COLUMBIA UNIVERSITY
School of Professional Studies

Applied Analytics: Frameworks and Methods 2

Spatial Analysis

Outline

- What is spatial analysis and the motivation behind it
- Explain spatial data
- Gather and represent data on a map
- Analyze spatial data
- Illustrations using R

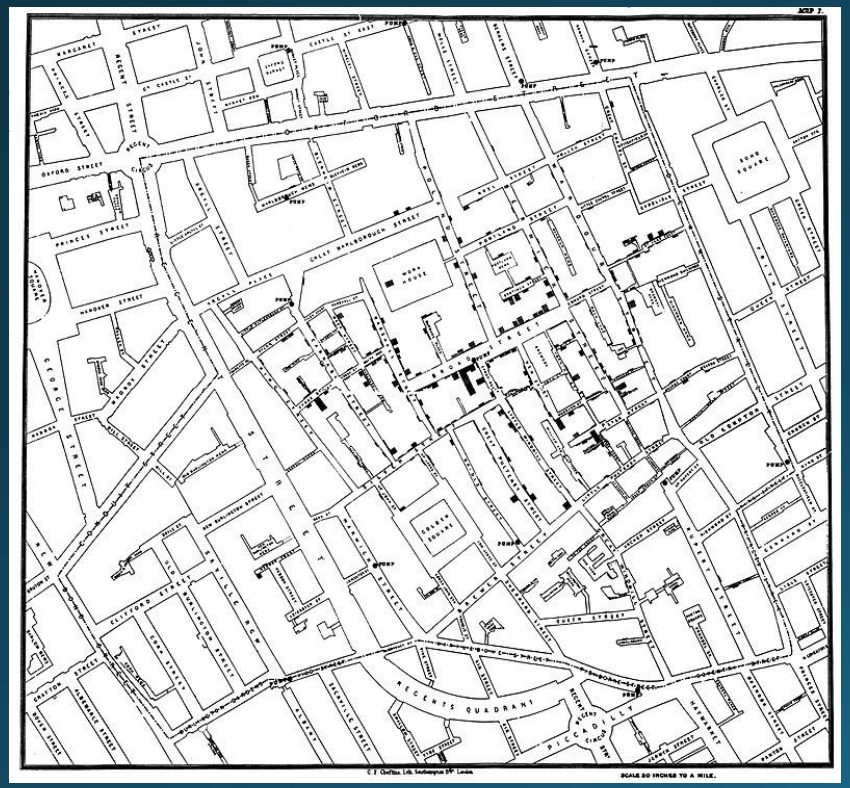
Spatial Analysis & Motivation Behind

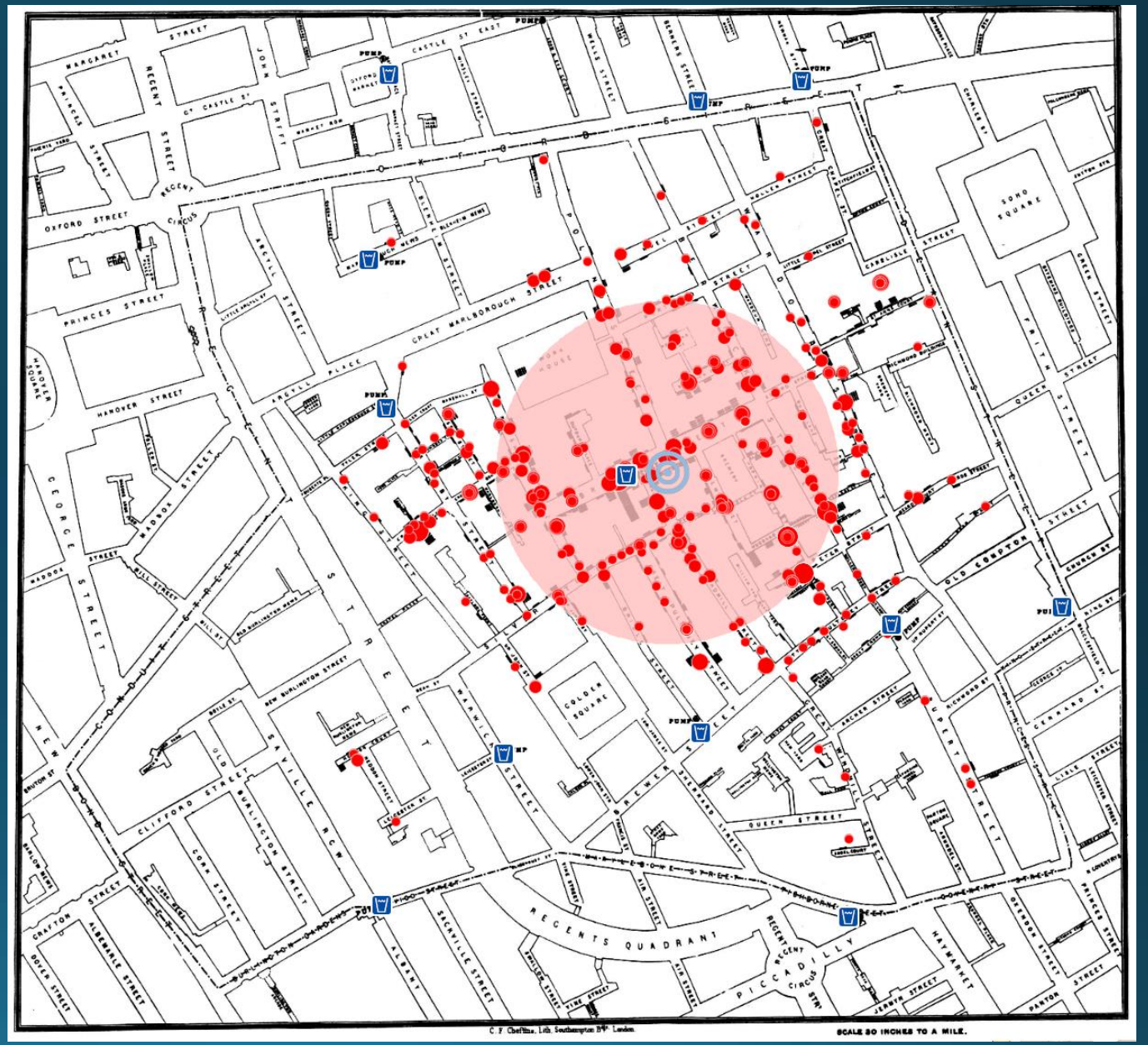
Spatial Analysis

- Using location of observation units to visualize, identify patterns, and conduct analysis
- Involves contributions from cartography, biology, ethological studies, ecological studies, epidemiology, statistics, econometrics, computer science and math.

Roots of Spatial Analysis

- John Snow's Cholera Map
- Plotting Cholera cases on a map helped point to a water pump as the source of the disease





Source: Wikipedia, Arcgis.com

Motivation for Examining Spatial Data

- The amount of location data is growing at a rapid rate aided by data generated by

  - Mobile devices: Not only smartphone service providers, but also smartphone apps and websites are gathering location information.

  - IoT devices such as smartwatches, fitbit, automatic vacuum cleaners are gathering location data

  - Machines are gathering location data. These include ride-sharing services (e.g., Uber, Lyft), and self-driving cars (e.g., Tesla).

Motivation for Examining Spatial Data

- Possible questions addressed

  - Does the spatial patterning of disease incidences give rise to the conclusion that they are clustered, and if so, are the clusters found related to factors such as age, relative poverty, or pollution sources?

  - Given a number of observed soil samples, which part of a study area is polluted?

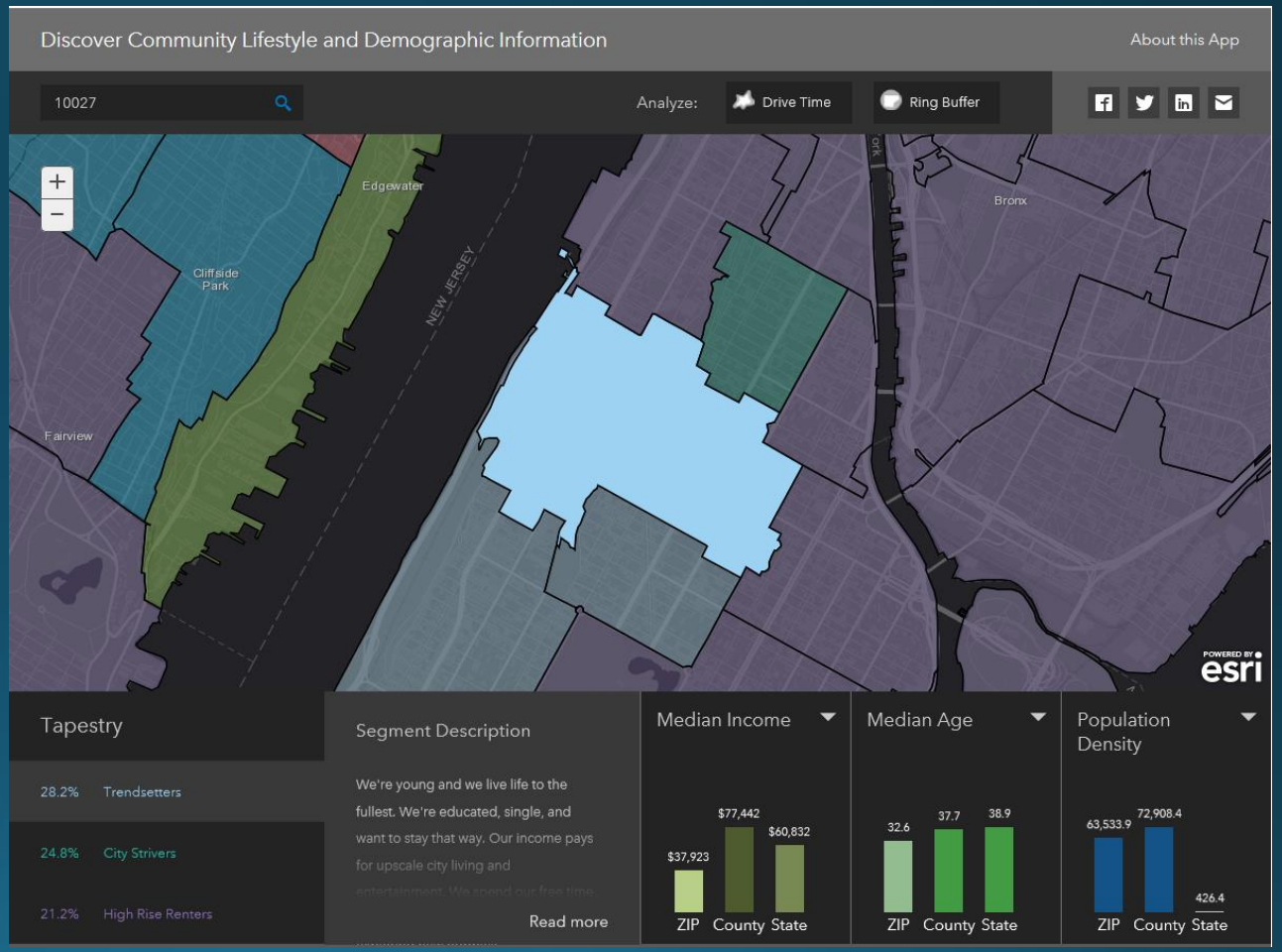
  - Given scattered air quality measurements, how many people are exposed to high levels of black smoke or particulate matter (e.g. PM10), and where do they live?

  - Do governments tend to compare their policies with those of their neighbors, or do they behave independently?

Motivation for Examining Spatial Data

- For marketers, where a person lives reflects the lifestyle segment they belong to.

- Keys to success in brick and mortar retail is Location, Location and Location



Source: ESRI

Motivation for Examining Spatial Data

- Pattern of points or lines may yield deeper insights.
- Visualization of 1.1 Billion Taxi Trips

Source:toddwschneider.com

Spatial Data

Spatial Data

- Spatial Data is data about location

- Geographic Location is determined by a coordinate system consisting of longitude and latitude.
  - Longitude: Location from east to west
  - Latitude: Location from north to south

- Here is location data on six AirBnb rentals
  - Longitude marked by W are negative
  - Latitudes marked N are positive

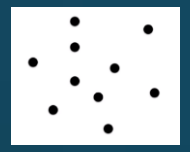
	longitude	latitude
1	-73.89593	40.75444
2	-73.94234	40.78510
3	-73.95749	40.70989
4	-73.93855	40.85039
5	-73.96821	40.76661
6	-73.95003	40.67575

Projection

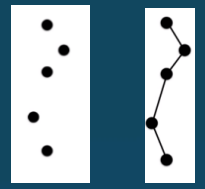
- A projection is the means by which to display the coordinate system and data on a flat surface, such as paper or screen.
- Mathematical calculations are used to convert the coordinate system used on the curved surface of earth to one for a flat surface.
- Since there is no perfect way to transpose a curved surface to a flat surface without some distortion, many different map projections exist that provide different properties. Some preserve shape, while some preserve distance.
  - ellipsoid: approximation of the shape of the earth
  - datum: position of ellipsoid relative to the earth.
  - Common projections:
    - Global datasets (datum = WGS84, ellps=WGS84)
    - North American datasets (datum=NAD83, ellps=GRS80)

Types of spatial data

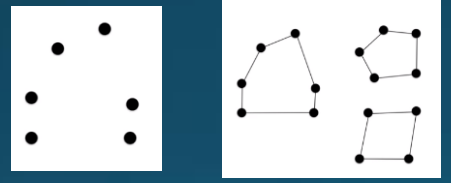
- Point data
  - Data associated with a point.
  - E.g., house or apartment



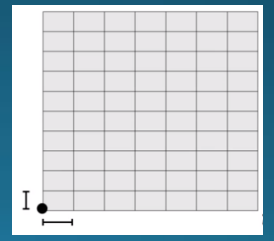
- Line data
  - Data is associated with a collection of points on a line
  - E.g., road, river



- Polygon data
  - Data associated with an area
  - E.g., County, City, State, Country



- Raster (or grid) data
  - A grid with data at every location or cell. If each cell is colored, the grid will look like a heat map



Source: https://www.omnisci.com/learn/geospatial

Raster Data

- A raster is a spatial (geographic) data structure that divides a region into rectangles called 'cells' (or'pixels') that can store one or more values for each of these cells
- This data structure is also referred to as a 'grid' and is often contrasted with 'vector' data that is used to represent points, lines, and polygons (Hijmans 2018)
- A RasterLayer object represents single-layer (variable) raster data.
- RasterStack and RasterBrick contain many RasterLayer

Spatial Data with R

- Data frames are not well-suited for representing spatial data for the following reasons.
  - For polygons, non-spatial data is repeated. For e.g., a state is defined by many coordinates but it only has one number for population.
  - Data frames work only when there is a uniform coordinate reference system. Differences in coordinate reference system creates challenges in merging different data sources.

- Packages
  - With its rich ecosystem of spatial analysis packages, R is one of the best tools to examine spatial data.
  - Various package including sp, sf, raster
  - sp is widely used

sp

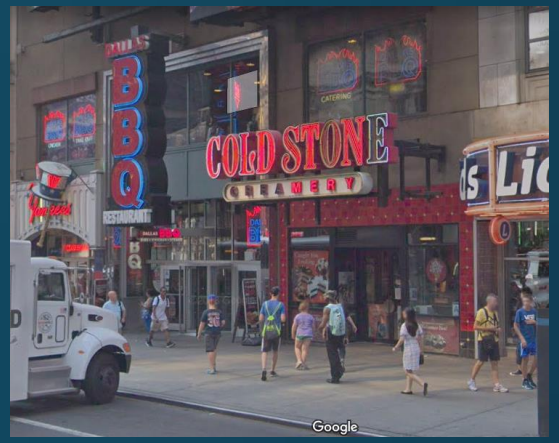
- S4 object
- Object has slots. Most important slots
  - @data – the data
  - @bbox – bounding box
  - @coordinates (since SpatialPoints don't have a hierarchy), @Lines, @polygons – coordinate information
  - @proj4string – projection type
- Different types of Spatial objects based on what they describe
  - Points: SpatialPoints
  - Lines: SpatialLines
  - Polygons: SpatialPolygons
- Spatial Object + Data
  - SpatialPointsDataFrame
  - SpatialLinesDataFrame
  - SpatialPolygonsDataFrame

Visualize Spatial Data

Visualizing Spatial Data

- Address: 241 W 42nd St, New York, NY 10036
- Longitude: -73.988640 (or 73.98 W)
- Latitude: 40.757020 (or 40.75 N)

- Longitude: x-coordinate
- Latitude: y-coordinate



Approach to Visualizing Spatial Data

- Key to representing data on a map is to think about adding layers of data onto a static map.

  For instance, one could add data on houses to a map of an area

  - Map
  - Additional geographic detail
  - Data Layer
  - More data layers …

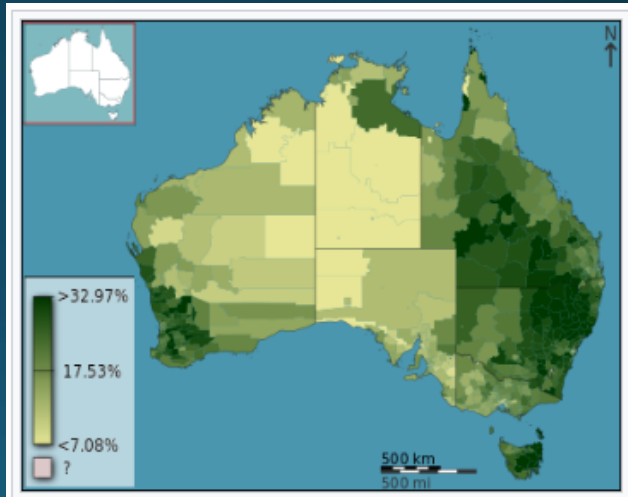
Visualizing Spatial Data

- ggplot2
  - Useful for plotting data frame data
  - However, data lacks a geographical context

- ggmap
  - Complements ggplot2 by providing geographic detail
  - ggmap downloads maps from web services (e.g., google maps and open street maps) and adds them as a layer to plots

Plotting Spatial Data: tmap

- library(tmap) offers a flexible, layer-based, and easy to use approach to create thematic maps, such as choropleths and bubble maps.
- Designed to work with spatial objects
- Like ggplot2, plot is built with layers



A choropleth map that visualizes the fraction of Australians that identified as Anglican at the 2011 census

Source: Wikipedia

Analyze Spatial Data

Analyze Spatial Data

- Assumes nearby georeferenced units are associated in some way
- Point pattern analysis
  - Identification of spatial clusters or groups of observations
  - Explanation for the clusters
  - Changes in spatial pattern over time
- Area patterns
- Interpolation and geostatistics

Illustrations - using R

Spatial Analysis

Read Data

You must read the data before trying to run code on your own machine. To read data use the following code after setting your working directory. To set your working directory, modify the following to set the file path for the folder where the data file resides. setwd('c:/thatawesomeclass/)

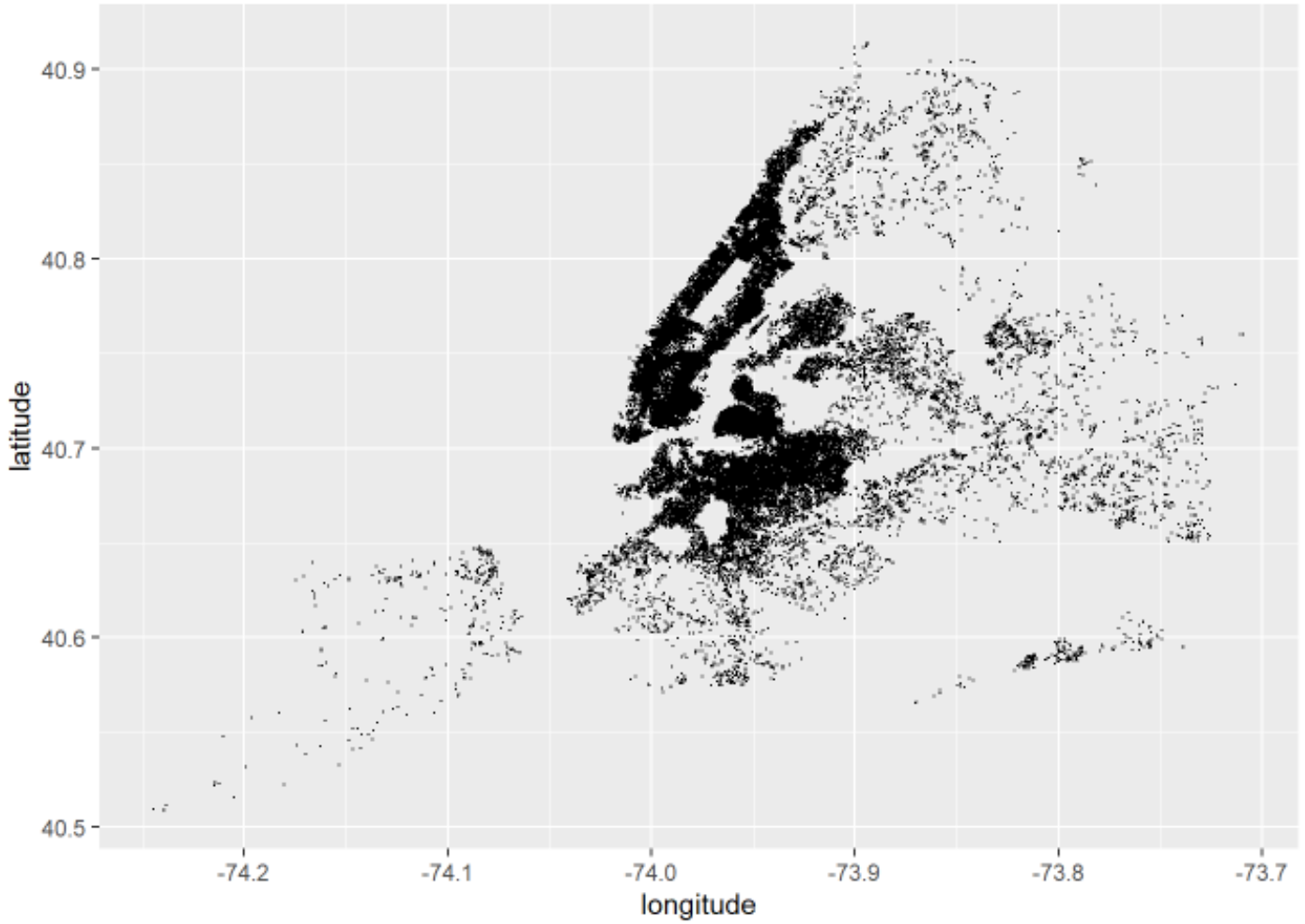
Here is AirBnb data on rentals in New York city for a couple of months. Since the data contains locations of the rentals, this is spatial data. The features of each rental are related to a single location, therefore this is called point data.

```
data = read.csv('analysisData_with_lonlat.csv')
```

Visualize airbnb data

What does the plot look like?

```
library(ggplot2)
ggplot(data) +
  geom_point(aes(x=longitude,y=latitude),size=0.1)
```



Mapping Data using Google Maps

Google API Data

Google has recently changed its API requirements, and ggmap users are now required to provide an API key and enable billing. ggmap itself is outdated on CRAN. Until the new version is available on CRAN, you can install an updated version by running the following code

```
# if(!requireNamespace("devtools")) install.packages("devtools")
# devtools::install_github("dkahle/ggmap", ref = "tidyup")
```

When you load ggmap, you can set your API key with register_google() (see ?register_google for details), but don't forget to enable the Maps Static API in the Google Cloud interface and enable billing!

For you to use the google maps code below you will have to get your own api from Google. Once you have obtained an api key, you can assign it to the object api_key.

```
library(ggmap)
register_google(key = api_key)
```

There are two steps to adding a map to a ggplot2 plot with ggmap:

- Download a map using get_map()
- Display the map using ggmap()

Geocoding with ggmap – Google API

COLUMBIA UNIVERSITY
School of Professional Studies

https://cloud.google.com/maps-platform/

https://cran.r-project.org/web/packages/ggmap/readme/README.html

Map of New York

Get map of New York City by setting location as center of the city.

```
library(ggmap)
newyork_map = get_map(location=c(-73.93,40.77), zoom=10, scale=4)
ggmap(newyork_map)
```


```
maptype = c("terrain", "terrain-background",
            "satellite", "roadmap", "hybrid",
            "toner", "watercolor", "terrain-labels",
            "terrain-lines", "toner-2010", "toner-2011",
            "toner-background", "toner-hybrid", "toner-labels",
            "toner-lines", "toner-lite"),
source = c("google", "osm", "stamen"),
```

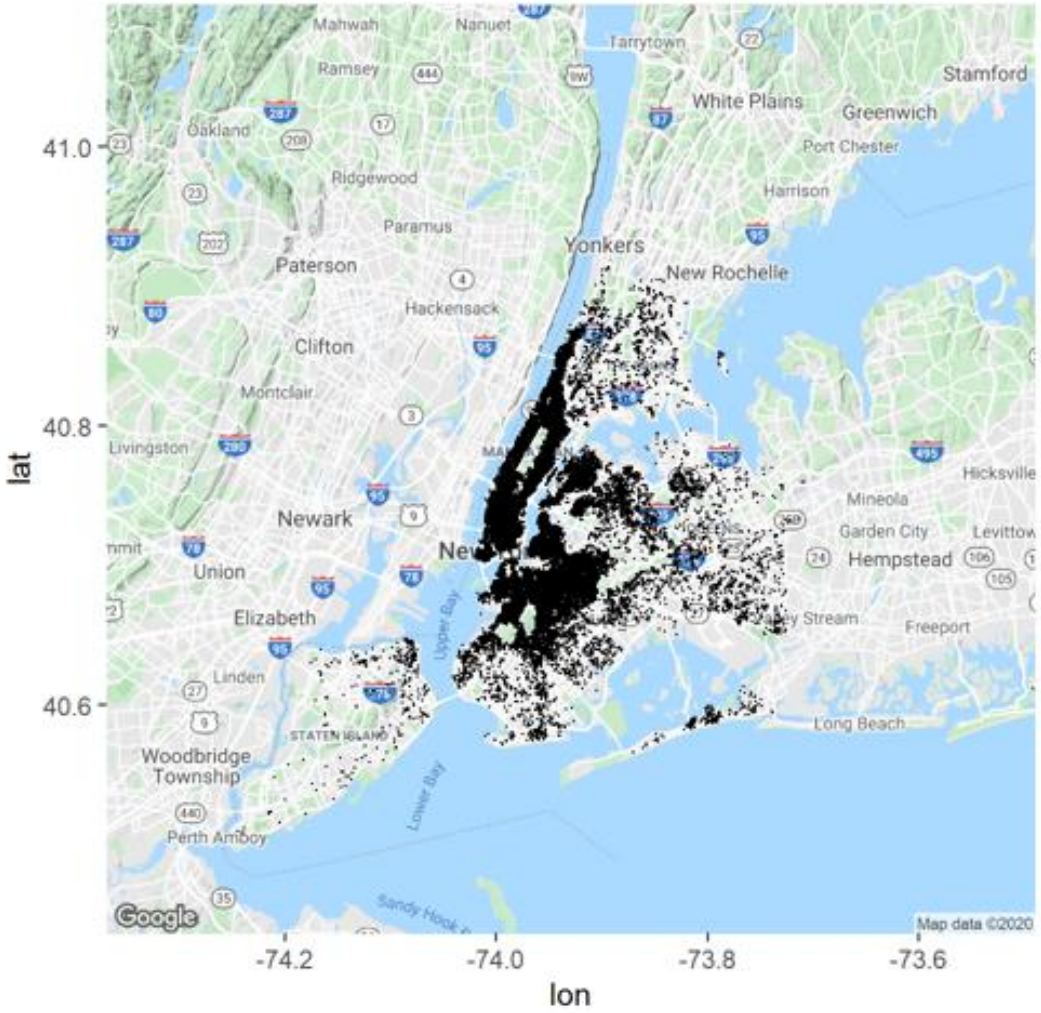
- get_map()
  - provide a number of arguments that control what kind of map to get.
  - default 'maptype' is terrain map
  - default 'source' is google
- Location
  - google search of "coordinates nyc" gives latitude and longitude of nyc.
- Zoom
  - an integer between 3 and 21
  - controls how far the mapped is zoomed in.
- Scale
  - controls the resolution of the downloaded maps
  - the default is 2

- Map of NYC + Rental Locations

COLUMBIA UNIVERSITY
School of Professional Studies

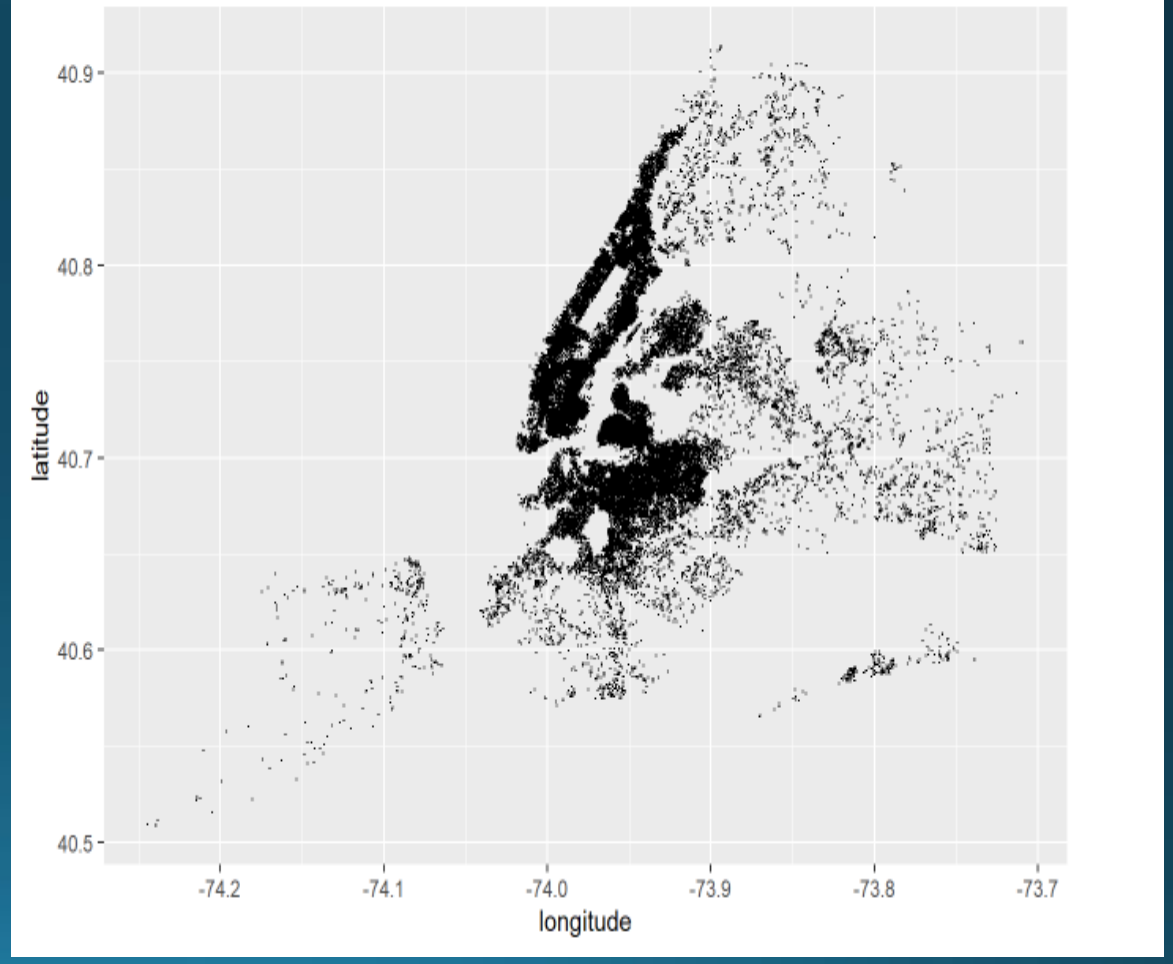
Add a data layer: Map of New York City + AirBnb Rentals

```
ggmap(newyork_map)+
  geom_point(data=data, aes(x=longitude,y=latitude), size=0.1)
```



- Recall from earlier: Rental Locations

```
ggplot(data) +
  geom_point(aes(x=longitude,y=latitude),size=0.1)
```



Address overplotting by setting alpha=0.2, and change the color of rental location to seagreen

```
ggmap(newyork_map)+
  geom_point(data=data, aes(x=longitude,y=latitude), size=0.1, alpha=0.2, color='seagreen')
```

- Map of NYC + Rental Locations

- Displaying the map is then as simple as calling ggmap() with your downloaded map
- Adding a map to your plot of the location explains some of the structure in the data
- This structure is really just a consequence of where the rentals are located.
- The value of displaying data spatially really comes when you add other variables to the display through the properties of your geometric objects, like color scale or size.

- Map of NYC + Rental Locations + Price

The most versatile way to add a custom continuous scale to ggplot2 plots is with:

- scale_color_gradientn(), or
- scale_fill_gradientn()

How do you know which to use? Match the function to the aesthetic you have mapped.

For example, here, we map price with color, and we use scale_color_gradientn()

Arguments

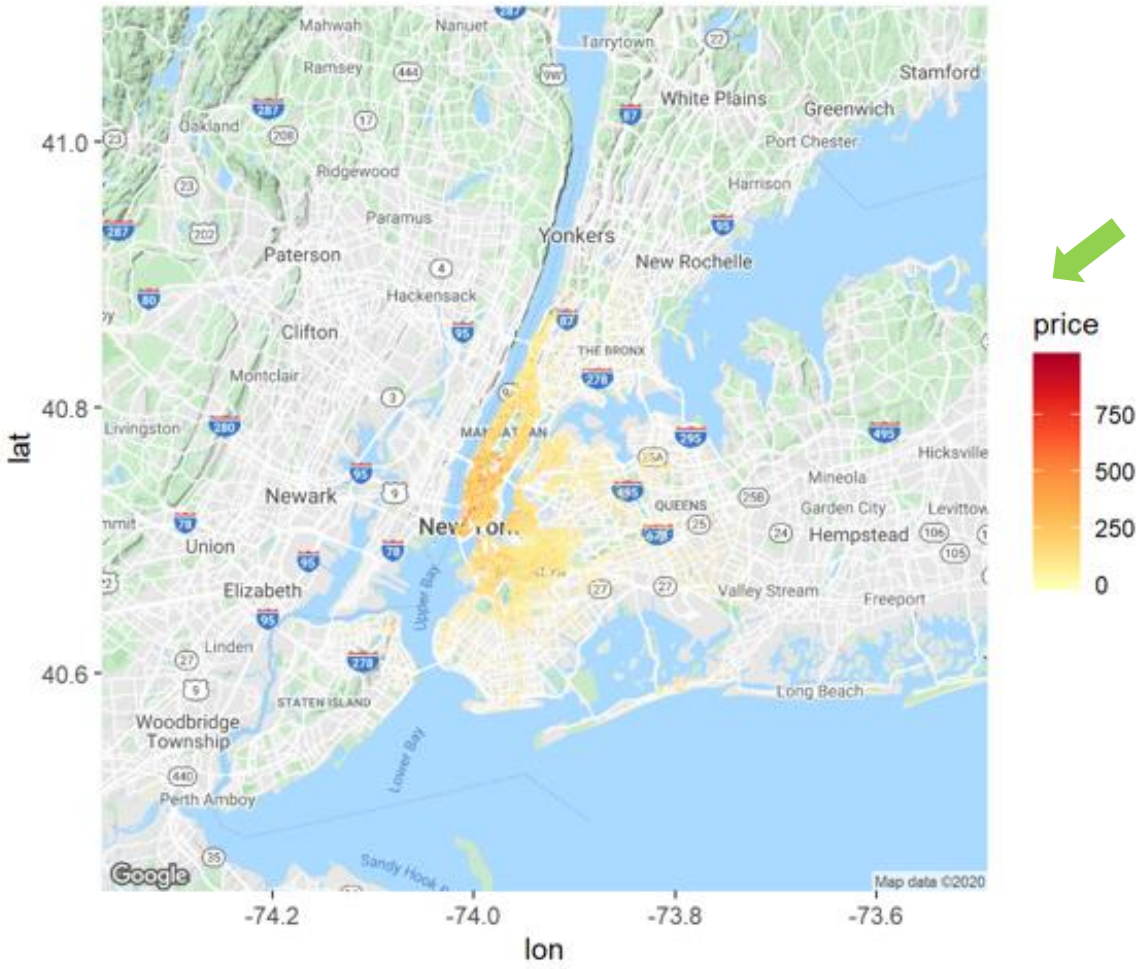
n	Number of different colors in the palette, minimum 3, maximum depending on palette
name	A palette name from the lists below

The sequential palettes names are
Blues BuGn BuPu GnBu Greens Greys Oranges OrRd PuBu PuBuGn PuRd Purples RdPu Reds YlGn YlGnBu YlOrBr YlOrRd

Map of New York city + AirBnb rentals + price

Represent price using Yellow-orange-red scale

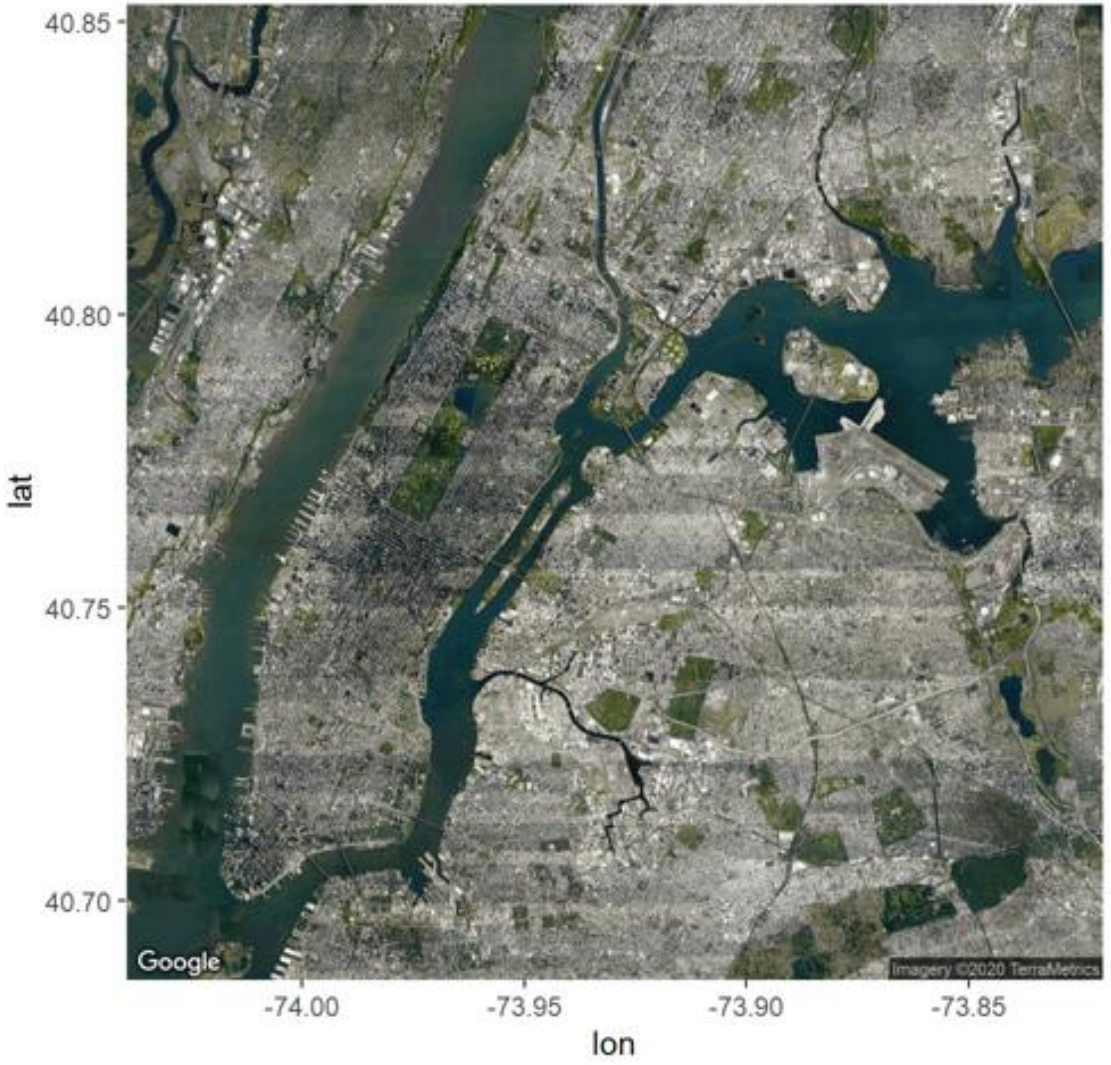
```
library(RColorBrewer)
ggmap(newyork_map)+
  geom_point(data=data, aes(x=longitude,y=latitude,color=price), size=0.1, alpha=0.2)+
  scale_color_gradientn(colors=brewer.pal(n=7, name='YlOrRd'))
```



- Satellite Map of NYC + Rental Locations

Satellite Map of Manhattan

```
manhattan = get_map(location=c(-73.93,40.77), zoom=12, scale=4, maptype='satellite')
ggmap(manhattan)
```



- Satellite Map of NYC + Rental Locations

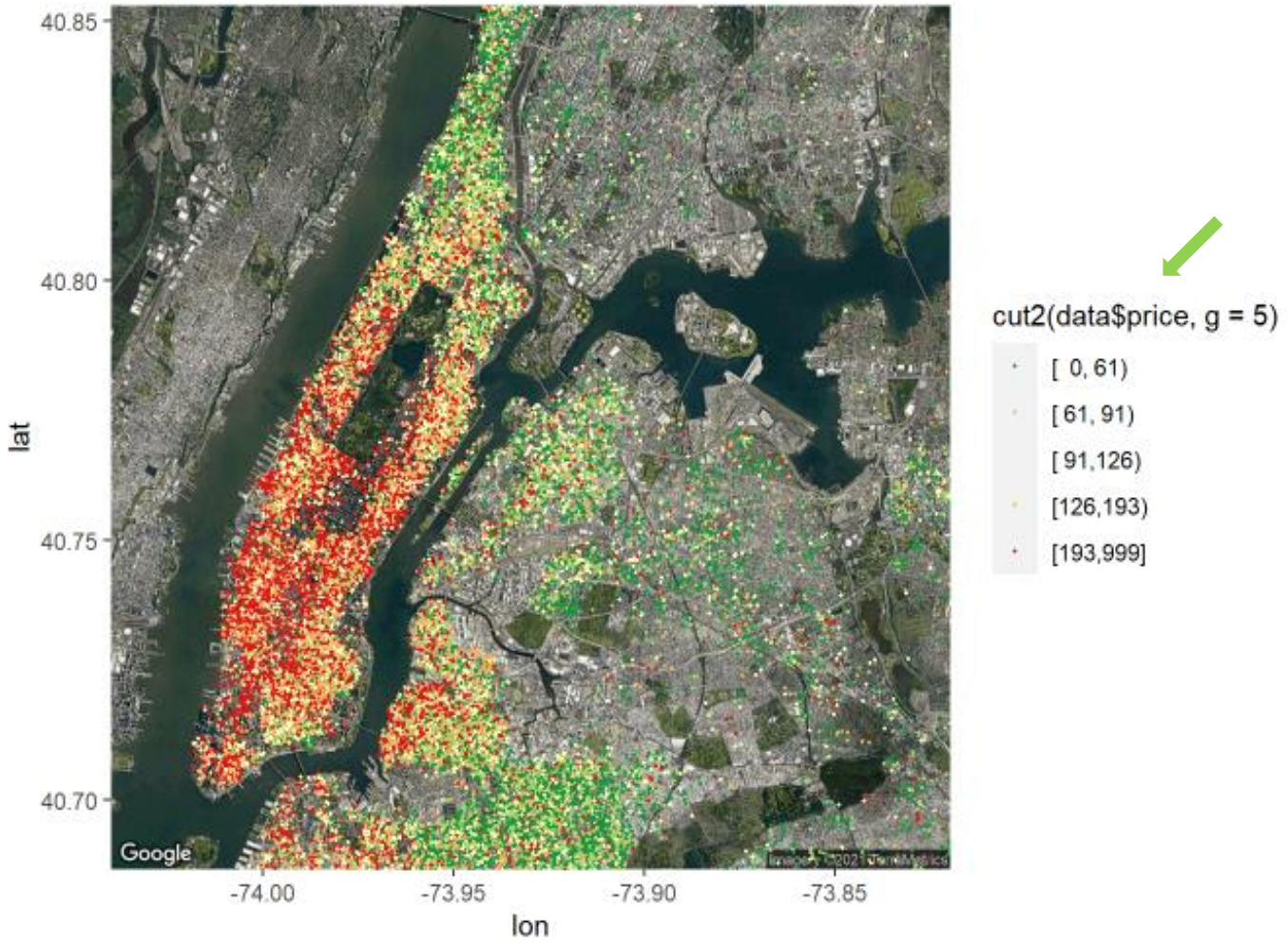
  + Price

direction	Sets the order of colours in the scale. If 1, the default, colours are as output by RColorBrewer::brewer.pal(). If -1, the order of colours is reversed.

Manhattan (Satellite) + Rentals + Price

Price is being converted into five percentile bins to make it easier to spot location differences in price.

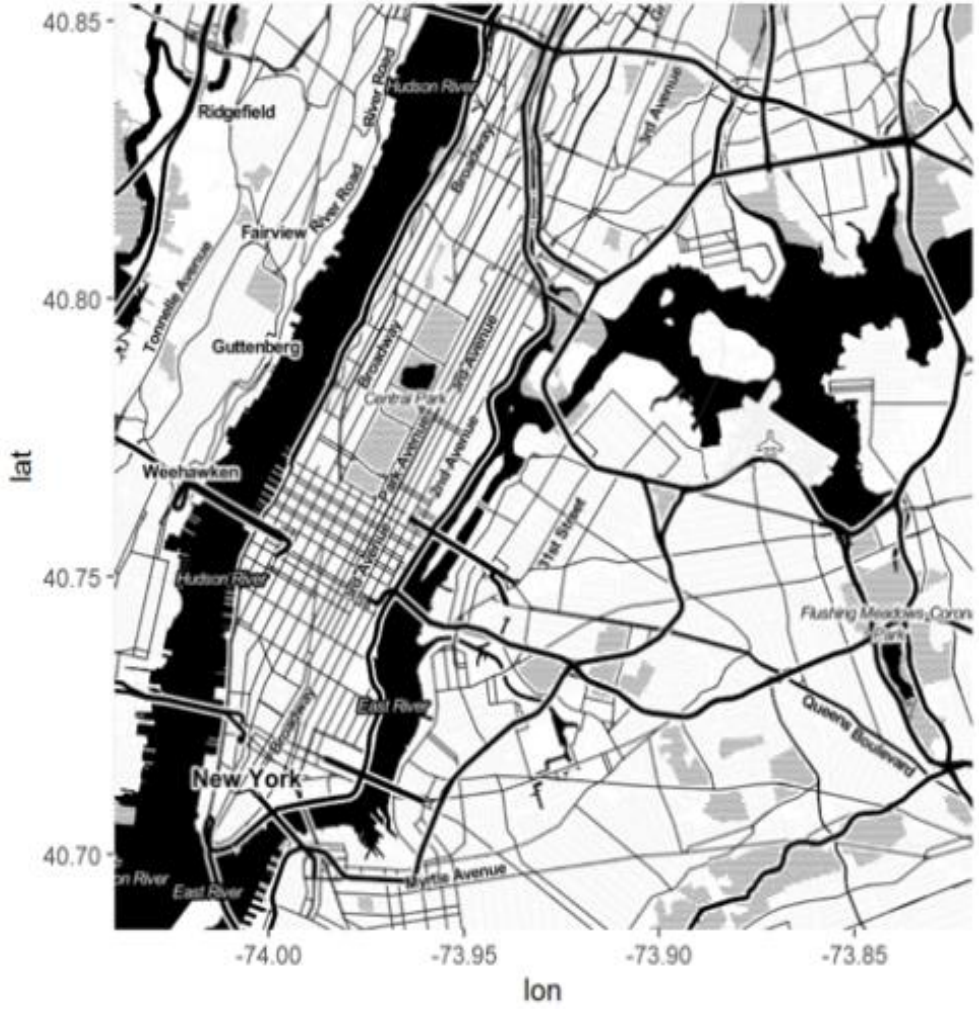
```
library(Hmisc)
ggmap(manhattan)+
  geom_point(data=data, aes(x=longitude,y=latitude,color=cut2(data$price,g=5)), size=0.3)+
  scale_color_brewer(type ='seq', palette='RdYlGn', direction=-1)
```



Toner Map of Manhattan

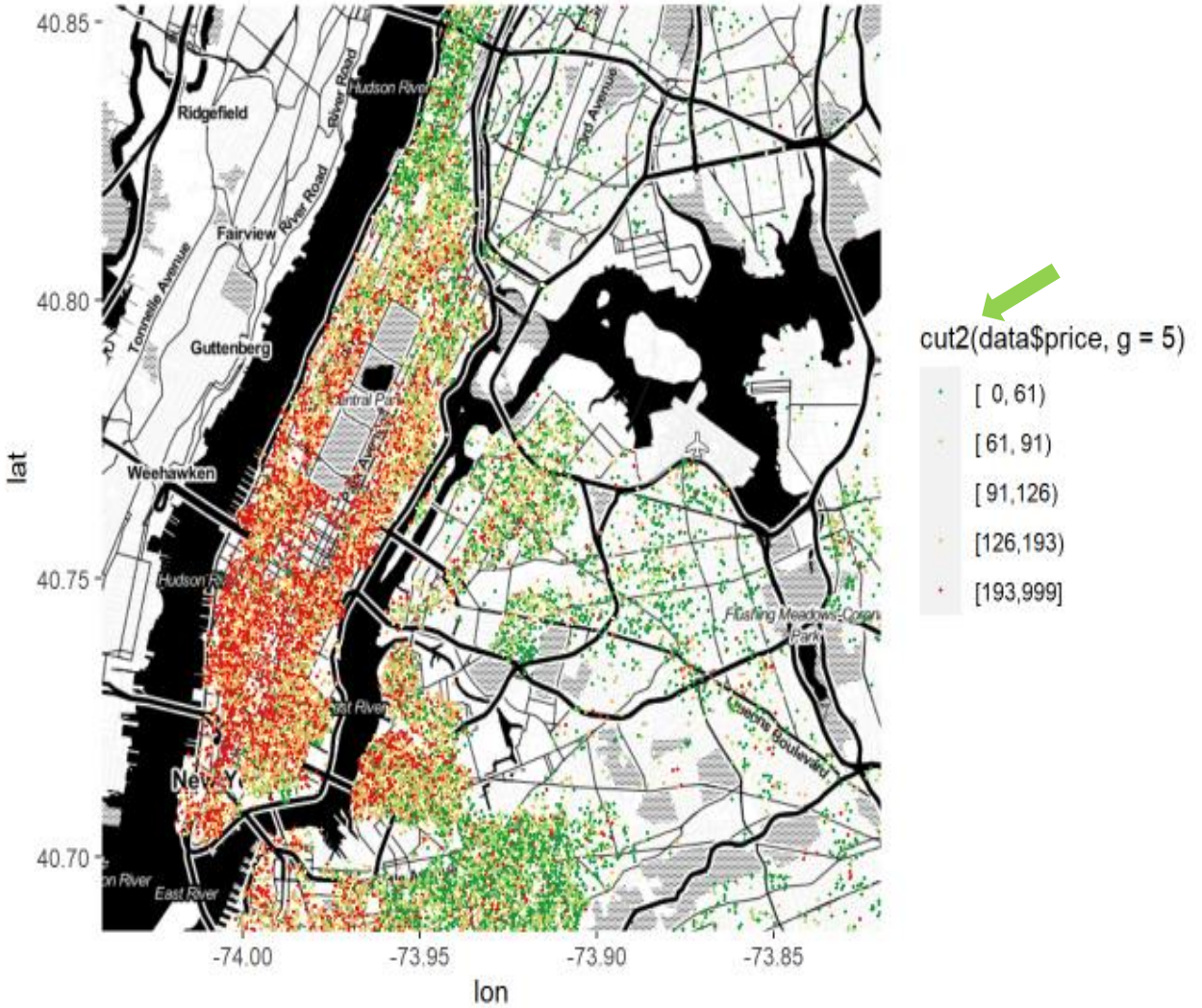
Sometimes, all we want is a simple old fashioned map.

```
manhattan = get_map(location=c(-73.93,40.77), zoom=12, scale=4, maptype='toner')
ggmap(manhattan)
```



Manhattan (Toner) + Rentals + Price

```
ggmap(manhattan)+
  geom_point(data=data, aes(x=longitude,y=latitude,color=cut2(data$price,g=5)), size=0.3)+
  scale_color_brewer(type='seq', palette='RdYlGn', direction =-1)
```

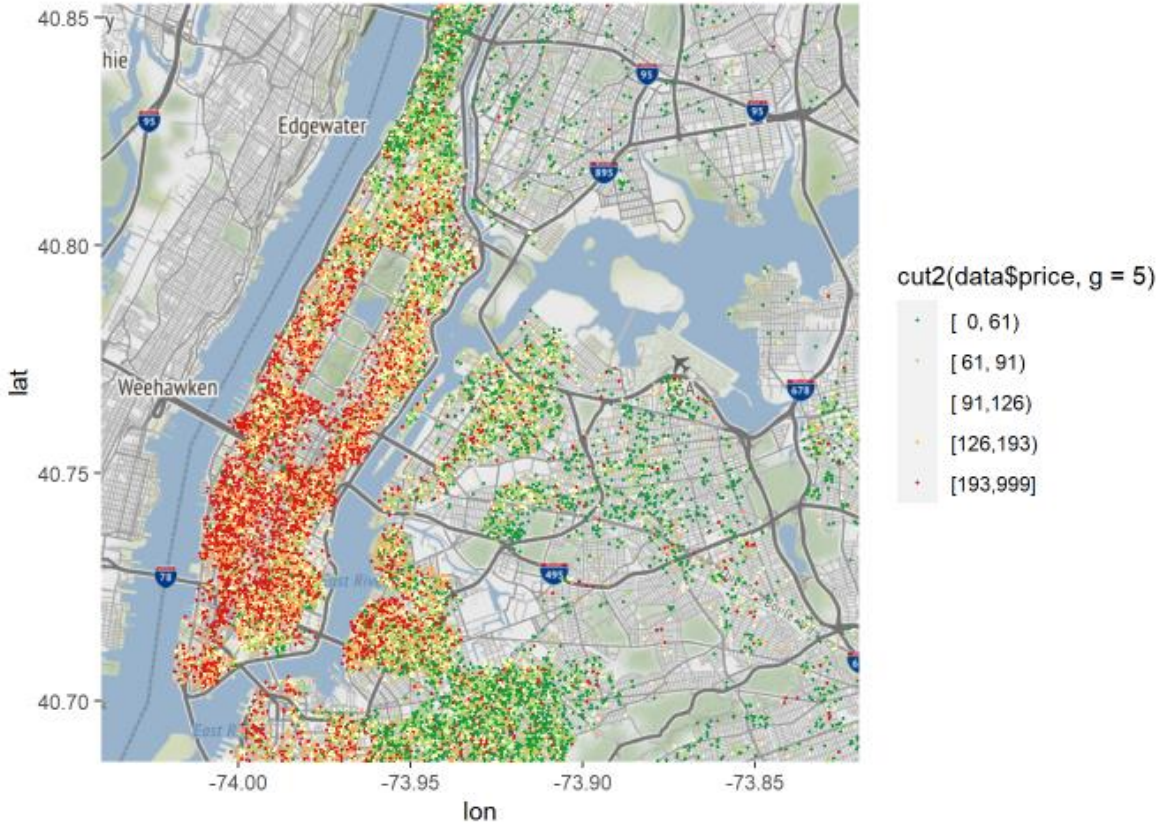


Terrain Map of Manhattan


```
manhattan = get_map(location=c(-73.93,40.77), zoom=12, scale=4, maptype='terrain', source='stamen')
ggmap(manhattan)
```

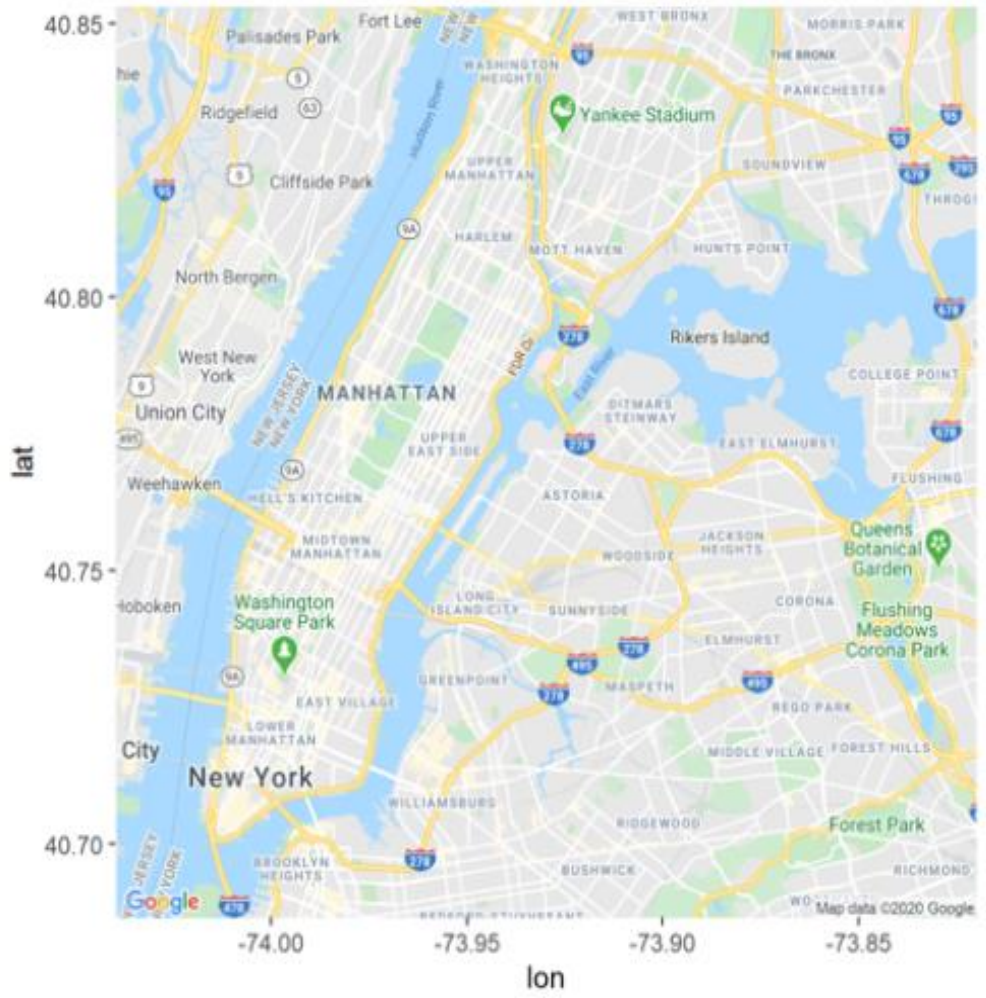
Manhattan (Terrain) + Rentals + Price

```
ggmap(manhattan)+
  geom_point(data=data,aes(x=longitude,y=latitude,color=cut2(data$price,g=5)), size=0.3)+
  scale_color_brewer(type='seq', palette='RdYlGn', direction=-1)
```



Roadmap of Manhattan

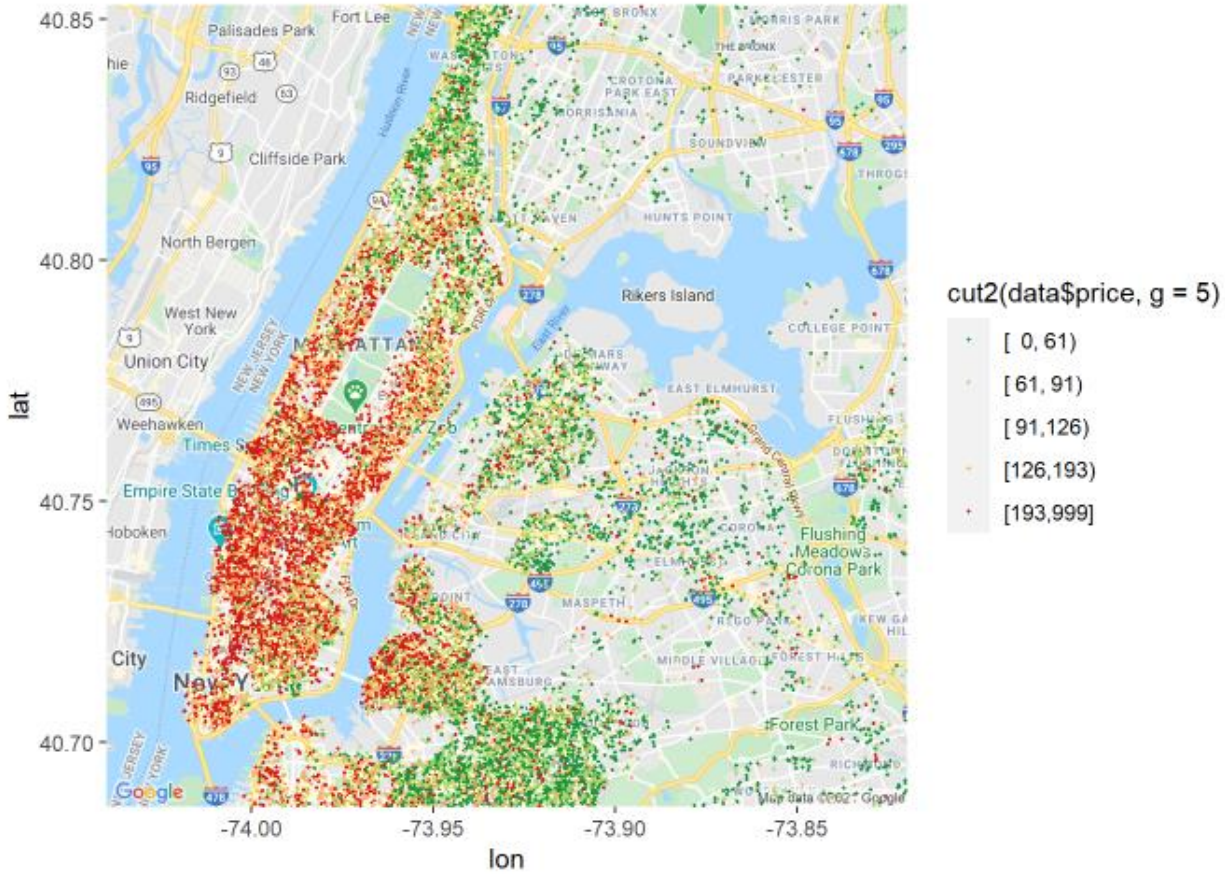
```
manhattan = get_map(location = c(-73.93,40.77),zoom = 12,scale = 4,maptype = 'roadmap')
ggmap(manhattan)
```



- Road Map of NYC + Rental Locations + Price

Manhattan (Roadmap) + Rentals + Price

```
ggmap(manhattan)+
  geom_point(data=data,aes(x=longitude,y=latitude,color=cut2(data$price,g=5)), size=0.3)+
  scale_color_brewer(type='seq', palette='RdYlGn', direction=-1)
```



Map of New York

Let's get familar with ggmaps and google maps by plotting Cold Stone Creamery (73.98W, 40.75N).
You will note ggmap uses a layered approach to plotting like ggplot2. However, unlike ggplot2 where the data is typically in the ggplot() layer, when using ggmap() the data pushed down to the plotting layers (e.g., geom_point, geom_label)

```
coldStoneCreamery_street = get_map(location=c(-73.988640,40.757020), zoom=18 , scale=2)
```

```
## Source : https://maps.googleapis.com/maps/api/staticmap?center=40.75702,-73.98864&zoom=18&size=640x64
0&scale=2&maptype=terrain&language=en-EN&key=xxx-x-ND2iyfra0D14HsZjfL_TKql14
```

```
ggmap(coldStoneCreamery_street)+
  geom_point(data=data.frame(lon=-73.988640,lat=40.757020),
             mapping=aes(lon,lat),size=4,color='steelblue')+
  geom_label(data=data.frame(lon=-73.988640,lat=40.757020),
             mapping=aes(lon,lat), label='Cold Stone Creamery', nudge_y=0.0002, size=2)
```

Mapping Data using library(tigris)

Above maps rely on Google API, which for now is free. An alternative is to use US Census Shapefiles from the TIGER/Line geodatabase. tigris is an R package that allows users to directly download and use TIGER/Line shapefiles. These files belong to the class sp.

```
library(tigris)
manhattan_tigris = tracts(state='NY', county='New York', cb=TRUE)
```

R: Download a Census tracts shapefile into R, and optionally... ▾ Find in Topic

tracts {tigris} R Documentation

Download a Census tracts shapefile into R, and optionally subset by county

Description

Description from the US Census Bureau (see link for source): Census Tracts are small, relatively permanent statistical subdivisions of a county or equivalent entity that are updated by local participants prior to each decennial census as part of the Census Bureau's Participant Statistical Areas Program. The Census Bureau delineates census tracts in situations where no local participant existed or where state, local, or tribal governments declined to participate. The primary purpose of census tracts is to provide a stable set of geographic units for the presentation of statistical data.

Usage

```
tracts(state, county = NULL, cb = FALSE, year = NULL, ...)
```

Arguments

`state`	The two-digit FIPS code (string) of the state you want. Can also be state name or state abbreviation.
`county`	The three-digit FIPS code (string) of the county you'd like to subset for, or a vector of FIPS codes if you desire multiple counties. Can also be a county name or vector of names.
`cb`	If cb is set to TRUE, download a generalized (1:500k) tracts file. Defaults to FALSE (the most detailed TIGER/Line file)
`year`	defaults to 2016
`...`	arguments to be passed to the underlying 'load_tiger' function, which is not exported. Options include `refresh`, which specifies whether or not to re-download shapefiles (defaults to `FALSE`).

Map of Manhattan with library(tmap)

library(tmap) offers a flexible, layer-based, and easy to use approach to create thematic maps, such as choropleths and bubble maps. It only works with sp, sf or raster objects, but that is okay because most spatial files are spatial objects.

```
library(tmap)
tm_shape(manhattan_tigris)+
  tm_borders()
```

tm_borders Create polygon borders

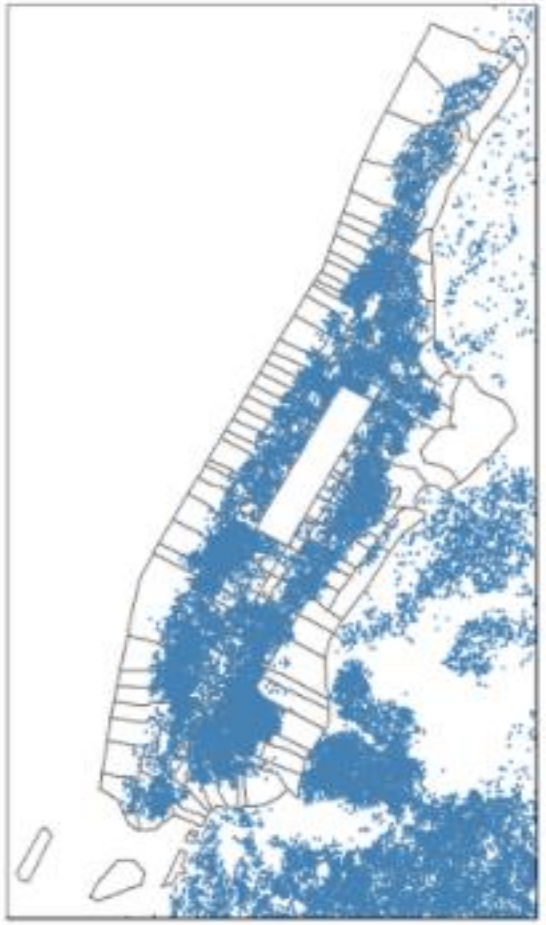
Convert DataFrame to sp

Of course, if the data is not a spatial object, it will have to be converted to one before it can be plotted using library(tmap). Furthermore, to overlay sp class data on a map, both must have the same projection.

```
library(sp)
data_sp = data
coordinates(data_sp) =~longitude+latitude
```

Map of Manhattan + Rentals

```
tm_shape(manhattan_tigris)+
  tm_borders()+
  tm_shape(data_sp)+
  tm_dots(col='steelblue', size=0.01)
```



Using Shape files

Read Shape File

library(rgdal) has a handy function to open shapefiles. In the code below, 'nyc_shape' is a folder on my computer that contains a set of nyc shape files that have been gathered from the open New York Neighborhood Tabulation Areas (NTA) database (data pulled on Aug.5, 2019)

Neighborhood Tabulation Areas (NTAs)

```
library(sp)
library(rgdal)
nyc = readOGR(dsn='nyc_shape', layer='nynta')
```

use readOGR() from the rgal package

```
## OGR data source with driver: ESRI Shapefile
## Source: "C:\Users\Kitty\Desktop\AAFM2 2021S\11 Spatial Analysis\Post\nyc_shape", layer: "nynta"
## with 195 features
## It has 7 fields
```

- Example

  To read a shapefile called California.shp with all its associated files in a directory called CA on the desktop:

  Set working directory to desktop folder and use the following code

  readOGR(dsn = "CA", layer = "California")

Examine the Spatial File

Object Type

```
class(nyc) # object type
```

```
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"
```

Coordinate system

```
summary(nyc)  # coordinate system
```

```
## Object of class SpatialPolygonsDataFrame
## Coordinates:
##        min       max
## x 913175.1 1067382.5
## y 120121.9  272844.3
## Is projected: TRUE
## proj4string :
## [+proj=lcc +lat_0=40.1666666666667 +lon_0=-74 +lat_1=41.0333333333333
## +lat_2=40.6666666666667 +x_0=300000 +y_0=0 +datum=NAD83 +units=us-ft
## +no_defs]
## Data attributes:
##     BoroCode   BoroName          CountyFIPS          NTACode
##  Min.   :1   Length:195         Length:195         Length:195
##  1st Qu.:2   Class :character   Class :character   Class :character
##  Median :3   Mode  :character   Mode  :character   Mode  :character
##  Mean   :3
##  3rd Qu.:4
##  Max.   :5
```

- Let's practice accessing slots by exploring the way polygons are stored inside SpatialDataFrame objects.

- summary()
  provides a concise description of the object, including its class (in this case SpatialPolygonsDataFrame), the extent of the spatial data, the Coordinate Reference System (CRS) information, and data attributes.
  - Extent: the spatial extent of the shapefile (geographic area that the shapefile covers) .
  - Coordinate Reference System (CRS): the projection of the data.

```
##    NTAName           Shape_Leng        Shape_Area
##  Length:195         Min.   : 11988   Min.   :  5581768
##  Class :character   1st Qu.: 23921   1st Qu.: 19392232
##  Mode  :character   Median : 30549   Median : 32629789
##                     Mean   : 42006   Mean   : 43228063
##                     3rd Qu.: 41877   3rd Qu.: 50223047
##                     Max.   :490421   Max.   :327765030
```

Slots

```
str(nyc,max.level = 2) # slots in sp object
```

```
## Formal class 'SpatialPolygonsDataFrame' [package "sp"] with 5 slots
##   ..@ data       :'data.frame':  195 obs. of  7 variables:
##   ..@ polygons   :List of 195
##   ..@ plotOrder  : int [1:195] 113 182 181 133 174 185 140 98 143 137 ...
##   ..@ bbox       : num [1:2, 1:2] 913175 120122 1067383 272844
##   .. ..- attr(*, "dimnames")=List of 2
##   ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
```

- argument max.level in str() function: restricts how far down the hierarchy of the object str() prints.

Projection Type

```
proj4string(nyc)
```

```
## [1] "+proj=lcc +lat_0=40.1666666666667 +lon_0=-74 +lat_1=41.0333333333333 +lat_2=40.6666666666667
+x_0=300000 +y_0=0 +datum=NAD83 +units=us-ft +no_defs"
```

```
nyc@proj4string
```

```
## CRS arguments:
##  +proj=lcc +lat_0=40.1666666666667 +lon_0=-74 +lat_1=41.0333333333333
## +lat_2=40.6666666666667 +x_0=300000 +y_0=0 +datum=NAD83 +units=us-ft
## +no_defs
```

Bounding Box

```
nyc@bbox # bounding box
```

```
##        min       max
## x 913175.1 1067382.5
## y 120121.9  272844.3
```

Data Slot

```
head(nyc@data) # data slot
```

BoroCode <int>	BoroName <chr>	CountyFIPS <chr>	NTAC... <chr>	NTAName <chr>	Shape_Leng <dbl>	Shape_Area <dbl>
0	3	Brooklyn	047	BK88	Borough Park	39247.23	54005019
1	4	Queens	081	QN51	Murray Hill	33266.90	52488277
2	4	Queens	081	QN27	East Elmhurst	19816.71	19726846
3	4	Queens	081	QN07	Hollis	20976.34	22887773
4	1	Manhattan	061	MN06	Manhattanville	17040.69	10647077
5	4	Queens	081	QN02	Springfield Gardens North	25433.58	28428438

6 rows

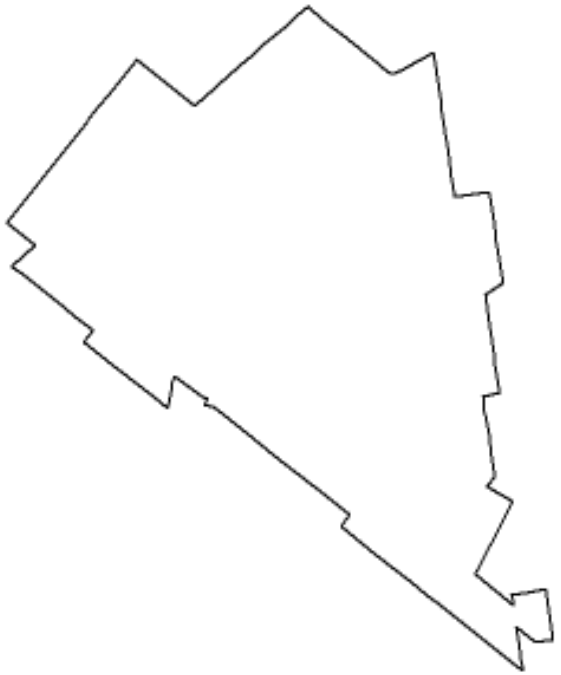
Subsetting a SpatialPolygonsDataFrame

```
str(nyc[1,], max.level=2)
```

```
## Formal class 'SpatialPolygonsDataFrame' [package "sp"] with 5 slots
##   ..@ data       :'data.frame':  1 obs. of  7 variables:
##   ..@ polygons   :List of 1
##   ..@ plotOrder  : int 1
##   ..@ bbox       : num [1:2, 1:2] 982208 162478 991773 174108
##   .. ..- attr(*, "dimnames")=List of 2
##   ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
```

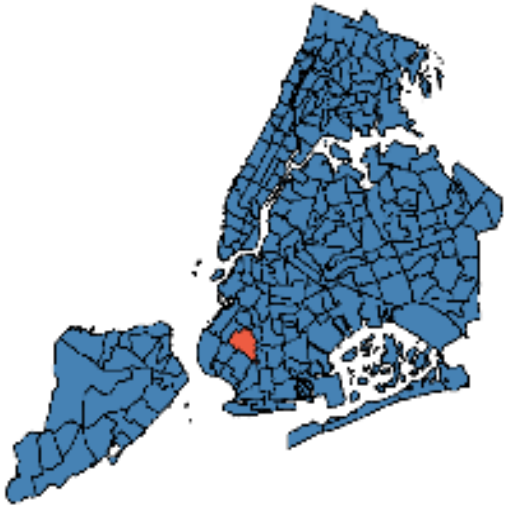
Extract rows. Subset the first census tract in the dataset and plot it

```
plot(nyc[1,])
```



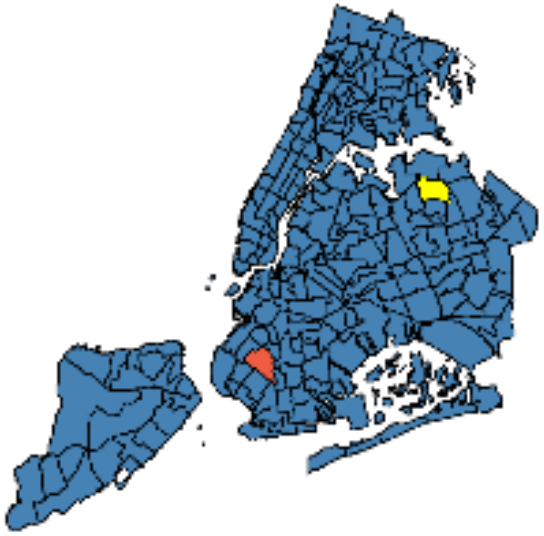
But, geographical entities only make sense in context. So, let us overlay census tract 1 on a plot of nyc.

```
plot(nyc,col='steelblue')
plot(nyc[1,], add=T, col='tomato2')
```



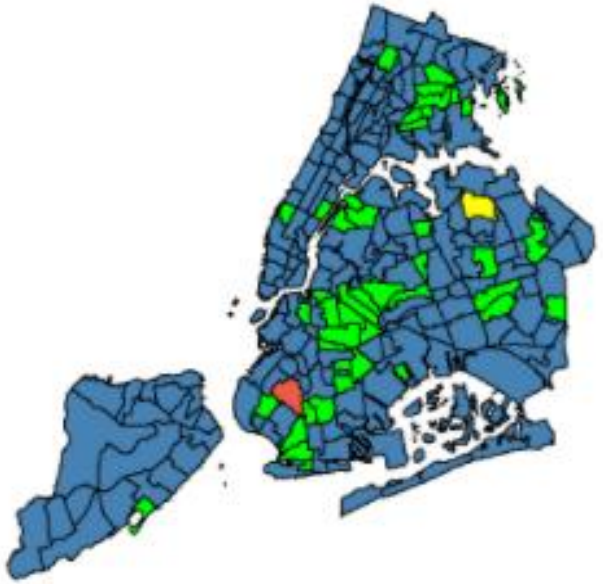
Next, extract census tract 2

```
plot(nyc,col='steelblue')
plot(nyc[1,], add=T, col='tomato2')
plot(nyc[2,], add=T, col='yellow')
```



Finally, extracust tracts 20:60 and plot.

```
plot(nyc,col='steelblue')
plot(nyc[1,], add=T, col='tomato2')
plot(nyc[2,], add=T, col='yellow')
plot(nyc[20:60,], add=T, col='green')
```



Extract Columns

```
nyc@data$BoroName #or nyc$BoroName
```

```
##   [1] "Brooklyn"      "Queens"        "Queens"        "Queens"
##   [5] "Manhattan"     "Queens"        "Queens"        "Brooklyn"
##   [9] "Bronx"         "Queens"        "Manhattan"     "Queens"
##  [13] "Queens"        "Queens"        "Brooklyn"      "Brooklyn"
##  [17] "Brooklyn"      "Bronx"         "Brooklyn"      "Queens"
##  [21] "Queens"        "Bronx"         "Queens"        "Queens"
##  [25] "Brooklyn"      "Brooklyn"      "Brooklyn"      "Brooklyn"
##  [29] "Brooklyn"      "Brooklyn"      "Brooklyn"      "Brooklyn"
##  [33] "Brooklyn"      "Brooklyn"      "Bronx"         "Brooklyn"
##  [37] "Brooklyn"      "Brooklyn"      "Brooklyn"      "Queens"
##  [41] "Brooklyn"      "Bronx"         "Brooklyn"      "Manhattan"
##  [45] "Queens"        "Queens"        "Bronx"         "Bronx"
##  [49] "Bronx"         "Bronx"         "Brooklyn"      "Brooklyn"
##  [53] "Queens"        "Bronx"         "Queens"        "Queens"
##  [57] "Staten Island" "Manhattan"     "Queens"        "Queens"
##  [61] "Queens"        "Queens"        "Queens"        "Queens"
##  [65] "Queens"        "Bronx"         "Queens"        "Queens"
##  [69] "Queens"        "Bronx"         "Brooklyn"      "Queens"
##  [73] "Brooklyn"      "Brooklyn"      "Queens"        "Queens"
##  [77] "Queens"        "Queens"        "Queens"        "Brooklyn"
##  [81] "Queens"        "Bronx"         "Bronx"         "Staten Island"
##  [85] "Staten Island" "Bronx"         "Staten Island" "Brooklyn"
##  [89] "Brooklyn"      "Staten Island" "Bronx"         "Queens"
##  [93] "Queens"        "Queens"        "Manhattan"     "Manhattan"
```

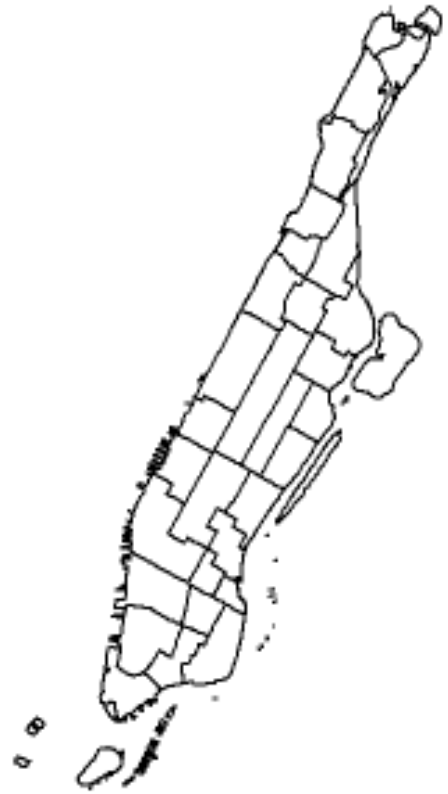
```
## [153] "Bronx"         "Bronx"         "Manhattan"     "Bronx"
## [157] "Bronx"         "Bronx"         "Manhattan"     "Manhattan"
## [161] "Manhattan"     "Manhattan"     "Bronx"         "Manhattan"
## [165] "Manhattan"     "Manhattan"     "Manhattan"     "Bronx"
## [169] "Bronx"         "Bronx"         "Bronx"         "Bronx"
## [173] "Bronx"         "Bronx"         "Bronx"         "Bronx"
## [177] "Manhattan"     "Manhattan"     "Bronx"         "Bronx"
## [181] "Brooklyn"      "Queens"        "Brooklyn"      "Brooklyn"
## [185] "Staten Island" "Queens"        "Queens"        "Staten Island"
## [189] "Staten Island" "Queens"        "Queens"        "Staten Island"
## [193] "Staten Island" "Brooklyn"      "Brooklyn"
```

```
table(nyc$BoroName)
```

```
##
##         Bronx      Brooklyn     Manhattan        Queens Staten Island
##            38            51            29            58            19
```

Plot only Manhattan

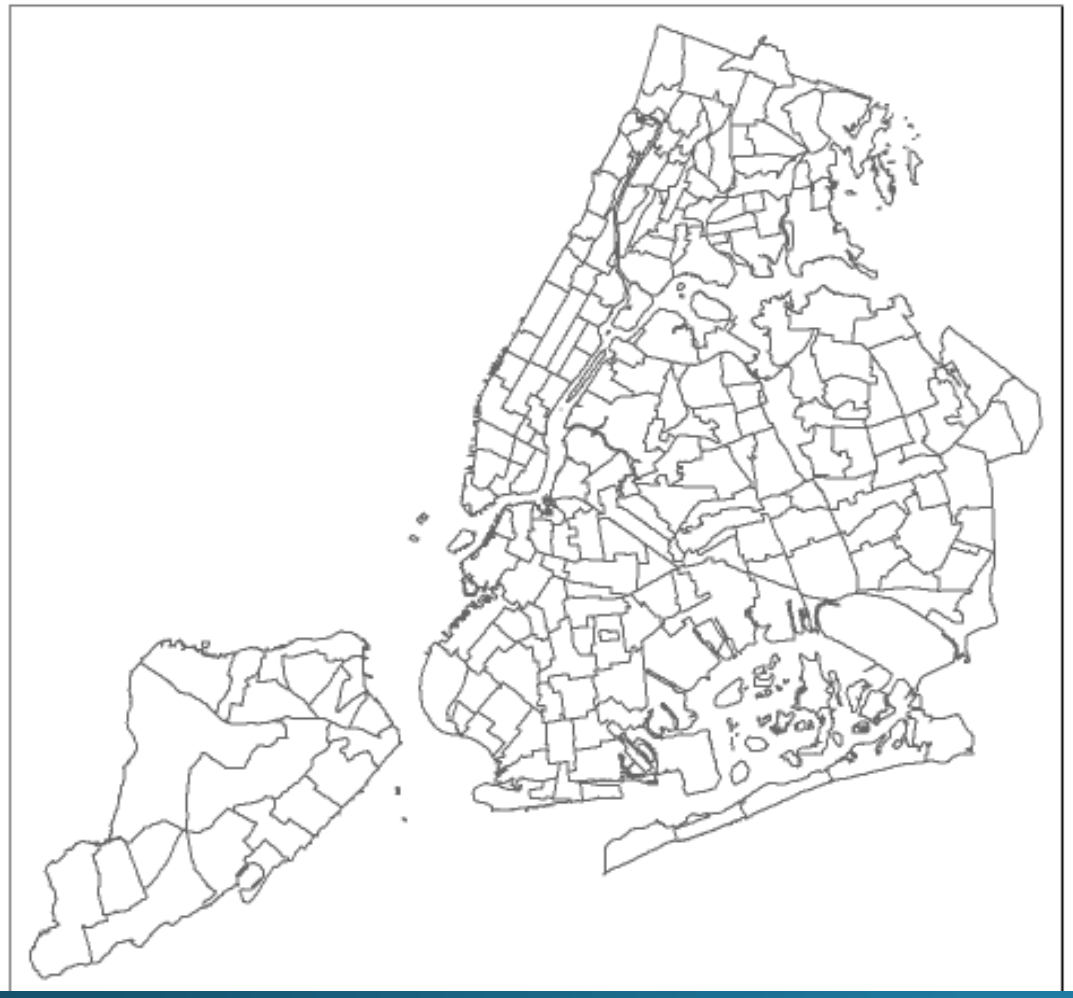
```
plot(nyc[nyc$BoroName=='Manhattan',])
```



Maps with library(tmap)

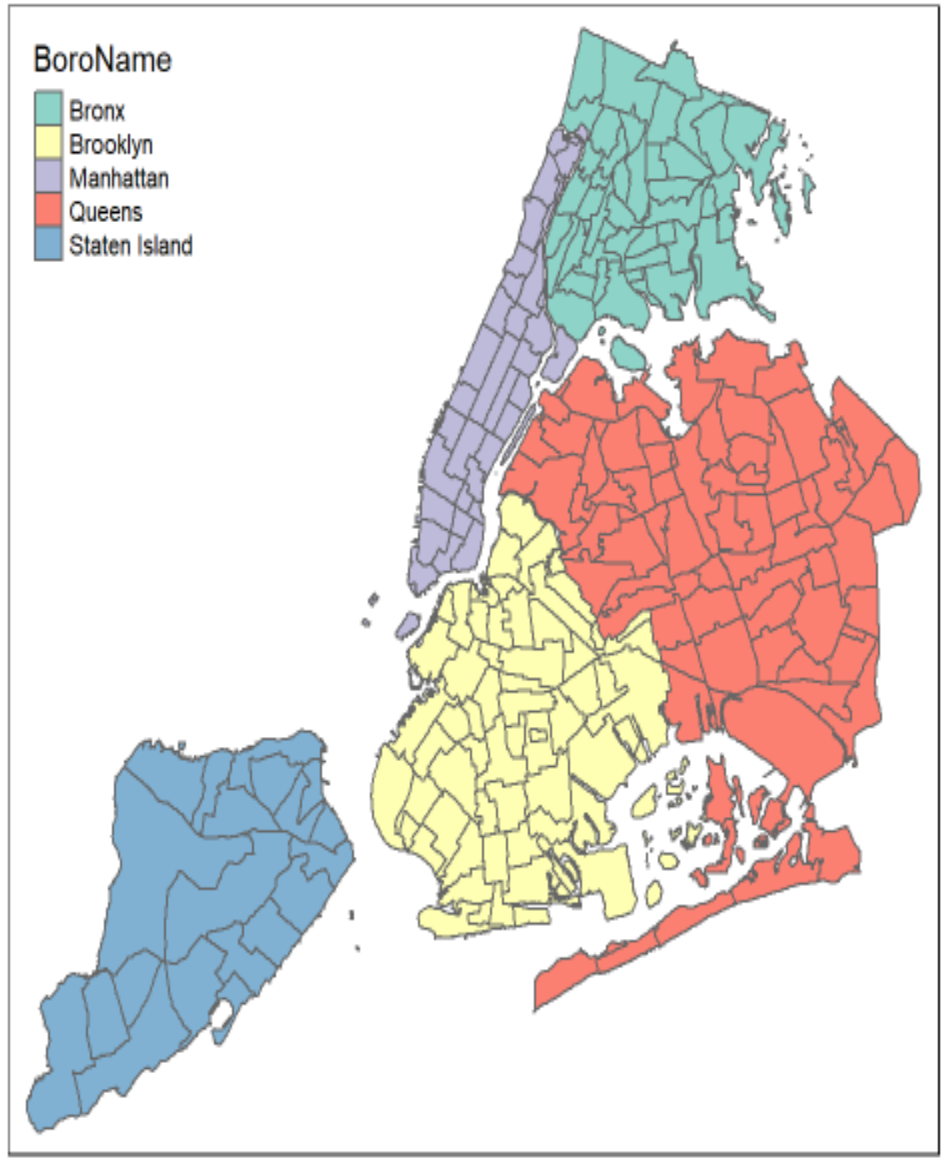
Plot the five boroughs Let's visualizing with library(tmap) Learn more about library(tmap) in this vignette

```
library(tmap)
tm_shape(nyc)+
  tm_borders()
```

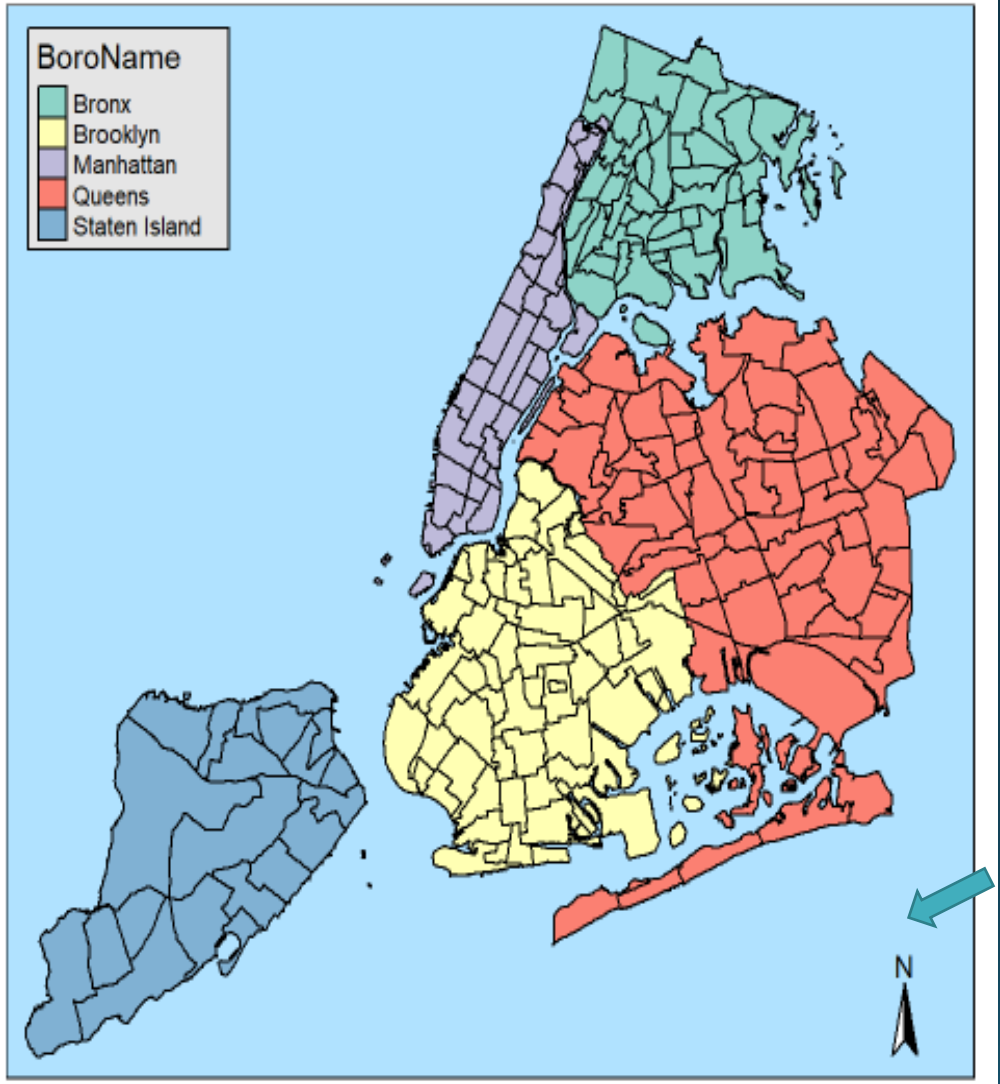


- tm_shape() defines default data for any subsequent layers
- tm_boarders() to show outlines

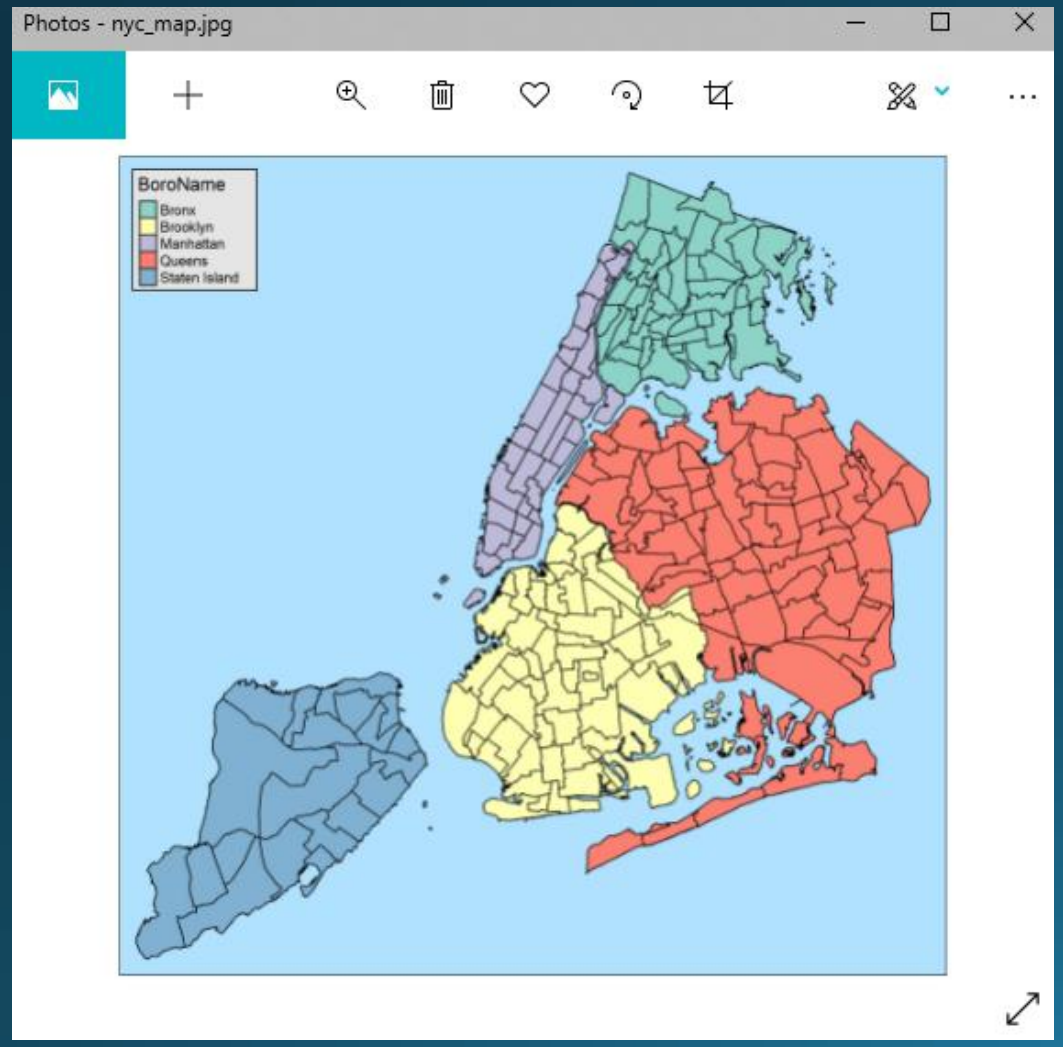
```
tm_shape(nyc)+
  tm_polygons('BoroName')
```

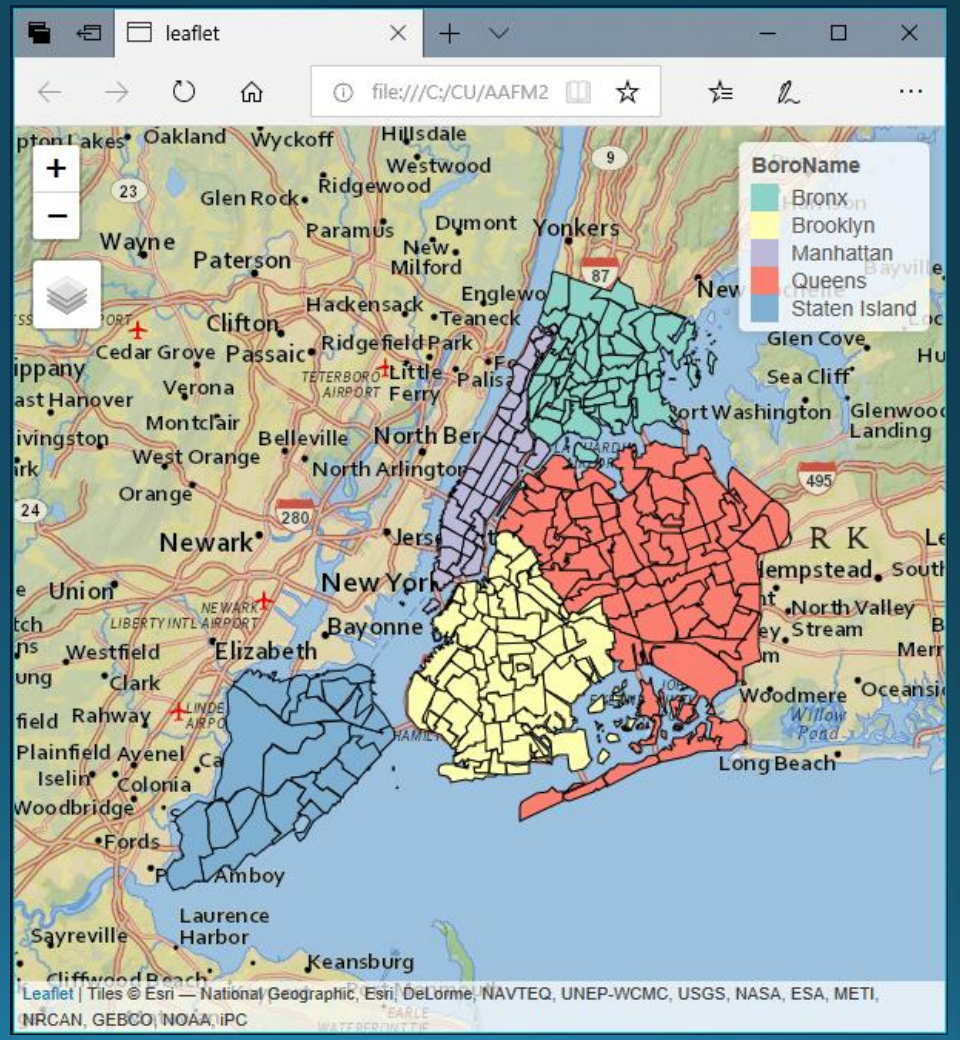



```
tm_shape(nyc)+
  tm_polygons('BoroName')+
  tm_style('natural')+  # works like library(ggthemes)
  tm_compass()
```



```
nyc_map =
  tm_shape(nyc)+
  tm_polygons('BoroName')+
  tm_style('natural')  # works like library(ggthemes)
tmap_save(nyc_map, filename='nyc_map.jpg')  # picture files
tmap_save(nyc_map, filename='nyc_map.html')  # Saving as html creates an interactive map
```





Merge Data with a Map

Let us plot population data on to a map of New York. Population Data is available from Open New York. In this instance, I have downloaded data on to my local machine before reading it into R. This is a good practice as repeatedly downloading relatively static data (population data is not updated daily or even weekly) by reading from the website directly places an unnecessary burden on the server.
Here, we read the data and create a field for change in population.

```
pop = read.csv('nyc_pop.csv')
library(dplyr)
library(tidyr)
nyc_pop =
  pop%>%
  spread(key=Year, value=Population)%>%
  rename(year_2000='2000', year_2010='2010')%>%
  mutate(pop_change=((year_2010-year_2000)/year_2000)*100)
head(nyc_pop)
```

G8

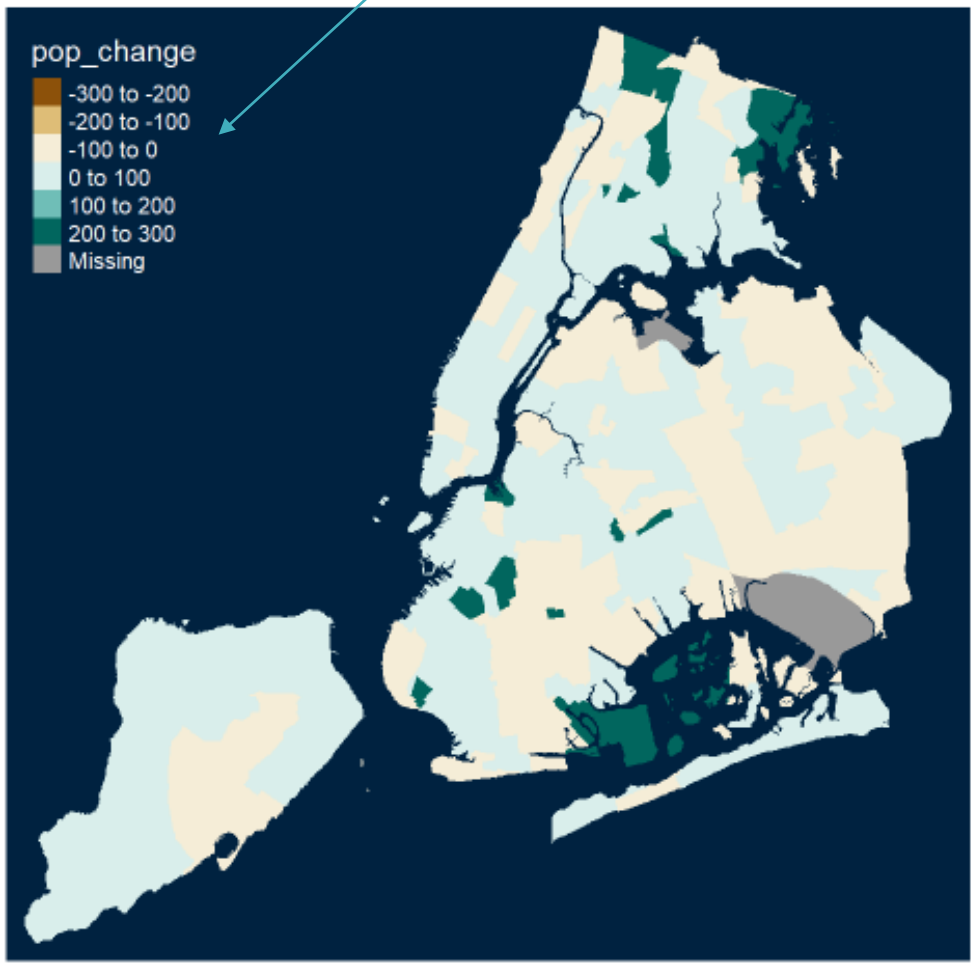
	A	B	C	D	E	F
1	Borough	Year	FIPS County Code	NTA Code	NTA Name	Population
2	Bronx	2000	5	BX01	Claremont-Bathgate	28149
3	Bronx	2000	5	BX03	Eastchester-Edenwald-Baychester	35422
4	Bronx	2000	5	BX05	Bedford Park-Fordham North	55329
5	Bronx	2000	5	BX06	Belmont	25967
6	Bronx	2000	5	BX07	Bronxdale	34309
7	Bronx	2000	5	BX08	West Farms-Bronx River	34542
8	Bronx	2000	5	BX09	Soundview-Castle Hill-Clason Point-Harding Park	50753
9	Bronx	2000	5	BX10	Pelham Bay-Country Club-City Island	27140
10	Bronx	2000	5	BX13	Co-Op City	40676
11	Bronx	2000	5	BX14	East Concourse-Concourse Village	58961
12	Bronx	2000	5	BX17	East Tremont	39280
13	Bronx	2000	5	BX22	North Riverdale-Fieldston-Riverdale	28013
14	Bronx	2000	5	BX26	Highbridge	33844
15	Bronx	2000	5	BX27	Hunts Point	25142
16	Bronx	2000	5	BX28	Van Cortlandt Village	50607

	Borough <fctr>	FIPS.County.Code <int>	NTA.Code <fctr>	NTA.Name <fctr>	year_2000 <int>	year_2010 <int>	pop_change <dbl>
1	Bronx	5	BX01	Claremont-Bathgate	28149	31078	10.405343
2	Bronx	5	BX03	Eastchester-Edenwald-Baychester	35422	34517	-2.554909
3	Bronx	5	BX05	Bedford Park-Fordham North	55329	54415	-1.651937
4	Bronx	5	BX06	Belmont	25967	27378	5.433820
5	Bronx	5	BX07	Bronxdale	34309	35538	3.582150
6	Bronx	5	BX08	West Farms-Bronx River	34542	35011	1.357767

6 rows | 1-5 of 8 columns

Merge and Plot

```
nyc_spdf = merge(x=nyc, y=nyc_pop, by.x='NTACode', by.y='NTA.Code')
tm_shape(nyc_spdf)+
  tm_fill(col='pop_change', palette=brewer.pal(n=7,name='BrBG'),
          breaks=c(-300,-200,-100,0,100,200,300))+
  tm_style('cobalt')
```



- Mapping the fill color with variable pop_change

Summary

- This module addressed the following topics
  - What is spatial analysis and the motivation behind it
  - Explain spatial data
  - Gather and represent data on a map
  - Analyze spatial data
  - Illustration using R