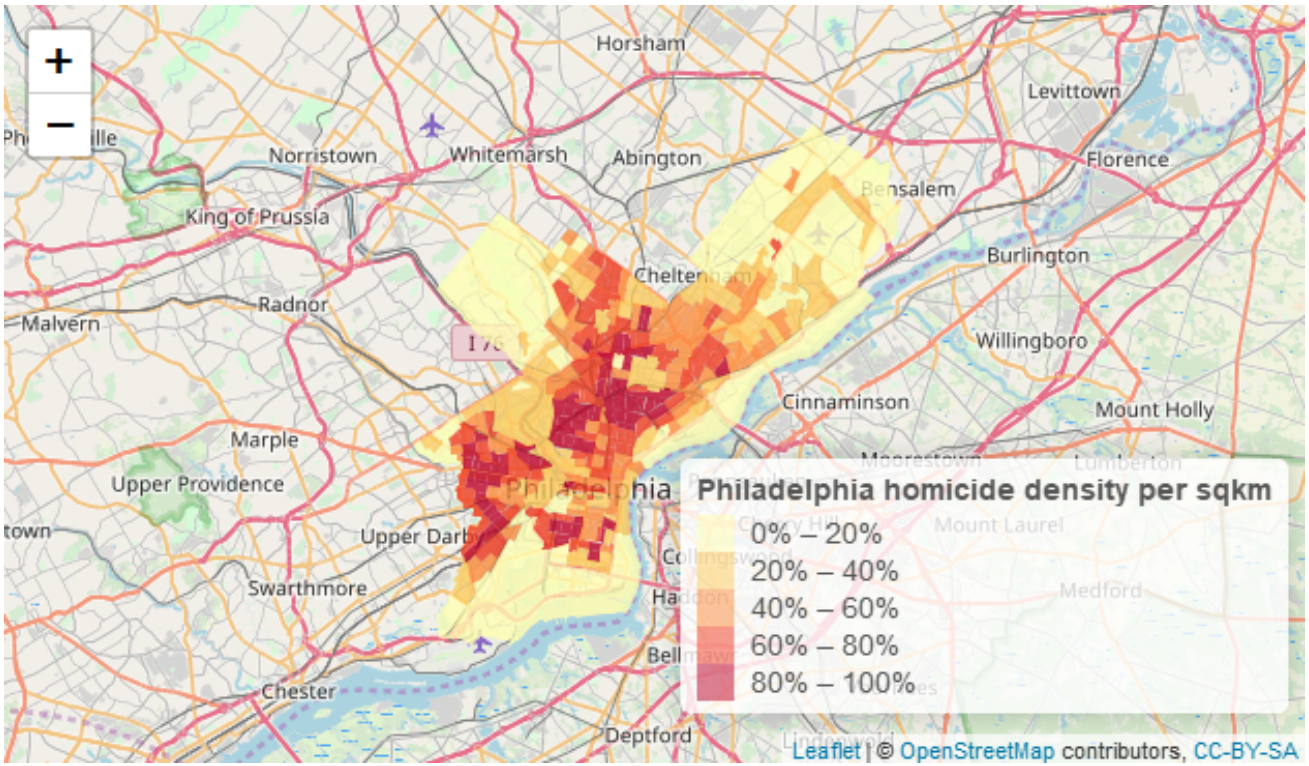
Web Mapping with **leaflet** (Cont'd)

- Lastly, we add a legend. We will provide the **addLegend()** function with:
  - the location of the legend on the map
  - the function that creates the color palette
  - the value we want the palette function to use
  - a title

```
381 # Add a legend using addLegend()
382 leaflet(philly_WGS84) %>%
383   addPolygons(
384     stroke = FALSE,
385     fillColor = ~pal_fun(homic_rate),
386     fillOpacity = 0.8, smoothFactor = 0.5,
387     popup = p_popup) %>%
388   addTiles() %>%
389   addLegend("bottomright",  # location
390             pal=pal_fun,    # palette function
391             values=~homic_rate,  # value to be passed to palette function
392             title = 'Philadelphia homicide density per sqkm') # legend
    title
```

Web Mapping with **leaflet** (Cont'd)

- Lastly, we add a legend. We will provide the **addLegend()** function with:
  - the location of the legend on the map
  - the function that creates the color palette
  - the value we want the palette function to use
  - a title



Web Mapping with **leaflet** (Cont'd)

- To set the labels for our breaks manually we replace the **pal** and **values** with the **colors** and **labels** arguments

```
394 # Set the labels for our breaks manually
395 ## get quantile breaks. Add .00001 offset to catch the lowest value
396 breaks_qt <- classIntervals(c(min(philly_crimes_sf$homic_rate) - .00001,
    philly_crimes_sf$homic_rate), n = 7, style = "quantile")
397 breaks_qt
398
399 library(RColorBrewer)
400
401 leaflet(philly_WGS84) %>%
402   addPolygons(
403     stroke = FALSE,
404     fillColor = ~pal_fun(homic_rate),
405     fillOpacity = 0.8, smoothFactor = 0.5,
406     popup = p_popup) %>%
407   addTiles() %>%
408   addLegend("bottomright",
409             colors = brewer.pal(7, "YlOrRd"),
410             labels = paste0("up to ", format(breaks_qt$brks[-1], digits = 2
    )),
411             title =  'Philadelphia homicide density per sqkm')
```

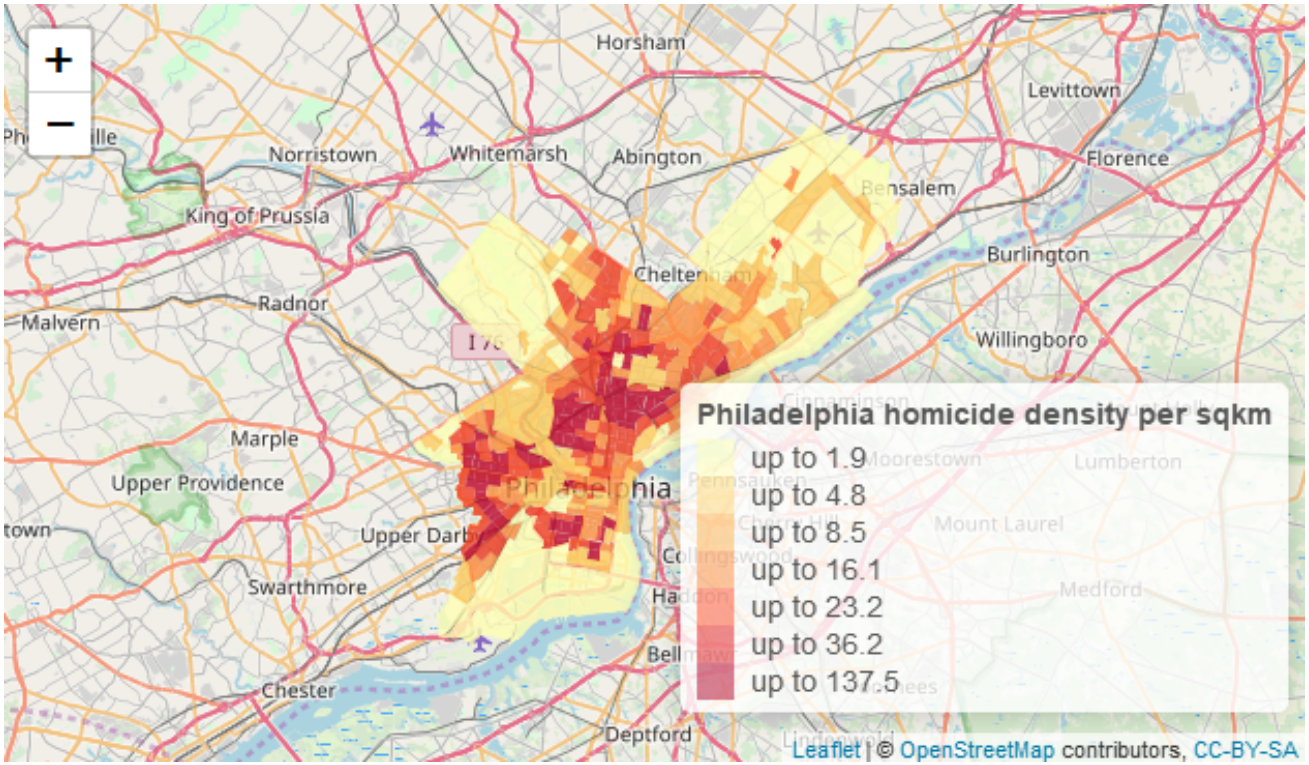
Web Mapping with **leaflet** (Cont'd)

- To set the labels for our breaks manually we replace the **pal** and **values** with the **colors** and **labels** arguments

```
394 # Set the labels for our breaks manually
395 ## get quantile breaks. Add .00001 offset to catch the lowest value
396 breaks_qt <- classIntervals(c(min(philly_crimes_sf$homic_rate) - .00001,
    philly_crimes_sf$homic_rate), n = 7, style = "quantile")
397 breaks_qt
398
399 library(RColorBrewer)
400
401 leaflet(philly_WGS84) %>%
402   addPolygons(
403     stroke = FALSE,
404     fillColor = ~pal_fun(homic_rate),
405     fillOpacity = 0.8, smoothFactor = 0.5,
406     popup = p_popup) %>%
407   addTiles() %>%
408   addLegend("bottomright",
409             colors = brewer.pal(7, "YlOrRd"),
410             labels = paste0("up to ", format(breaks_qt$brks[-1], digits = 2
    )),
411             title =  'Philadelphia homicide density per sqkm')
```

Web Mapping with **leaflet** (Cont'd)

- To set the labels for our breaks manually we replace the **pal** and **values** with the **colors** and **labels** arguments



Web Mapping with **leaflet** (Cont'd)

- Finally, Add a control to switch to another basemap and turn the philly polygon off and on

```
414 # Add a control to switch to another basemap from "Carto", and option to
    turn the polygon off and on
415 leaflet(philly_WGS84) %>%
416   addPolygons(
417     stroke = FALSE,
418     fillColor = ~pal_fun(homic_rate),
419     fillOpacity = 0.8, smoothFactor = 0.5,
420     popup = p_popup,
421     group = "philly") %>%
422   addTiles(group = "OSM") %>%
423   addProviderTiles("CartoDB.DarkMatter", group = "Carto") %>%
424   addLegend("bottomright",
425             colors = brewer.pal(7, "YlOrRd"),
426             labels = paste0("up to ", format(breaks_qt$brks[-1], digits = 2
    )),
427             title = 'Philadelphia homicide density per sqkm') %>%
428   addLayersControl(baseGroups = c("OSM", "Carto"),
429                    overlayGroups = c("philly"))
```

Web Mapping with **leaflet** (Cont'd)

- Finally, Add a control to switch to another basemap and turn the philly polygon off and on