R Data Science: Spatial Data Science 1

steppe

February, 5th, 2022

Contents

Contents

List of Figures

List of Tables

Assigned Reading: Chapter 3 of Spatial Data Science

1 Topology

While in the last lecture we emphasized the geographic components of data on in space, we will know turn our attention to some simple geometric properties of two and three dimensional objects.

Essential to most GIS operations is Topology,

1.1 Theoretical Relations

Most sets of relations between objects in two dimension are considered below by the Dimensionally Extended 9-Intersection Model (DE-9IM). The DE-9IM was developed to query spatial databases and still serves as the standard for describing relations.

In the example in the Upper Middle panel

The dimensions which contain the intersection of the 'Interior' of 'A' to the 'Boundary' of 'B' is equal to a line.

$$dim[I(a) \cap B(b)] = 1$$

- where 'I' is the 'Interior' (of 'a') & 'B' is the 'Boundary' (of 'b')

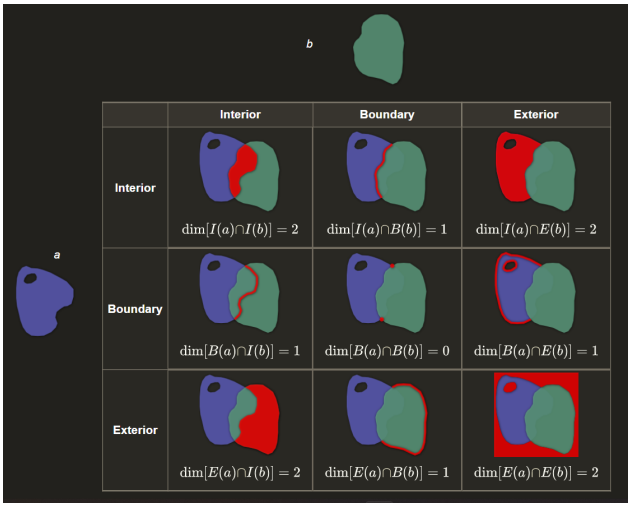


Figure 1: DE-9IM, By Krauss

is the 'Intersection'

- '1' denotes that the product of the intersection is a line

In the example in the Upper Middle panel

The dimensions which contain the intersection of the 'Interior' of 'A' to the 'Boundary' of 'B' is equal to a line.

1.2 Applications

When identifying solutions to a problem we are generally interested in how features affect each other. This schema provides us a framework for organizing our area of analyses, and subsetting the appropriate data.

We are generally interested in :

- *dim[I(a) I(b)] = 2* Which values are in both polygons
- *dim[I(a) B(b)] = 1* Which values are in 'a', and at the boundary of 'b'
- *dim[I(a) E(b)] = 2* Which values are in 'a', but not in 'b'

- *dim[B(a) I(b)] = 1* Which values are at the boundary of 'a', and in 'b'
- *dim[B(a) B(b)] = 0* Which values are at the shared boundaries of 'a' and 'b'
- *dim[B(a) E(b)] = 1* Which values are at the boundary of 'a', and outside 'b'

- *dim[E(a) I(b)] = 2* Which values are in 'b' but not in 'a'
- *dim[E(a) B(b)] = 1* Which values are at the boundary of 'b' and outside 'a'
- *dim[E(a) E(b)] = 2* Which values are outside both 'a' and 'b'

2 Geometric Operations on Spatial Predicates

2.1 Theoretical Properties

Disjoint Neither 'A' nor 'B' touch at any position

Touches The boundaries of 'A' and 'B' touch

Covers Feature 'A' is encapsulated by 'B' on many sides, at least a boundary of 'A' shares at least a partial border with a border of 'B'

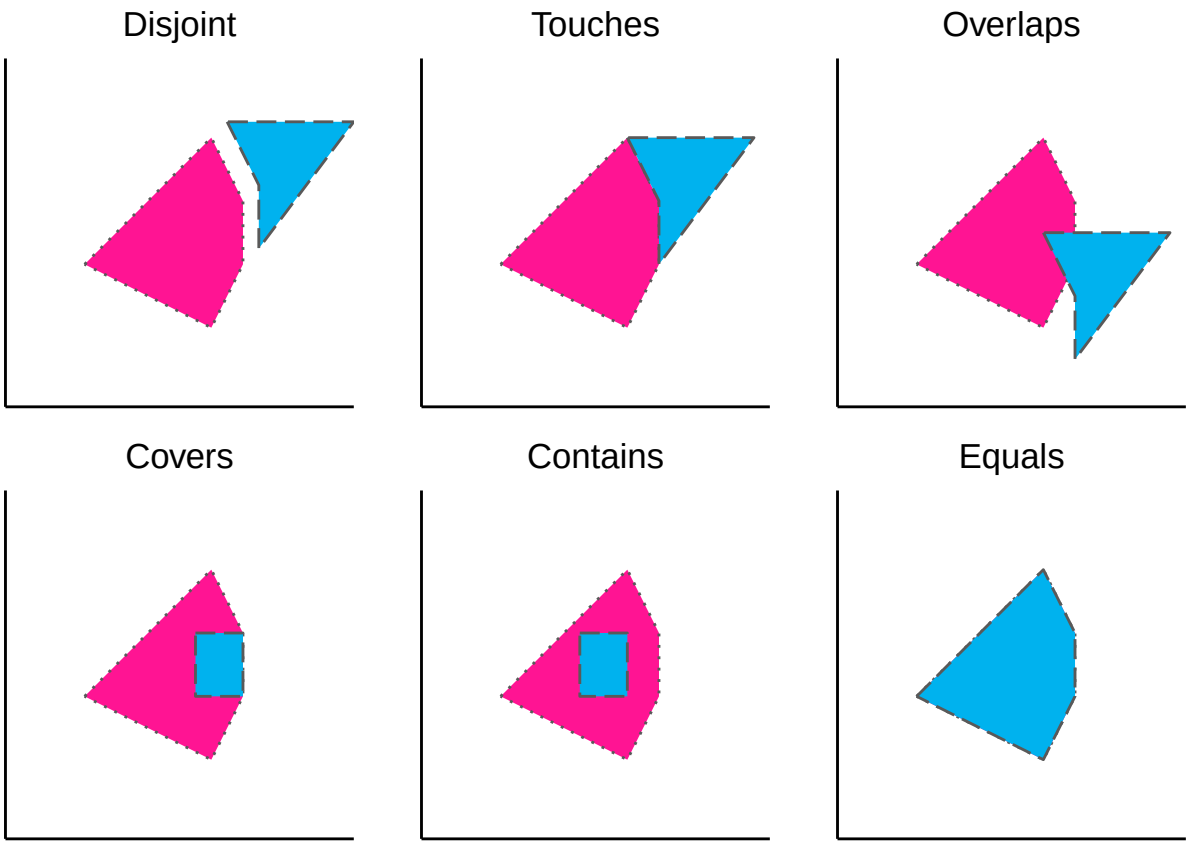


Figure 2: Spatial Predicates after Egenhofer and Herring

Contains All borders of one feature are contained by another

Overlap Portions of the interior of feature 'A' overlap the interior of 'B'

Equals 'A' and 'B' have identical extents.

In addition to these logical tests, postGIS implements the following three tests. 'Intersects' obviously tends to find the most applications, as it generalizes from 3 other logical tests.

Intersects (includes: 'st_contains', 'st_overlaps', 'st_covers')

Within (includes: 'st_contains')

CoveredBy (includes: 'st_equals')

2.2 Applications

We may test these relationships quite readily using the sf package

`st_disjoint(A, B_disjoint, sparse = F) TRUE`

`st_touches(A, B_touches, sparse = F) TRUE`

`st_equals(A, B_overlap, sparse = F): TRUE`

`st_covers(A, B_covers, sparse = F) TRUE`

`st_contains(A, B_contains, sparse = F) TRUE`

`st_equals(A, B_equals, sparse = F): TRUE`

`st_equals(A, B_disjoint, sparse = F): FALSE`

Applications:

`st_intersects(A, B_equals, sparse = F): TRUE`

`st_intersects(A, B_equals, sparse = F): TRUE`

`st_intersects(A, B_equals, sparse = F): TRUE`

`st_within(A, B_equals, sparse = F): FALSE`

`st_coveredby(A, B_equals, sparse = F): TRUE`

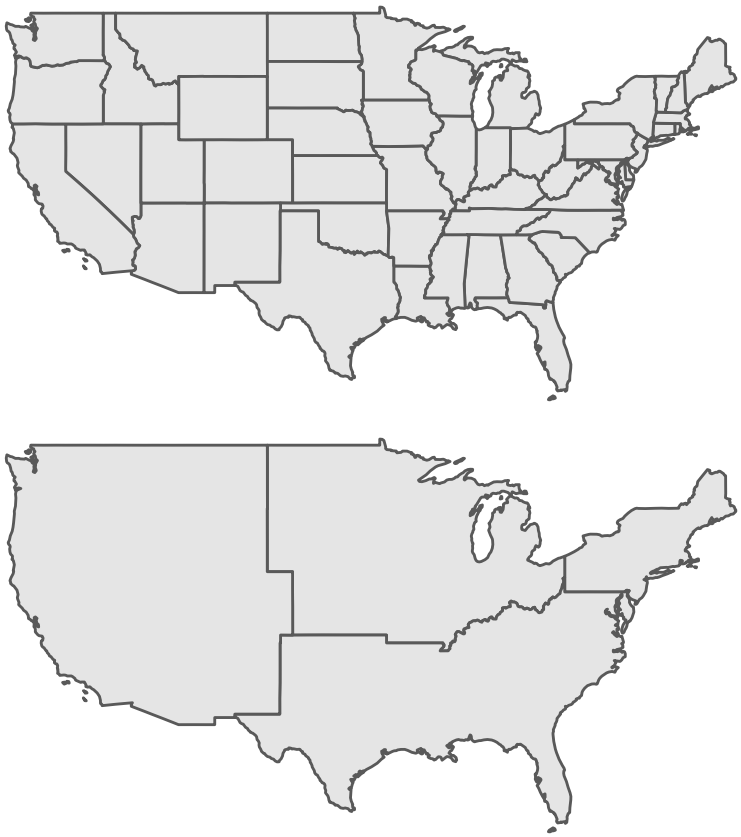
2.3 Spatial Join

- Combine two spatial objects, based on their spatial relations.
- May use nearly all of the predicates above, and more!
- Allows left or inner join (only records present in both objects)

```
nghbrhd_parks <- st_join(chi_neighb, chi_parks,
                join = st_intersects,
                left = TRUE
                ) %>%
  count(pri_neigh)
```

2.4 Aggregate Simple Features via group by & summarize

- Can be used to combine the geometries of features



3 Spatial autocorrelation

3.1 Tobler's First Law

"I invoke the first law of geography: everything is related to everything else, but near things are more related than distant things." - Waldo Tobler

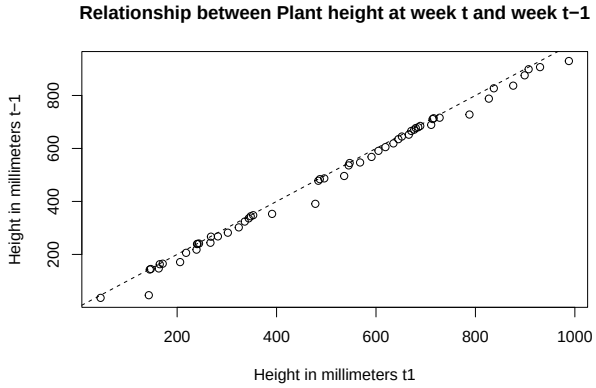
Accordingly, if we want to model processes in space, considerable information may be found from those locations in closet proximity.

3.2 Autocorrelation

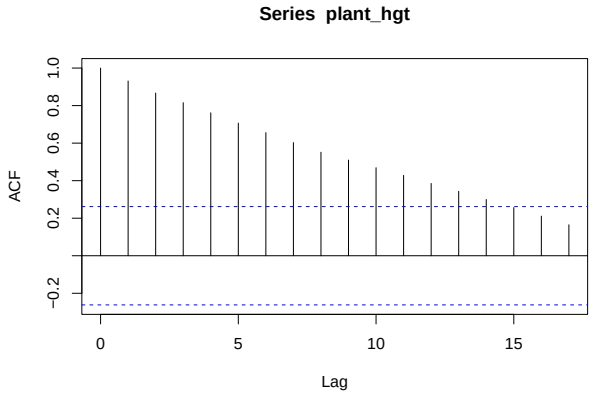
The correlation between two values of a variable as function of the 'lags' between them.

3.2.1 Temporal Autocorrelation

If we were to measure the height of a fast growing plant in a greenhouse every week for a year, each measured value would be strongly related to the value collected the week preceding it.




```
[1] 0.9967624
```



Until we go to around 15 lags away - or in other words 15 weeks back, our data set is significantly auto-correlated. In statistics this will adversely affect the results of our models. However, this information can also be used.

3.3 Spatial Autocorrelation

Spatial auto correlation is a 2 dimensional phenomenon, and is slightly harder to visualize.

3.3.1 Calculate Morans I

To determine whether there is spatial auto-correlation present in a data set we can use Moran's Index.

Here we will see if there is a an spatial pattern in the presence of Park Areas in Chicago. First, we will intersect each park to the neighborhood they are in. Then we will determine what proportion of each neighborhood is Park.

```
Neighbour list object:
Number of regions: 98
Number of nonzero links: 504
Percentage nonzero weights: 5.247813
Average number of links: 5.142857
Link number distribution:

 1  2  3  4  5  6  7  8  9 10
 2  8 12 15 18 20 11  6  5  1
```

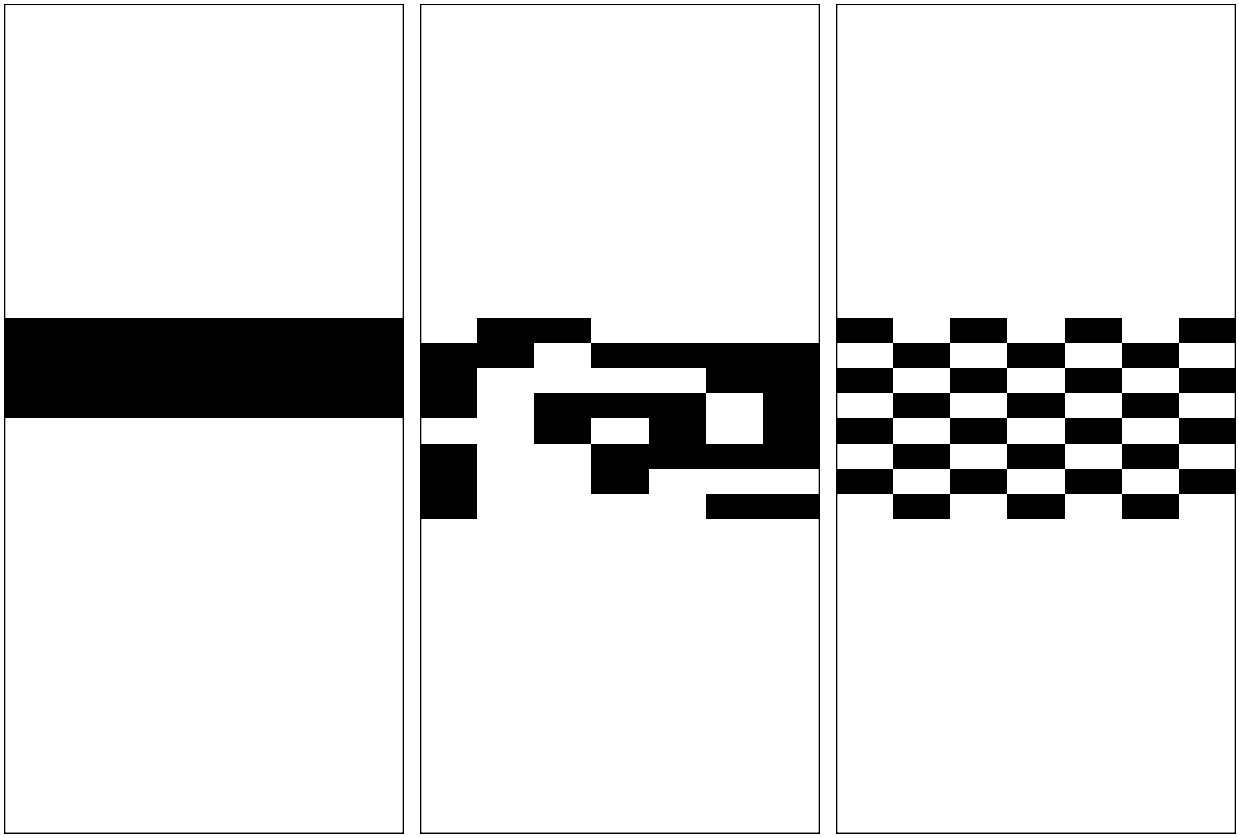


Figure 3: Example figure of spatial autocorrelation. From L to R, Positive, None, Negative

```
2 least connected regions:
Edison Park Boystown with 1 link
1 most connected region:
West Town with 10 links
```

```
    Monte-Carlo simulation of Moran I

data:  chi_neigh_sp$prop_park
weights: spatial_weights_list
number of simulations + 1: 1000

statistic = 0.17899, observed rank = 994, p-value = 0.006
alternative hypothesis: greater
```

The null hypothesis of this test is that the data are randomly distributed The statistic ranges from 1 (perfect clustering of similar values) to -1 (perfect dispersion; opposites values close).

This test tells us that there is some spatial structure in our data, but we do not know the geographic range to which this structuring is present.

If we recall the location of the Chicago Parks, the most noticeable cluster of them is along the Lake Shore. I presume that the Moran.I index is slightly above random due to the presence of these. It also points out an interesting point regarding the history of the development of Chicago. A focus on restricting development along the Shore. Chicago is quite unique in that it's coasts are publicly accessible, and can largely be owed to the contributions of Montgomery Ward.

This amount of spatial auto-correlation would not adversely affect analyses, not too mention, this is an real pattern of these data.

https://www.chicagotribune.com/news/ct-xpm-1995-10-19-9510190079-story.html 02.04.2022 By: Stephen Lee and Tribune Staff Writer. Chicago Tribune. 10.19.1995

https://www.fotp.org/lakefront-protection-and-public-trust.html 02.04.2022

4 Change Grain of Raster Cells.

- Spatial *Extent* of projects often constrained by areas funding (political boundaries via funds from agencies), or computation power.
- *Grain* of projects almost always constrained by computer power, or data sets.

Within an *extent* an analyst wants to find a *grain* relevant to the their study. *Grain* can be, somewhat readily, altered.

4.1 Raster Cell Aggregation (Coarser Grain)

Rasters may be of a resolution which is to fine to warrant use due to computational limits.

4.2 Raster Interpolation (Finer Grain)

- Predict the value of a variable at an un-sampled location
- Using the values of this variable at sampled locations
- Utilizes the property of spatial auto-correlation

While a great number of raster data sets are developed form satellite imagery and represent the classification of observed values at each location in space, other raster data sets have values which are predicted in space. The most evident example of these are raster products of climate variables. All rasters of climate variables are based on measurements taken from meteorological stations, and then the values between these stations are predicted using spatial interpolation.

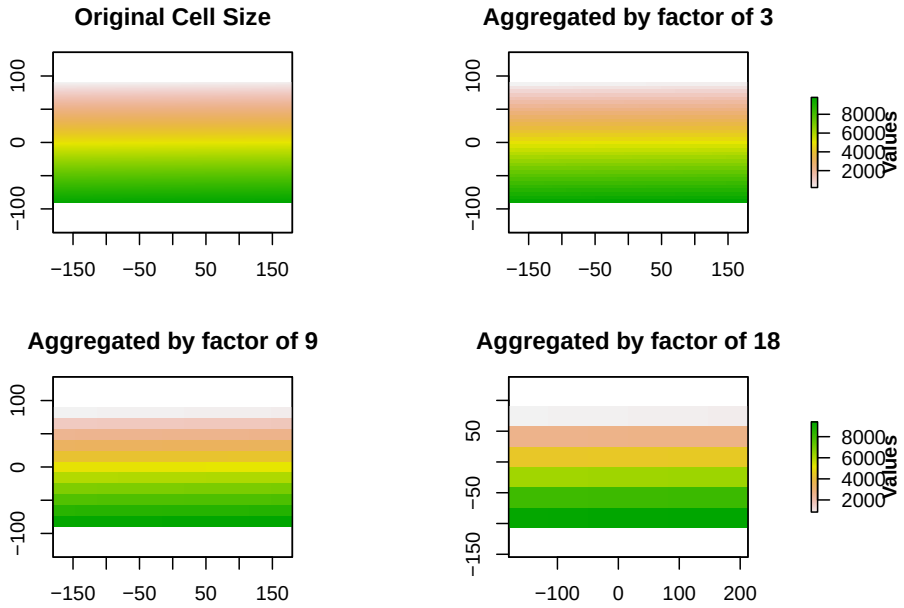


Figure 4: Raster Aggregation

Figure 5: Surveying in the White Cloud Mtns. Idaho, by Hubert Szycygiel

Two of the most commonly used spatial interpolation techniques are 'Inverse Distance Weighting' and 'Kriging' interpolation. Our interaction with these processes will be brief, but we will illustrate their uses so that you may recognize the source of spatial data products in the future.

We will leverage simple versions of these techniques to create Rasters of finer resolution than they are currently.

4.2.1 Inverse Distance Weighting Theory

- First Interpolation Method
- Values at points further in space have less weight
- Scale of weight generally decreases linearly or linear^2

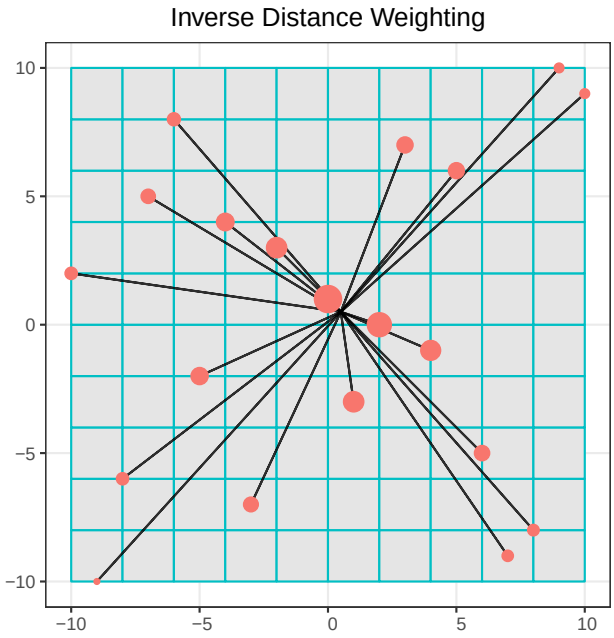


Figure 6: Inverse Distance Weighing Interpolation Theory

```
Warning:
Grid searches over lambda (nugget and sill variances) with  minima at the endpoints:
  (GCV) Generalized Cross-Validation
   minimum at  right endpoint  lambda  =  1.16197e-06 (eff. df= 728.65 )
```

To read more about Inverse Distance Weighting, and view worked out calculations of the process please visit: https://www.geo.fu-berlin.de/en/v/soga/Geodata-analysis/geostatistics/Inverse-Distance-Weighting/index.html

Inverse Distance Weighting is the original spatial interpolation technique and makes great use of Tobler's First Law. While using this technique, we want to predict the value of a variable, such as rainfall, in a location.

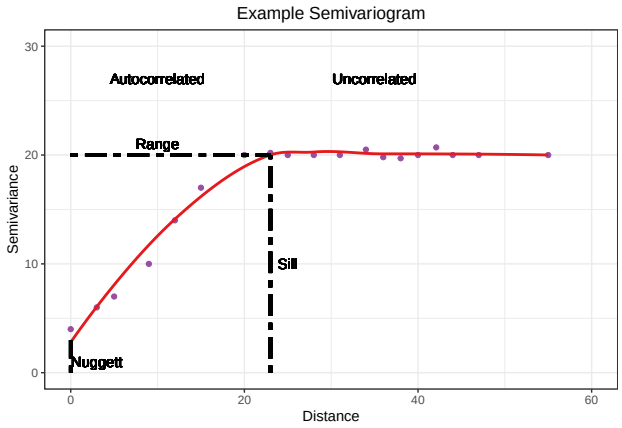
4.3 Kriging Interpolation

https://mgimond.github.io/Spatial/interpolation-in-r.html

4.3.1 Semivariogram

A semivariogram is a method for measuring the correlation between observations at sets of distance.

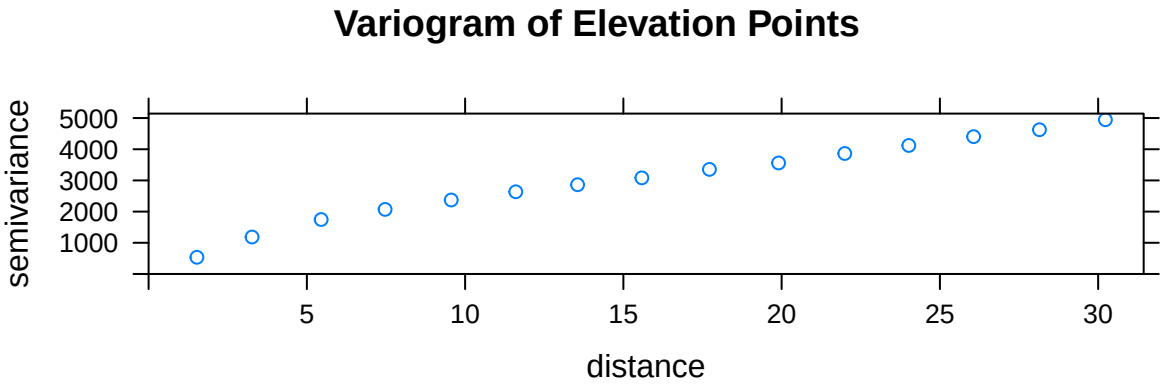
```
## `geom_smooth()` using formula 'y ~ x'
```

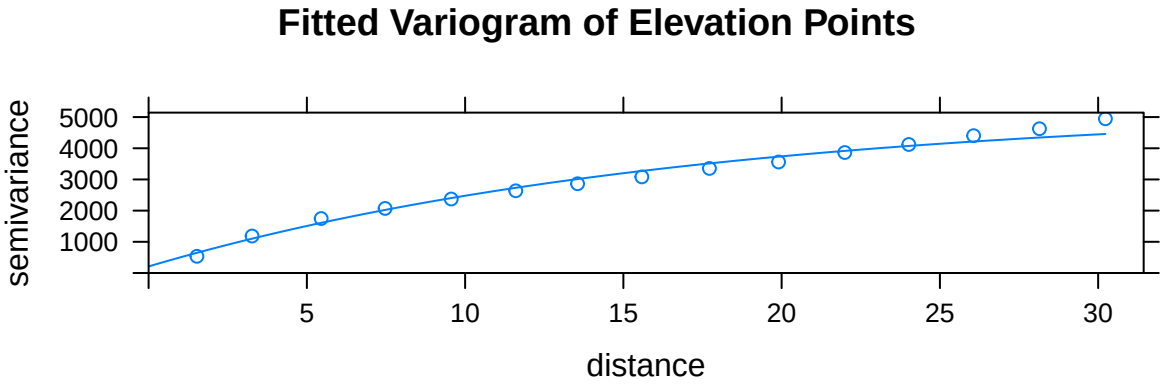


- Nugget: Variation due to errors in the observed measurements.
- Sill: Distance at which the maximum amount of variance in the data is reached.
- Range: Distance at which observations are independent.
- Model: The red line indicates a model which has been fitted to the points (observations), in this example with fictitious data the model is closet to a spherical model.

To determine the range of spatial auto-correlation we can use a Semivariogram

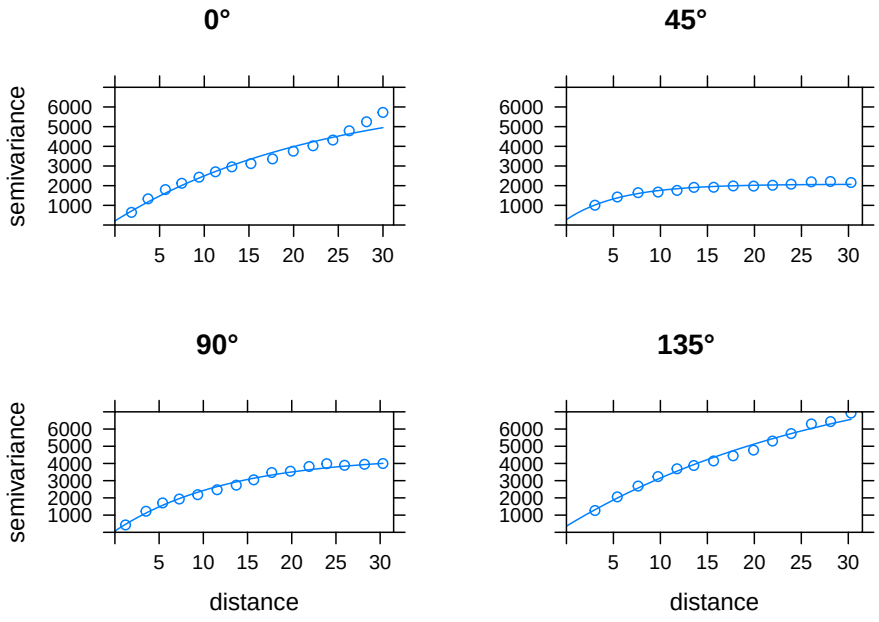
```
variogram_elevation <- variogram(rast_vals~1, data = points)

FittedModel <- fit.variogram(variogram_elevation, vgm(c( "Exp", "Sph", "Gau", "Mat"),
                                  fit.kappa = TRUE))

a <- plot(variogram_elevation, main = "Variogram of Elevation Points")
b <- plot(variogram_elevation, model=FittedModel, main = "Fitted Variogram of Elevation Points", yaxt="n
cowplot::plot_grid(a, b,  ncol = 1)
```






```
rm(variogram_elevation)
```

We can also view directions with predominant auto-correlation.



We see that heading in both 45 and 90 degree angles the effect of spatial auto-correlation deteriorates

4.4 Kriging Inteprolation applied

```
## [using ordinary kriging]
```

4.5 Compare Interpolated Values

We can visually assess the results of our two interpolations using maps.

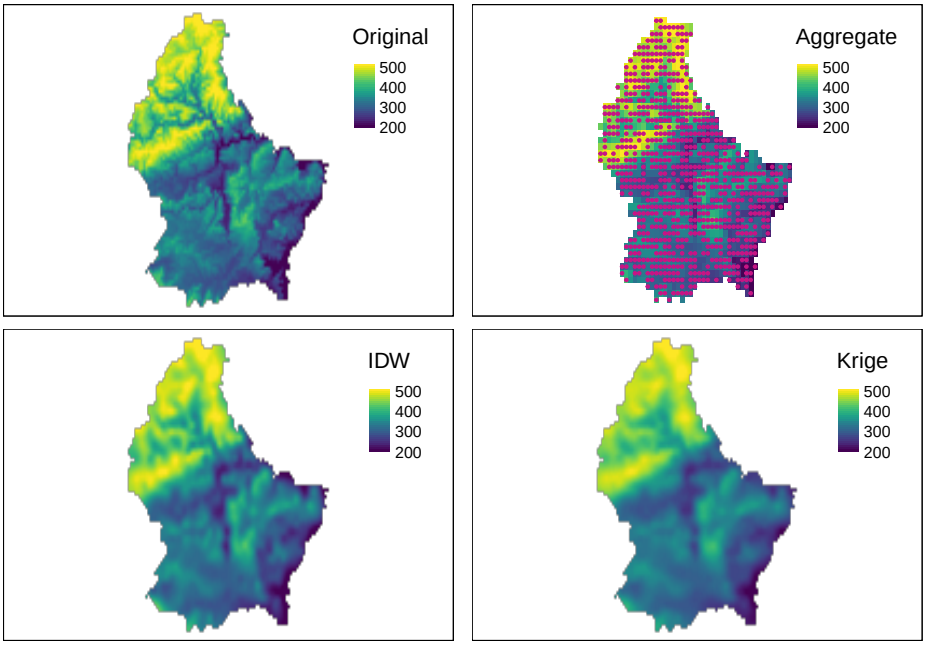
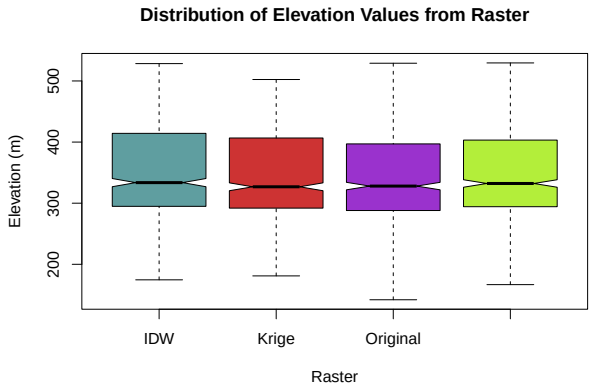


Figure 7: Comparision of Interpolated Values

Remember after importing the *Original* dataset, we merged four cells into one to create our *Aggregate* dataset. We then sampled 1/4 of the aggregated cells, to compute both the *IDW* and *Krige* rasters. we can see that both of them 'smooth' the Aggregated raster. Based on visual inspection, both of our interpolated surfaces appear to match up with the Original raster quite well - for many applications.

```
    Kruskal-Wallis rank sum test

data:  compare$elevation and compare$Type
Kruskal-Wallis chi-squared = 5.1827, df = 3, p-value = 0.1589
```



We can actually check whether the values are similar to the Original raster, and they are.

Kruskal Wallis null hypothesis: the means of each group is the same, we cannot reject the null hypothesis. We see that Krige may 'smooth' the results at either end more than IDW, this is a known parameter of it.

5 Assorted Spatial Vector Operations

This is really a grab bag of what I and others seem to use most the often. We are going to focus on vector data.

5.1 Import a Vector file

- Many formats of vector files may be imported to R.

```
files <- paste0('./spatial_lecture_data/Chicago_Neighborhoods/' ,
                list.files('./spatial_lecture_data/Chicago_Neighborhoods', ".shp"))
chi_neighb <- read_sf(files)
rm(files)
```

5.2 Make a geometry valid as a Simple Feature

You may import vector data which is not compliant with the Standards of the geometry for simple features. If you do this you will get a shocking warning, but the fix is quite simple.

```
p1 = st_as_sfc("POLYGON((0 0, 0 10, 10 0, 10 10, 0 0))")
# create a geometry which will not be valid
```

```
st_is_valid(p1)
st_is_valid(p1, reason = TRUE)
```

```
st_is_valid(p1): FALSE
```

```
st_is_valid(p1, reason = TRUE): Self-intersection[5 5]
```

We see that this geometry contains a line which intersects itself. Ff we refer to our first lecture in our introductory slides on Simple features we can across a rule *"Lines composing polygons cannot intersect"* - the feature we created violates this rule.

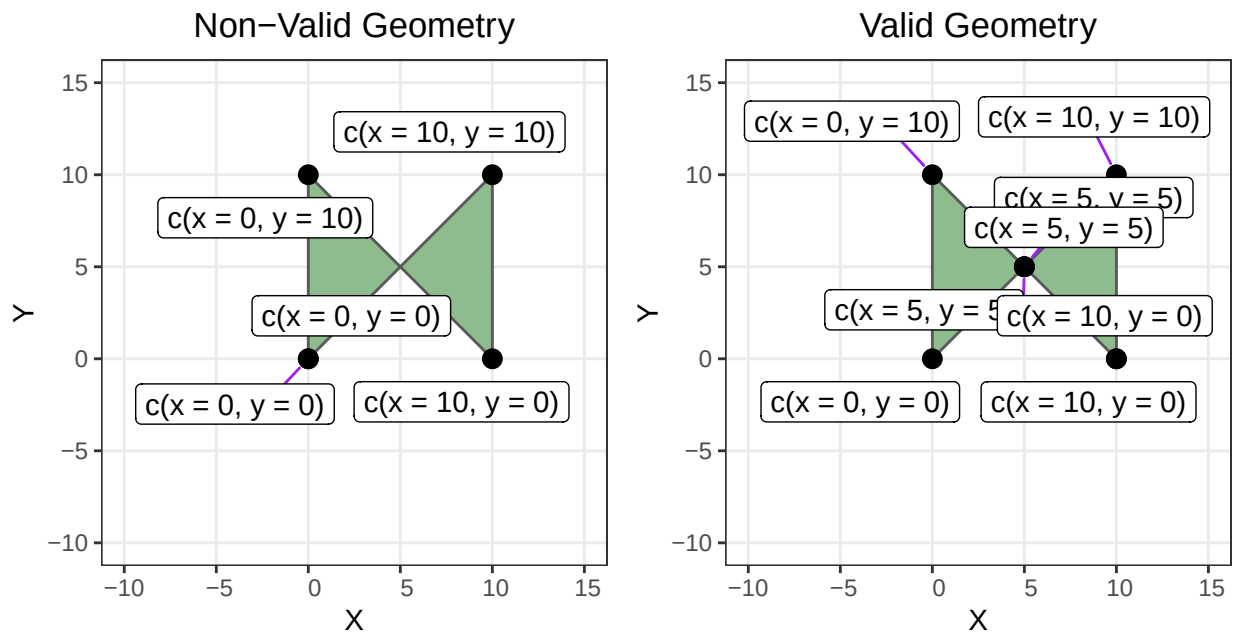
In this case we can retain all geometric features by simply applying the function st_make_valid()

```
p2 <- st_make_valid(p1)
```

We can plot both the old and new features side by side, and label the coordinates, to see how SF made our geometry valid.

```
If we call class(p1), we can see that our original simple feature collection (sfc) contained: sfc_POLYG
```

```
If we call class(p2), we can see that our valid simple feature collection (sfc) contains: sfc_MULTIPOLY
```



As you see from our initial non-valid polygon, a second was created which resolves the geometry problem. Now our feature is represented by two polygons. From our earlier lecture please recall that the start and end points composing the line and polygon simple feature geometries are always the same coordinate pair. Hence you can see that the coordinates of contention, x = 5 & y = 5, have become the new start and end points of our second polygon.

6 Area Calculations

SF is readily capable of calculating the area of polygon features.

```
st_area(chi_neighb)
```

```
## Units: [m^2]
## [1]  4498293.6   200563.5  3016684.2   972375.8 11596322.7
```

These data are always returned in units of meters, regardless of the coordinate system specifications. However, different Coordinate Systems will give different results

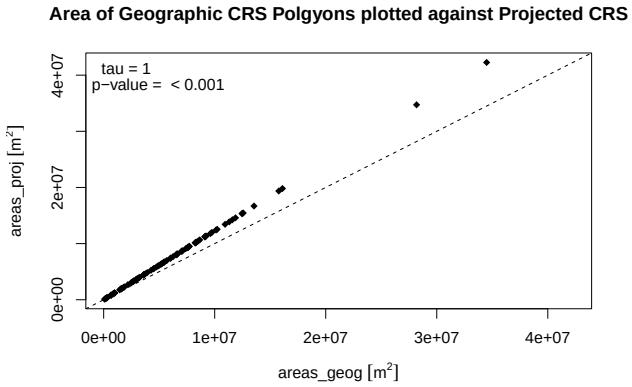
```
integer(0)
```

```
81852.82 [m^2]
```

```
42261063 [m^2]
```

Table 1: Size of the Largest and Smallest Neighborhoods in Chicago

Smallest		Largest	
Neighborhood	Area (m^2)	Neighborhood	Area (m^2)
Magnificent Mile	81.85282	O'Hare	34493.02
Greektown	153.11705	South Deering	28181.48
Printers Row	200.56353	Englewood	16103.87
Millenium Park	257.43359	Austin	15773.36
Boystown	312.21139	Hegewisch	13540.40




```
integer(0)
```

When we use the 'which' logical test, we see that all areas in the results from both sets of area calculations are different. When we plot the values we see that both sets have nearly perfect correlation.

But neither of these values are as accurate as they could be. We want to use an Equal Area projected coordinate system instead.

Here we use the NAD83 Conus Albers, a favorite of the United States Geological Survey (USGS).

```
equal_areas_proj <- chi_neighb%>%
  st_transform(5070) %>%
# We want to use an equal area projection!
  st_area()

equal_areas_proj <- sort(equal_areas_proj)
```

While metric values are easy to convert by 'hand', we can also convert them using the 'units' package.

```
units(equal_areas_proj) <- units::make_units(km^2)
head(equal_areas_proj)
```

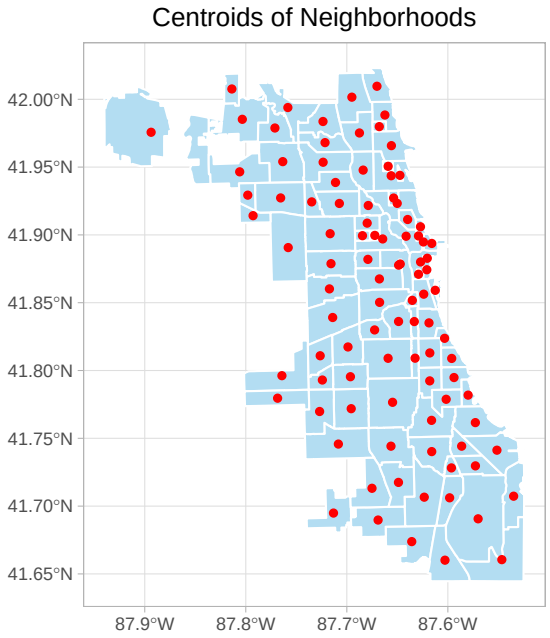
```
Units: [km^2]
[1] 0.08197556 0.15334606 0.20086315 0.25781886 0.31268310 0.32418247
```

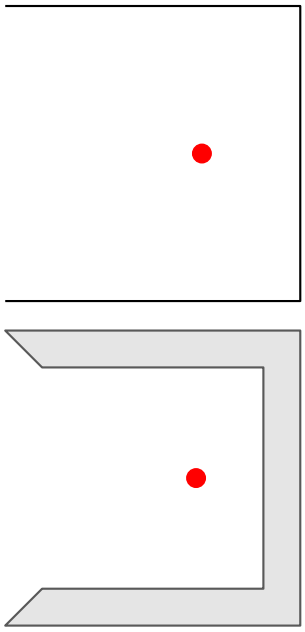
7 Centroids & Point on Surface

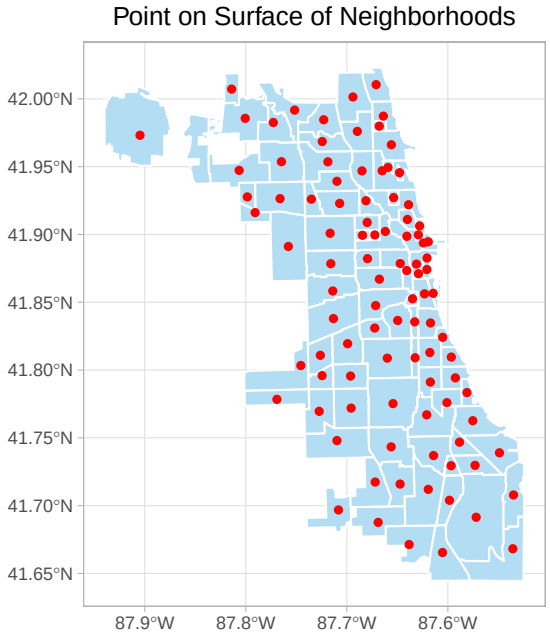
- Centroid the "geometric center of mass of a geometry" of an object.

- If you just need an arbitrary point on the surface of an object Point on Surface will suffice.

```
chi_cent <- st_centroid(chi_neighb)
chi_pos <- st_point_on_surface(chi_neighb)
```








```
## Warning in rm(chi_pos, chi_cent, a, b, c_line, c_polygon, c_line_cen,
## c_poly_cen, : object 'a' not found
```

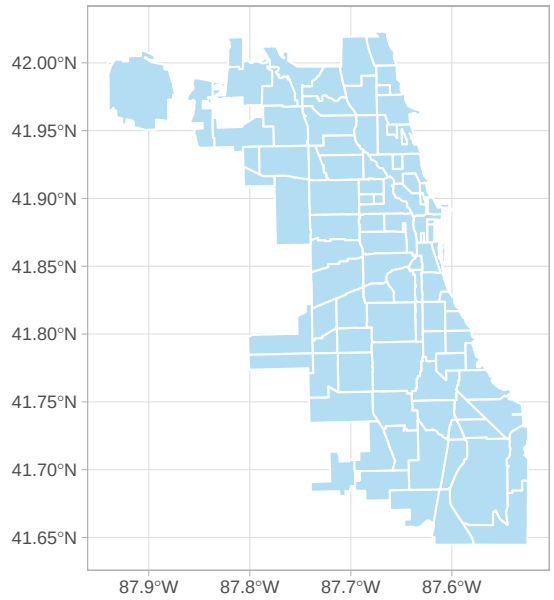
```
## Warning in rm(chi_pos, chi_cent, a, b, c_line, c_polygon, c_line_cen,
## c_poly_cen, : object 'b' not found
```

8 Combine & Union Features

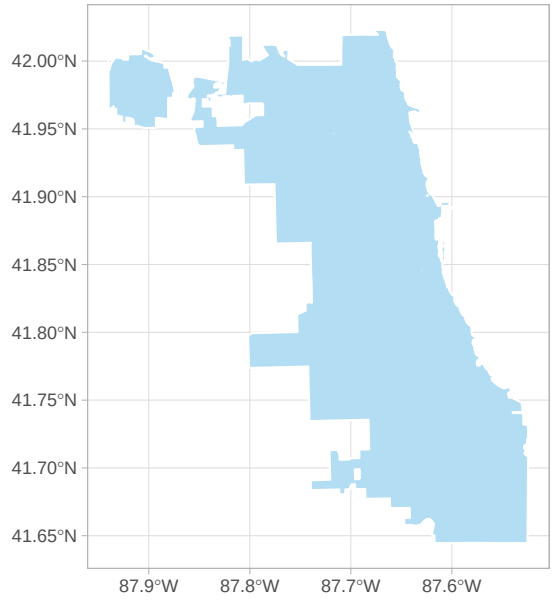
- *Combine* will form a multipolygon
  - Retain all geometries as a multipolygon collection
  - Lose the attributes of each feature
- *Union* will form a multipolygon
  - Retain only outer boundary geometries of features
  - Lose the attributes of each feature

```
chi_combine <- st_combine(chi_neighb)
chi_union <- st_union(chi_neighb)
```

Combine forms a Multipolygon without feature attributes

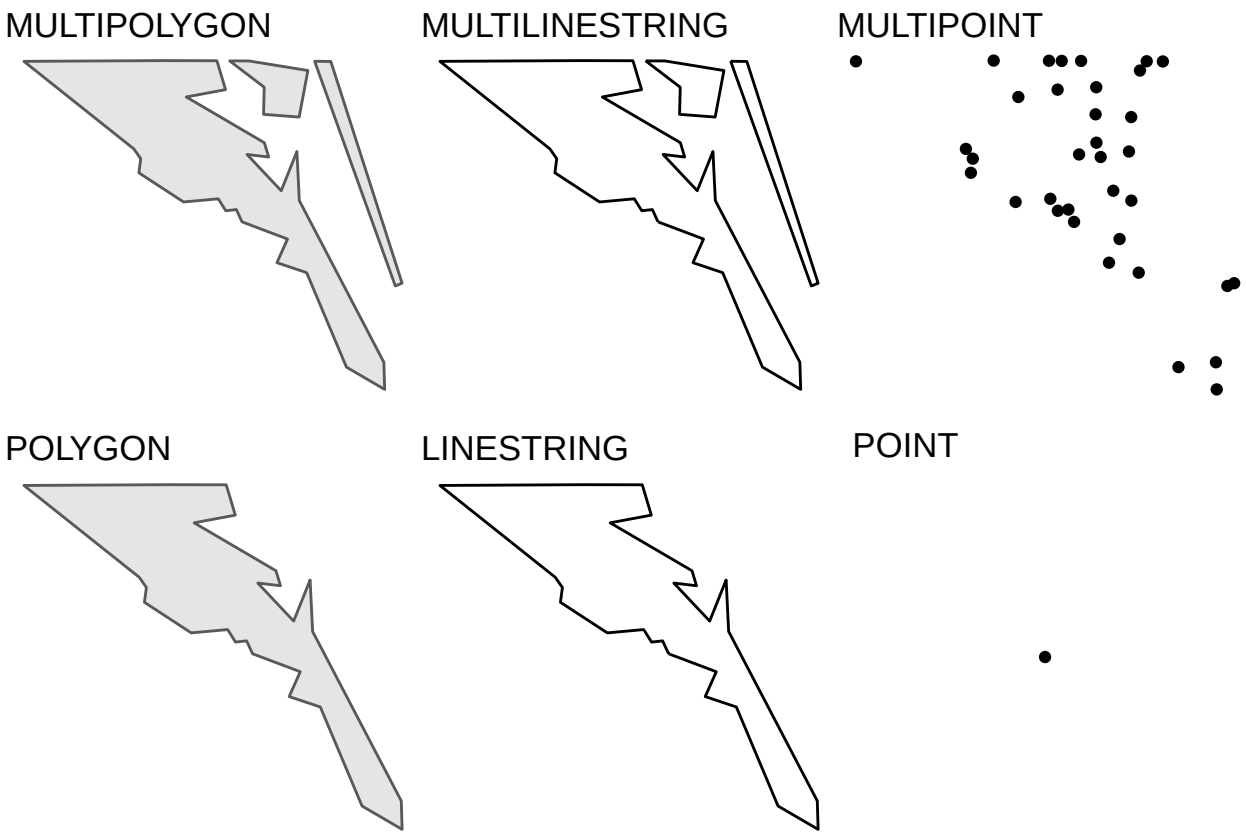


Union also forms a multi*polygon without feature attributes



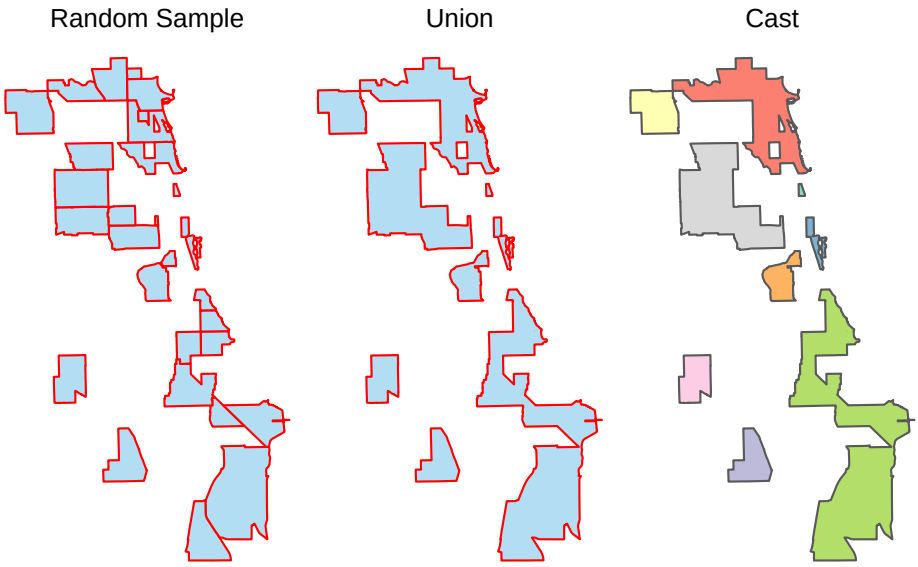
8.1 Cast

- Can be used to extract types of geometries from a multi(polygon, line, point)geometry, or geometry collection.



- Divide a Multipolygon into individual polygons.

```
chi_sub <- chi_neighb[sample(size = 30, 1:nrow(chi_neighb)),]
chi_union <- st_union(chi_sub)
chi_cast <- st_cast(chi_union, to = "POLYGON")
```



9 Buffers & Convex Hulls

- Buffer: Enlarge a feature by a specified distance in X & Y dimensions
- Convex Hull: Encapsulate a feature in X & Y dimensions.

```
chi_buffer_5k <- st_buffer(chi_union, dist = 3000)

chi_ch_by_neigh <- st_convex_hull(chi_neighb)
chi_ch_cit <- st_convex_hull(chi_union)
chi_ch_buf <- st_convex_hull(st_buffer(chi_union, dist = 3000))
```

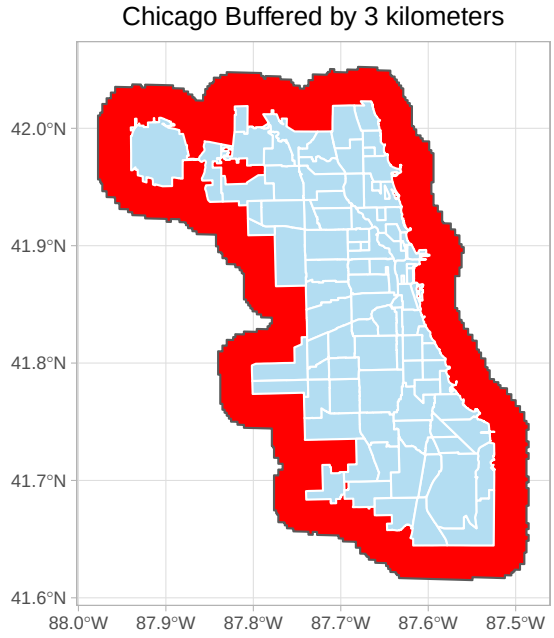


Figure 8: Buffer

10 Measuring Distances

Theory: Remember the earth is a ellipsoid, distances must be calculated along the earths curved surface

- P Point of origin
- Q destination
- u & v : antipodal points, positions across the ellipsoid from each other

```
AM_FM <- st_distance(AM, FM)
AM_SM <- st_distance(AM, SM)
FM_SM <- st_distance(FM, SM)
```

Distance_st	Journey
1144.3	American to Field
334.7	American to Smithsonian
955.4	Field to Smithonian

We can visualize this with some help from Charlie Joey Hadley whom some sf compliant great circle distance code

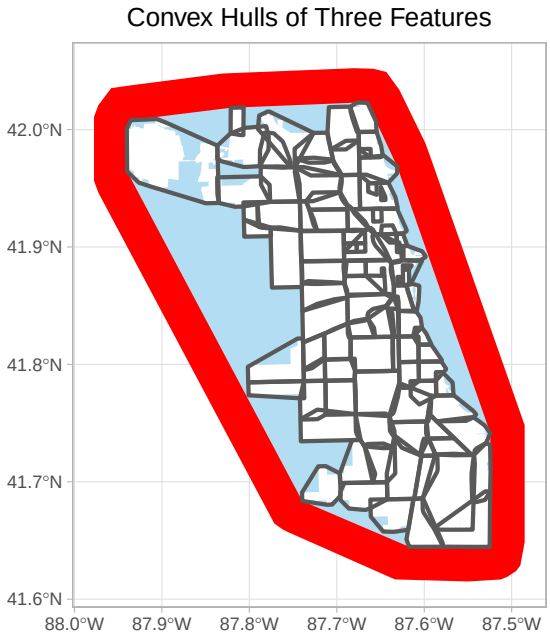


Figure 9: Convex Hull of Chicago Neighborhoods, The City, and a buffer of 3km

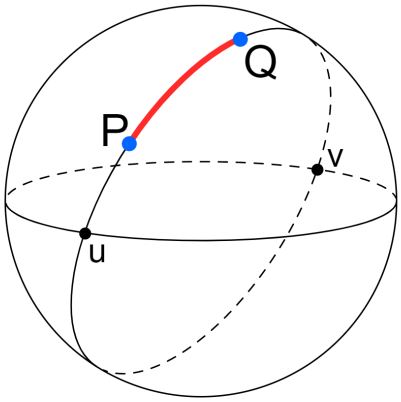
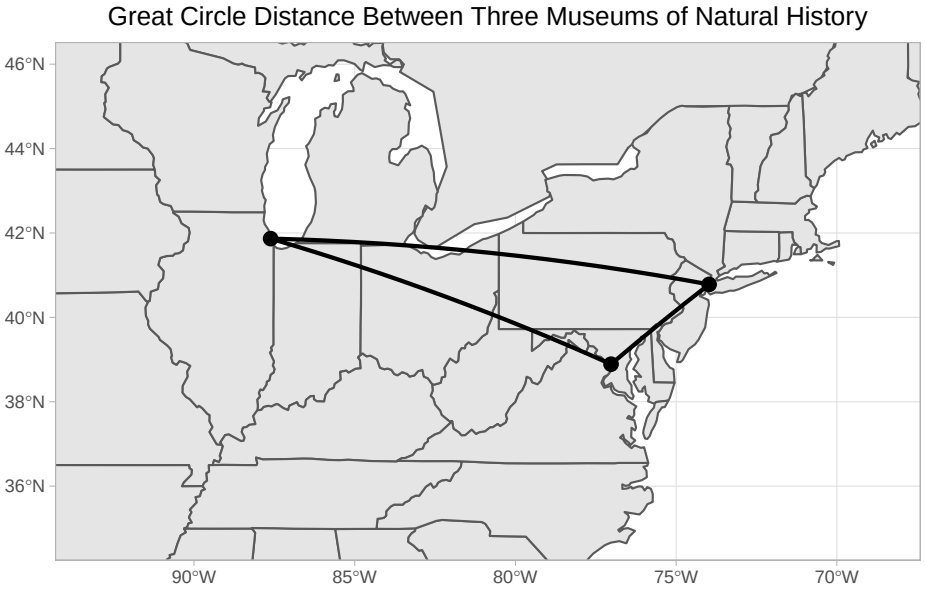


Figure 10: Great Circle Distance, by: ChecCheDaWaff




```
gcr_distance <- st_length(great_circle_routes)
```

In this example we calculate the distance between each of the 98 neighborhoods in Chicago, sum up the total distance of each neighborhood from each other, and divide by the number of neighborhood (98) minus itself (1) to collect and visualize a mean distance between all neighborhoods.

```
neighborhood_centroid_dist <- st_distance(st_centroid(chi_neighb))
```

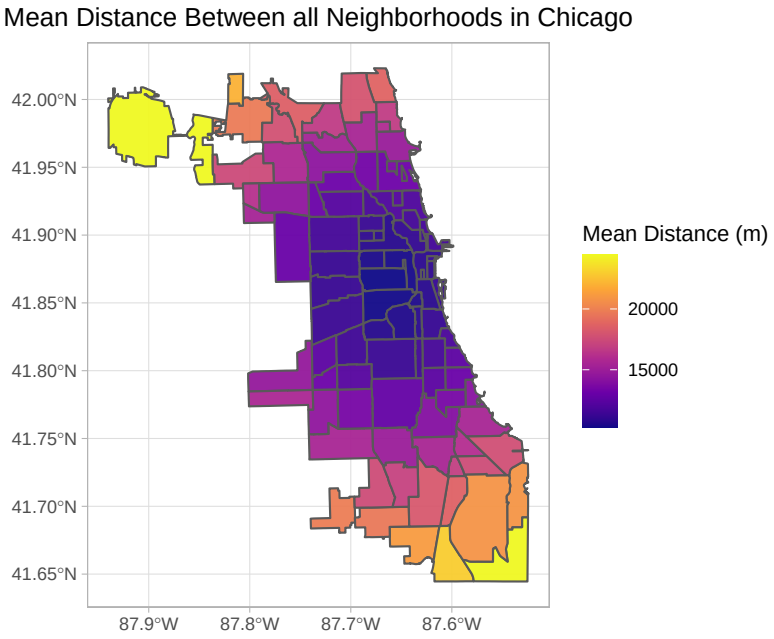


Table 2: The five most Central Chicago Neighborhoods

Neighborhood	Distance
Little Italy, UIC	10291.97
Lower West Side	10350.90
West Loop	10426.27
Greektown	10455.72
United Center	10498.78

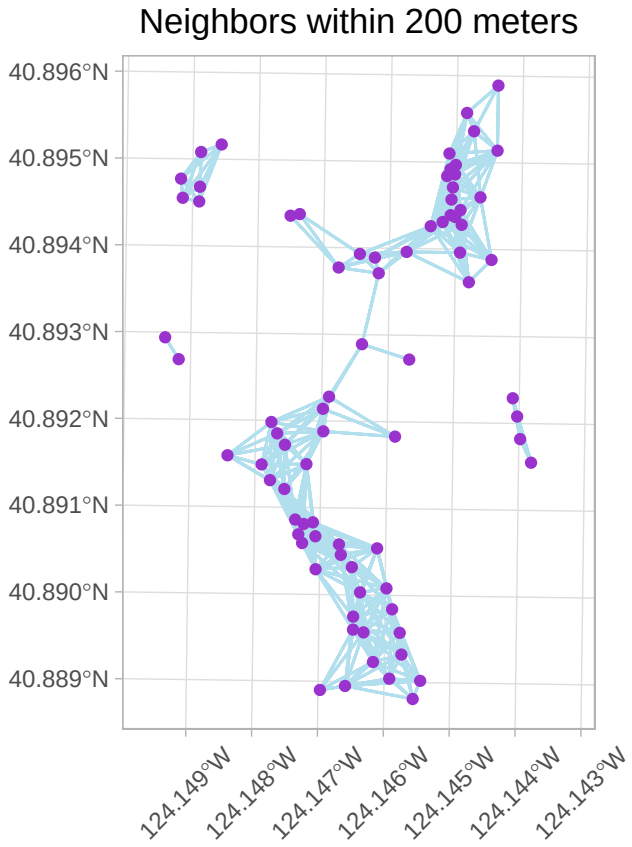
11 An introduction to some types of Spatial Analyses

11.1 Identifying Distance Based Neighbors

Earlier in the lecture we saw that we could develop a list of neighbors from polygon geometries based on adjacency and whether the polygons touched. We can also identify the neighbors of geometries based on the distance between them. Here we will use a set of Points to do this.

Each of these points represents the location of a coastal dune loving plant in Northern California.

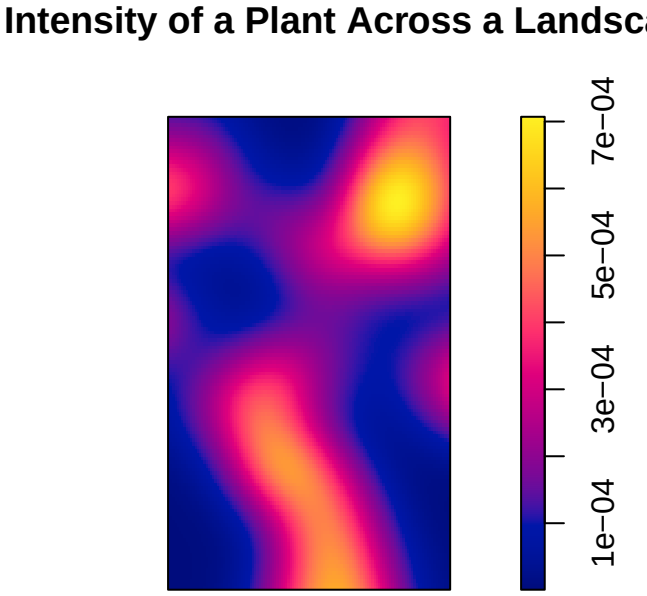
```
Neighbour list object:
Number of regions: 81
Number of nonzero links: 1780
Percentage nonzero weights: 27.13001
Average number of links: 21.97531
2 regions with no links:
13 43
Link number distribution:

 0  4  5  6  7  8  9 15 17 18 20 21 22 23 24 25 26 27 28 29 30 32 33 34
 2  1  1  1  4  3  2  2  2  1  5  3  1  2  9 10  8  9  2  3  3  1  4  2
1 least connected region:
81 with 4 links
2 most connected regions:
26 27 with 34 links
```



From a distance neighbor based linkage analysis, we can for, example determine how many individual plants of this species are located in a close enough proximity that a pollinator may cross pollinate individuals. In the readout from this analysis we see that each plant has the potential to

11.2 Kernel Density Estimation of a Point Pattern

Based on the occurrence's of where individual plants are located, we may be interested in estimating how many plants grow in each unit of area across this landscape. Kernel density estimation may be used to accomplish this. Here this calculation will generate it's predictions on a surface which we may map. Areas which are more orange, have higher *intensity* that is the estimated average density of points per cell.



Note that our field data do not by themselves indicate the abundance or density of individuals in the landscape. While it may appear we are just shading based on the number of points, we actually have developed a new smoothed prediction of intensity.

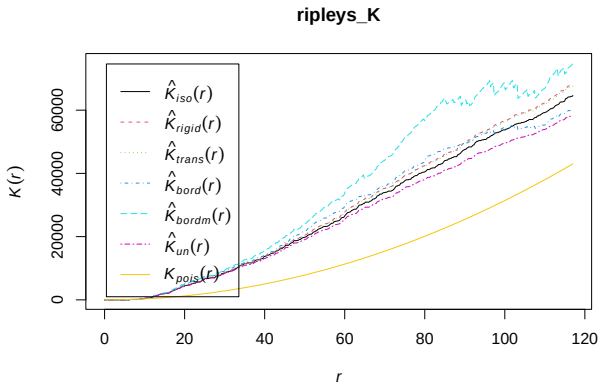
11.3 Cluster a Point Pattern

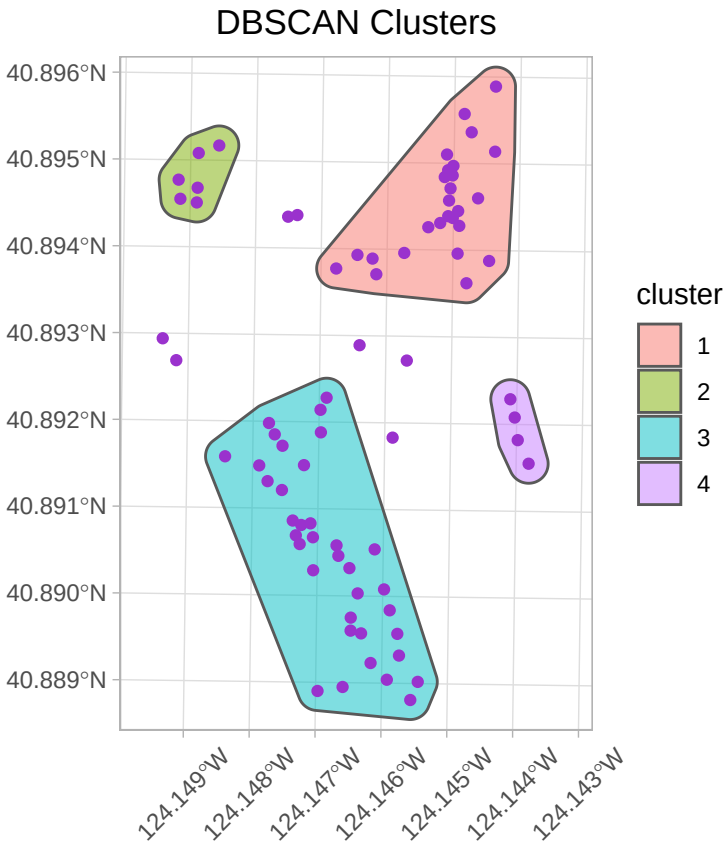
- *D*ensity-*b*ased *s*patial *c*lustering of *a*pplications with *n*oise (DBSCAN).

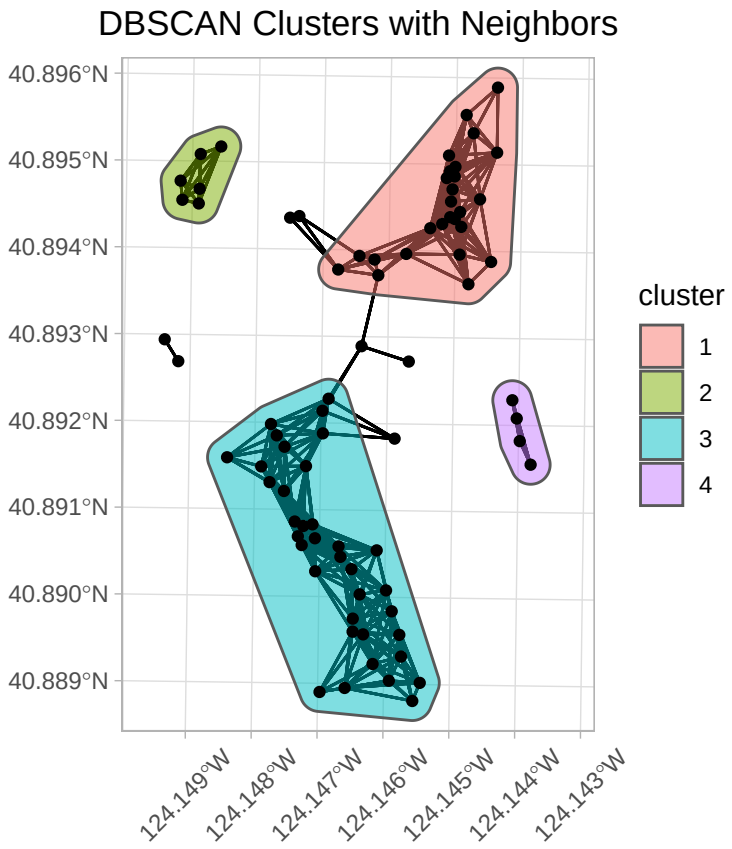
If we combine the ideas of aggregating individuals in space based upon their proximity, with the estimation of the intensity of points, we come out with the ability to aggregate spatial objects into groups based upon nothing more than their relationships in space. Here I showcase the ability to use the DBSCAN algorithm to quickly assign each individual point to a group.

This is a bit of a low hanging fruit for this algorithm, i.e. we could have readily assigned most of these groups accurately ourselves. But do take note of the few connections which are not assigned to groups, and understand the power of this technique to identify tangentially associated localities.

For example, if after calculating Ripley's K, we see that the distribution of our data differ in any way from a random point process, we have evidence that our points are not randomly distributed. When our line is above the curve of a poisson point process their is statistically significant evidence of clustering of points, when the line is beneath the poisson curve, than their is evidence of overdispersion.


ripleys_K
$\hat{K}_{iso}(r)$
$\hat{K}_{rigid}(r)$
$\hat{K}_{trans}(r)$
$\hat{K}_{bord}(r)$
$\hat{K}_{bordm}(r)$
$\hat{K}_{un}(r)$
$K_{pois}(r)$
K(r)
0
20000
40000
60000
0
20
40
60
80
100
120
r


DBSCAN Clusters
40.896°N
40.895°N
40.894°N
40.893°N
40.892°N
40.891°N
40.890°N
40.889°N
124.149°W
124.148°W
124.147°W
124.146°W
124.145°W
124.144°W
124.143°W
cluster
1
2
3
4



12 More cartography

We have already covered a great number of mapping applications. For example we have plotted Rasters using both the plot() function from the Raster Library, and tmap. We have also formed grids of rasters using tmap, and we have added vector features on top of rasters. Throughout the last two scripts we have also prepared multi panel plot's of vector data.

```
shhh <- suppressPackageStartupMessages
shhh(library(spData))
shhh(library(rcartocolor))
shhh(library(ggspatial))
shhh(library(ggrepel))

data(world)
data("us_states", package = "spData")
north_carolina <-  read_sf(system.file("shape/nc.shp", package = "sf")) %>%
  mutate(SIDS_RATE = (SID74/BIR74) * 100)

NC_places <- tigris::places("North Carolina", progress_bar = FALSE)
NC_places <- NC_places %>%
  filter(str_detect(NAMELSAD, "city")) %>% # too many!
  filter(str_detect(NAMELSAD, "Asheville|Raleigh|Charlotte|Greensboro")) %>%
  ## oops there was a Name columnn....
  dplyr::select(NAME) %>%
  st_centroid() %>%
    mutate(longitude = unlist(map(.$geometry,1)),
```

```
           latitude = unlist(map(.$geometry,2)))

nc_bb <- st_transform(north_carolina, 32017) %>%
  st_buffer(150000) %>%
  st_bbox()
nc_bb <-  nc_bb %>%
  st_as_sfc() %>%
  st_transform(crs = 4326)

nc_bb1 <- st_bbox(nc_bb)
world <- world %>% st_crop(nc_bb)
```

12.0.1 Chloropeth

As R is primarily used for statistics and numerical analysis, if one is creating a map in here, I presume you are trying to express a property.

```
ggplot() +
  geom_sf(data = north_carolina, aes(fill = SIDS_RATE)) +
  scale_fill_carto_c(palette = "Burg") +
  theme_light() +
  labs(fill="Birth Rate 1974",
       title = "All Births with SIDS 1974") +
  theme(plot.title = element_text(hjust = 0.5))
```

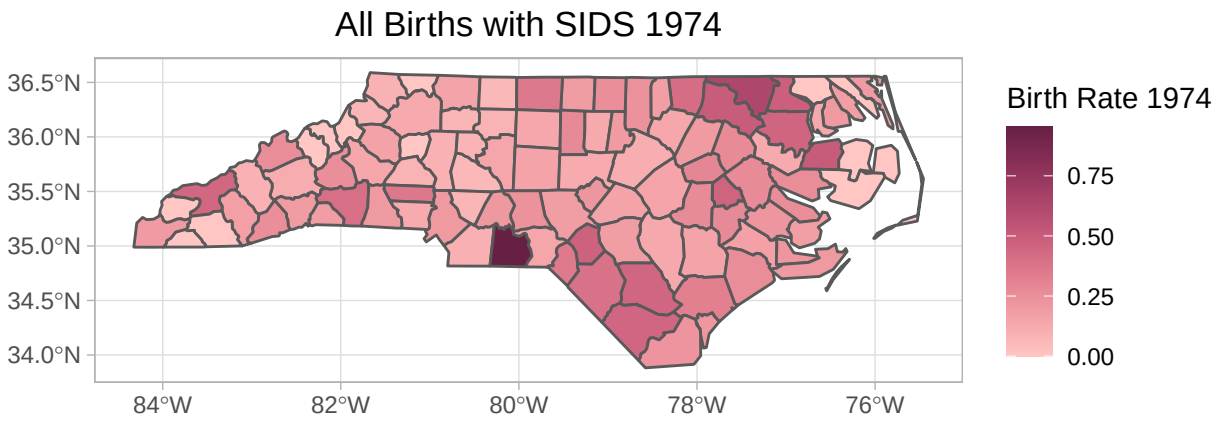


Figure 11: Vector Chloropeth map

12.1 Vector With Cartographic Elements

```
ggplot() +
  geom_sf(data = us_states, fill = "moccasin") +
  geom_sf(data = north_carolina, aes(fill = SIDS_RATE)) +
  geom_sf(data = NC_places) +
  scale_fill_carto_c(palette = "Burg") +
  theme_classic() +
  labs(fill="Proportion", title = "All Births with SIDS 1974") +

  # Define extent of map
  coord_sf(xlim = c(nc_bb1[1],nc_bb1[3]), ylim = c(nc_bb1[2],nc_bb1[4])) +

  # Add the Ocean to the map, add labels for cities.
  theme(plot.title = element_text(hjust = 0.5),
        panel.background = element_rect(fill = "royalblue1")) +
  geom_text_repel(data = NC_places, aes(x = longitude, y = latitude, label = NAME),
                  nudge_x = c(-0.5, -1.0, -0.5, 0.5),
                  nudge_y = c(-0.5,  0.5,  1.0, 1.3)) +

  # Add Scale bar and North Arrow
  annotation_scale(location = "bl", width_hint = 0.2) +
  annotation_north_arrow(location = "bl", which_north = "true",
        pad_x = unit(0.25, "in"), pad_y = unit(0.25, "in"),
        style = north_arrow_fancy_orienteering)
```

12.1.1 Insets

Inset maps are commonly used to specify the locality of a study area. For example, if it would be beneficial for the audience to understand where North Carolina is located within the US, setting the extent of the finer map on the larger map in an inset pane can help.

```
# Create the map of the USA
geom1 <- ggplot() +
  geom_sf(data = us_states, fill = "white") +
  geom_sf(data = nc_bb, fill = NA, color = "red", size = 1.2) +
  theme_void()

#Create the map of NC
geom2 <- ggplot() +
  geom_sf(data = north_carolina, aes(fill = SIDS_RATE)) +
  scale_fill_carto_c(palette = "Burg") +
  theme_void() +
  labs(fill="Proportion", title = "All Births with SIDS 1974")+
  theme(legend.position = c(0.25, 0.1),
        plot.title = element_text(hjust = 0.7),
        legend.direction = "horizontal",
        legend.key.width = unit(10, "mm"))

cowplot::ggdraw() +
  cowplot::draw_plot(geom2) +
  cowplot::draw_plot(geom1, x = 0.04, y = 0.7, width = 0.3, height = 0.3)
```

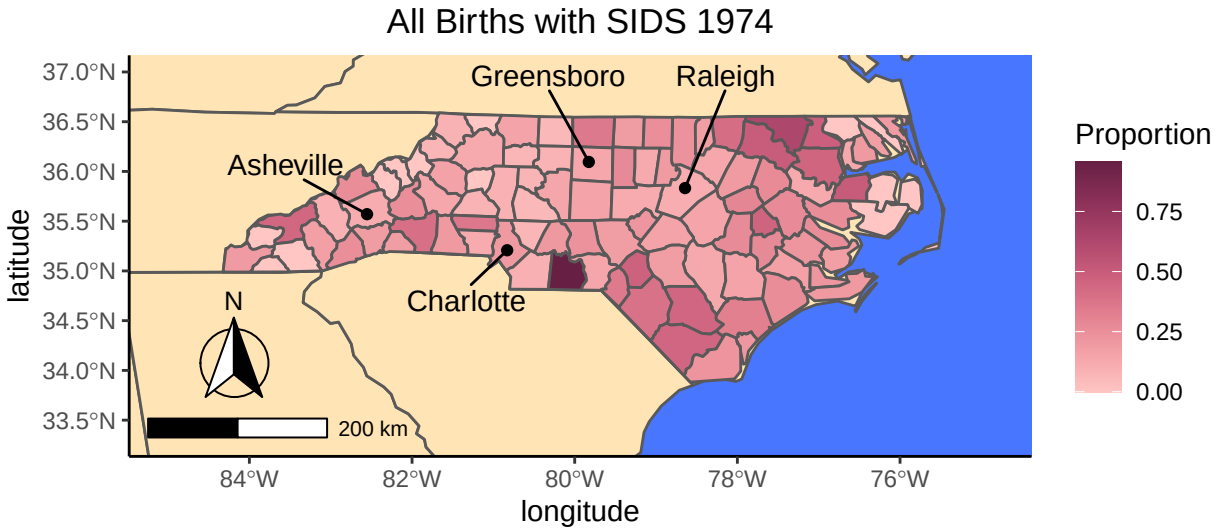


Figure 12: Chloropeth Map with Cartographic Elements

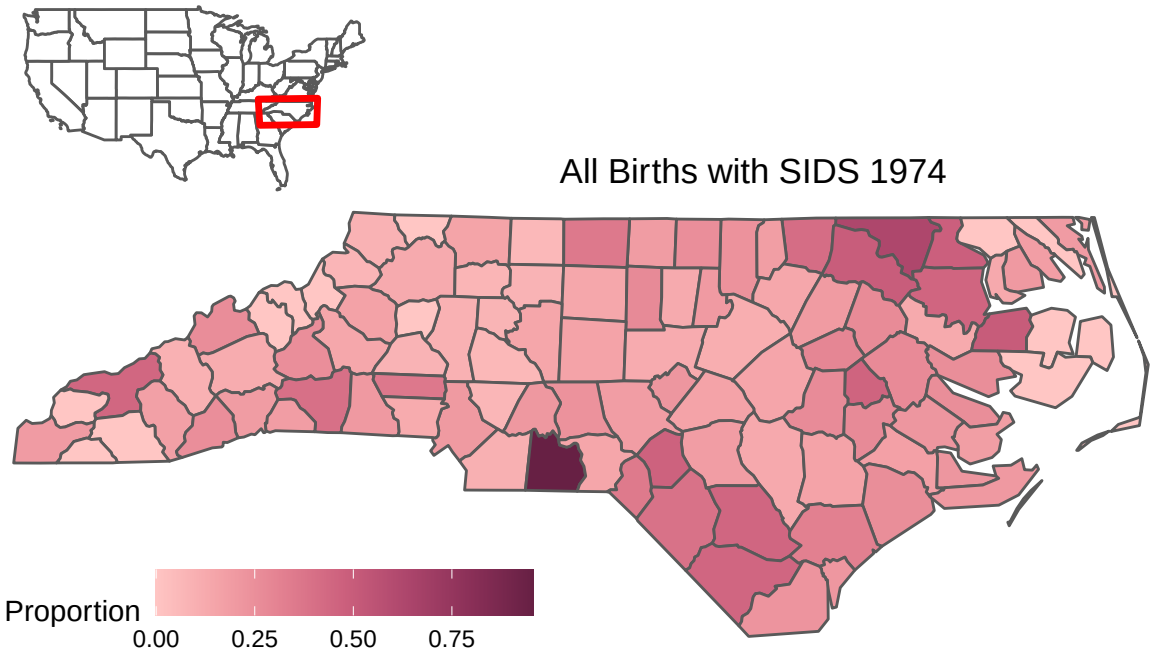


Figure 13: Chloropeth Map with an Inset

12.1.2 Backgrounds

We can also add imagery to the background of a map.

```
site_bbox <- north_carolina %>%
  st_buffer(250) %>%
  st_transform(4326) %>%
  st_bbox(north_carolina)
names(site_bbox) <- c('left', 'bottom', 'right', 'top')

basemap <- ggmap::get_stamenmap(site_bbox, zoom = 8, maptype = "terrain")

ggmap::ggmap(basemap) +
  geom_sf(data = north_carolina,
          alpha = 0.8,
          inherit.aes = FALSE, aes(fill = SIDS_RATE)) +
  coord_sf(crs = st_crs(4326))  +
  scale_fill_carto_c(palette = "Burg") +
  theme_void() +
  labs(fill="Proportion", title = "All Births with SIDS 1974")+
  theme(legend.position = c(0.235, 0.1),
        legend.background = element_rect(fill="cornsilk2", size=.5),
        plot.title = element_text(hjust = 0.5),
        legend.direction = "horizontal",
        legend.key.width = unit(10, "mm"))
```

13 Wrapping up the Spatial Data Science Module:

Future Bonus SDS Office Hours Wednesday Night at 5:00 - 6:00 in F-413.

Notes on this Lab My lecture notes are in the R script I used to generate all of the novel figures for this presentation. This script is much longer and more complex than Lecture 1's script. Likewise this presentation is an .HTML file and can be launched from your computer (it was rendered directly from R using the script).

14 Works Cited

https://cran.r-project.org/web/packages/gstat/vignettes/gstat.pdf Accessed 1.21.2021

https://chicagoreader.com/news-politics/cityscape-how-the-lakefront-was-won/

Bivand, R. https://cran.r-project.org/web/packages/spdep/vignettes/nb_sf.html Accessed 1.17.2021

Clementini, Eliseo; Di Felice, Paolino; van Oosterom, Peter (1993). "A small set of formal topological relationships suitable for end-user interaction". In Abel, David; Ooi, Beng Chin (eds.). Advances in Spatial Databases: Third International Symposium, SSD '93 Singapore, June 23–25, 1993 Proceedings. Lecture Notes in Computer Science. 692/1993. Springer. pp. 277–295. doi:10.1007/3-540-56869-7_16. ISBN 978-3-540-56869-8. Accessed 1.16.2021

Egenhofer, M.J.; Herring, J.R. (1990). "A Mathematical Framework for the Definition of Topological Relationships"

Ester, Martin; Kriegel, Hans-Peter; Sander, Jörg; Xu, Xiaowei (1996). Simoudis, Evangelos; Han, Jiawei; Fayyad, Usama M. (eds.). A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press. pp. 226–231. CiteSeerX 10.1.1.121.9220. ISBN 1-57735-004-9.

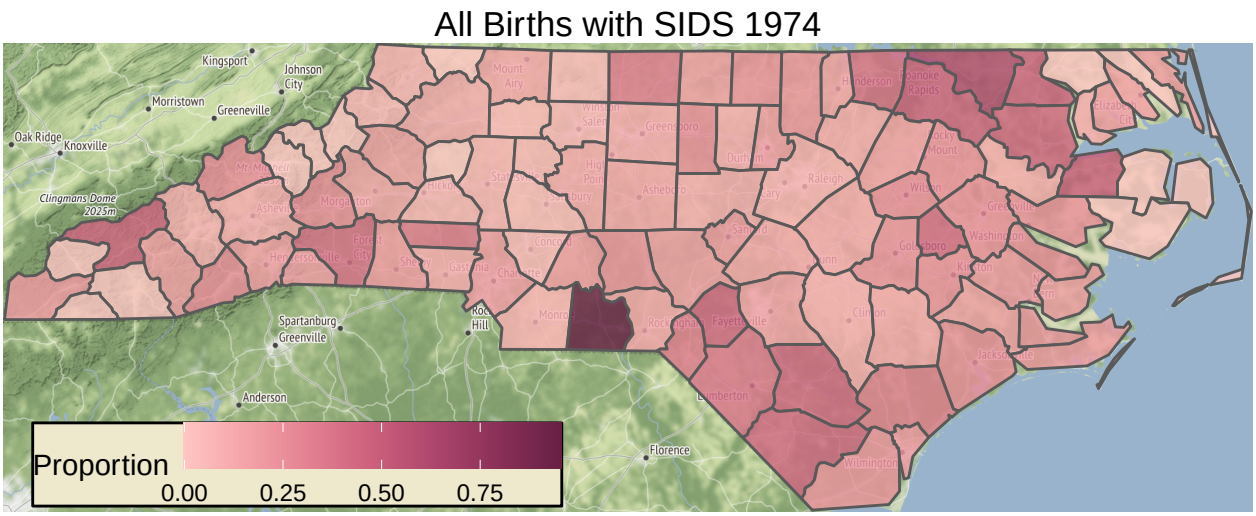


Figure 14: Chloropeth Map with a Basemap

Lee, S., and Tribune Staff Writer https://www.chicagotribune.com/news/ct-xpm-1995-10-19-9510190079-story.html 02.04.2022. Chicago Tribune. 10.19.1995

https://www.fotp.org/lakefront-protection-and-public-trust.html 02.04.2022

Hadley, C.J. https://www.findingyourway.io/blog/2018/02/28/2018-02-28_great-circles-with-sf-and-leaflet/ Accessed 1.16.2021

https://desktop.arcgis.com/en/arcmap/10.3/tools/3d-analyst-toolbox/how-kriging-works.html Accessed 1.17.2021

Tobler W., (1970) "A computer movie simulating urban growth in the Detroit region". Economic Geography, 46(Supplement): 234–240.

https://geocompr.github.io/post/2019/ggplot2-inset-maps/ Accessed 1.22.2022

Gimond, M. https://mgimond.github.io/Spatial/point-pattern-analysis-in-r.html Accessed 1.22.2022

Moreno, M., Basille, M. https://r-spatial.org/r/2018/10/25/ggplot2-sf-2.html Accessed 1.22.2022

Dennett, A. https://rstudio-pubs-static.s3.amazonaws.com/126356_ef7961b3ac164cd080982bc743b9777e.html Accessed 1.21.2022