R Data Science: Spatial Data Science 2

steppe

2/14/2022

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.

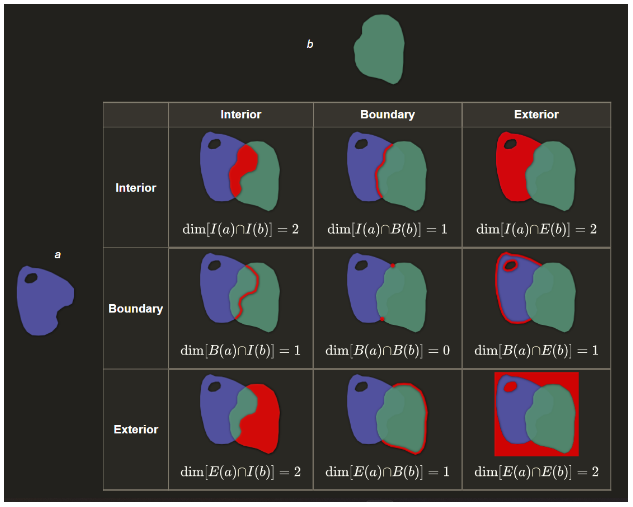

b
a
Interior
Boundary
Exterior
Interior
Boundary
Exterior
$\dim[I(a) \cap I(b)] = 2$
$\dim[I(a) \cap B(b)] = 1$
$\dim[I(a) \cap E(b)] = 2$
$\dim[B(a) \cap I(b)] = 1$
$\dim[B(a) \cap B(b)] = 0$
$\dim[B(a) \cap E(b)] = 1$
$\dim[E(a) \cap I(b)] = 2$
$\dim[E(a) \cap B(b)] = 1$
$\dim[E(a) \cap E(b)] = 2$

Figure 1: DE-9IM, By Krauss

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’)

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’

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_touches’, ‘st_overlaps’, ‘st_covers’, ‘st_contain’, ‘st_equals’)

Within (includes: ‘st_contains’)

CoveredBy (includes: ‘st_equals’)

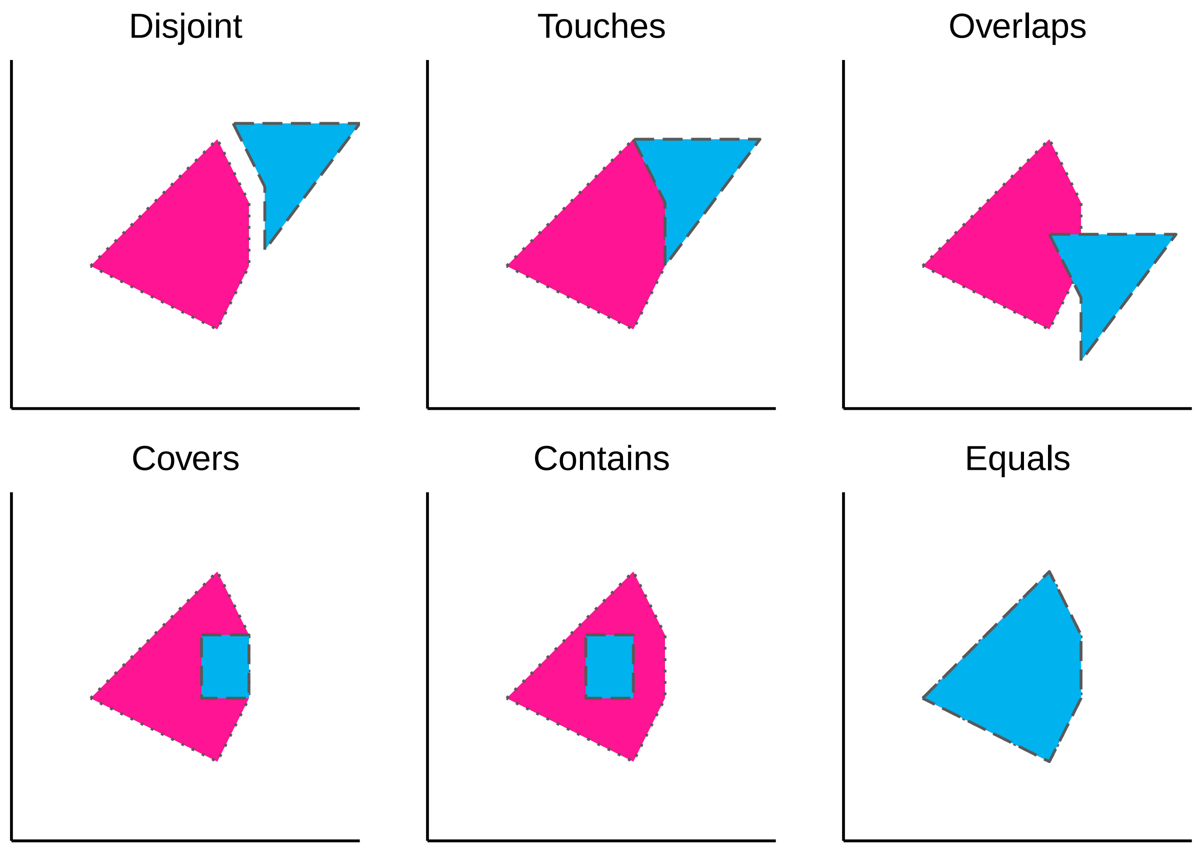


Figure 2: Spatial Predicates after Egenhofer and Herring

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`

- postGIS implementation

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

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

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

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

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

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

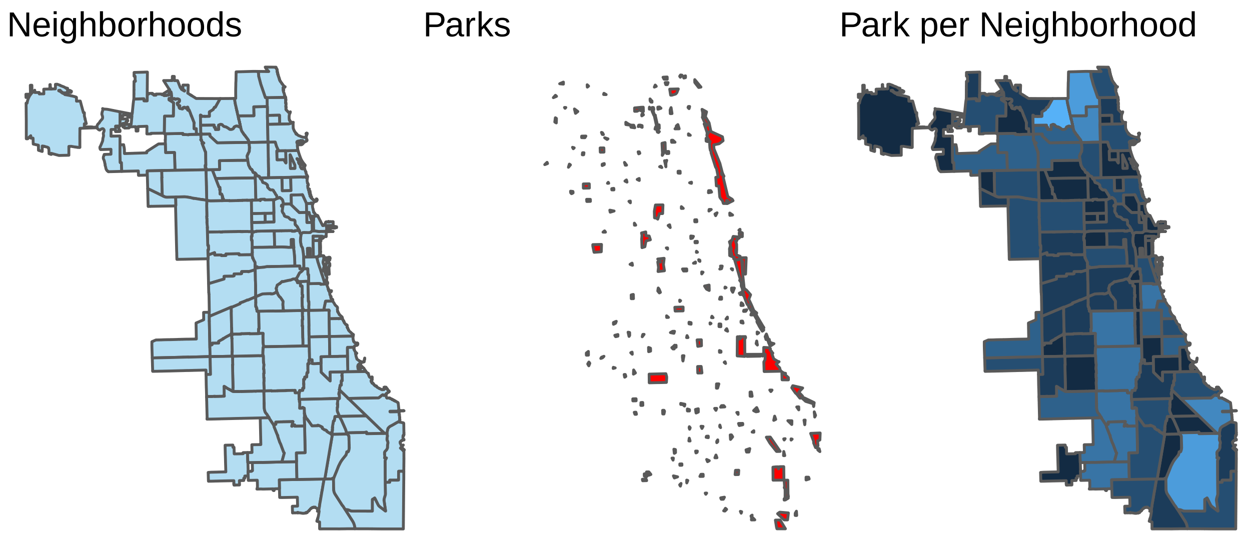
`st_within(B_contains, A, sparse = F): TRUE`

`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)

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



3 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.

3.1 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)
```

3.2 Convert to and from sp objects

- To convert from an sp to an sf object:

```
us_states.sf <- st_as_sf(us_states.sp)
class(us_states.sf)
```

```
[1] "sf"         "data.frame"
```

- to convert from an sf object to an sp object:

```
us_states.sp <- as(us_states.sf, 'Spatial')
class(us_states.sp)
```

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

3.3 Make tabular point data spatial

- Specify columns holding coordinates
- Specify the original crs the data were collected in
- Should the coordinate holding columns be removed?

```
museums.sf <- st_as_sf(museums,
                        coords = c(x = 'longitude', y = 'latitude'),
                        crs = 4326,
                        remove = F
                        )
```

attribute	latitude	longitude	geometry
Field Museum	41.86666	-87.61643	POINT (-87.61643 41.86666)
Science and Industry	41.79006	-87.58227	POINT (-87.58227 41.79006)

3.4 st transform

- Determine the CRS of a simple feature

```
st_crs(museums.sf)
```

```
## Coordinate Reference System:
##   User input: EPSG:4326
##   wkt:
## GEOGCRS["WGS 84",
##     DATUM["World Geodetic System 1984",
##         ELLIPSOID["WGS 84",6378137,298.257223563,
##             LENGTHUNIT["metre",1]]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["geodetic latitude (Lat)",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["geodetic longitude (Lon)",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     USAGE[
##         SCOPE["Horizontal component of 3D system."],
##         AREA["World."],
##         BBOX[-90,-180,90,180]],
##     ID["EPSG",4326]]
```

- Change coordinates from one Coordinate Reference System to another

```
museums.conus_albers <- st_transform(museums.sf, crs = 5070)
```

```
[1] "EPSG:5070"
```

3.5 Remove simple feature list column

```r
museums <- st_drop_geometry(museums.sf)
st_geometry(museums.sf) <- NULL
```

3.6 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.

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

```r
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.

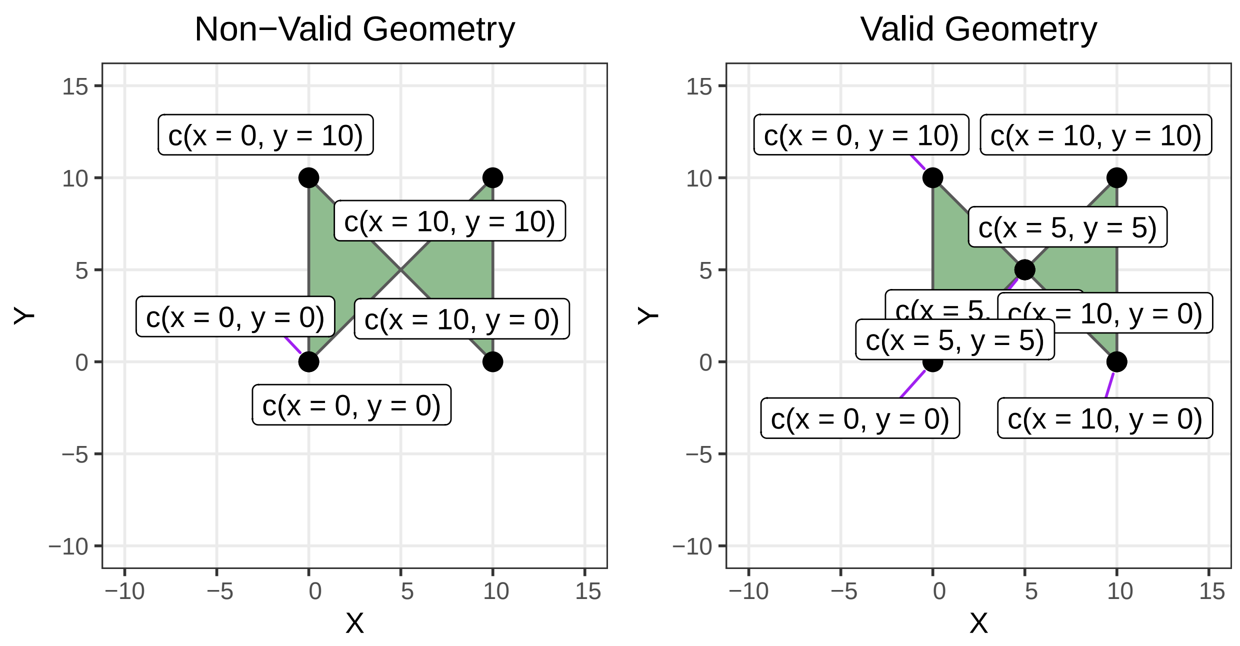
In this case we can retain all geometric features by simply applying the function st_make_valid()

```r
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`



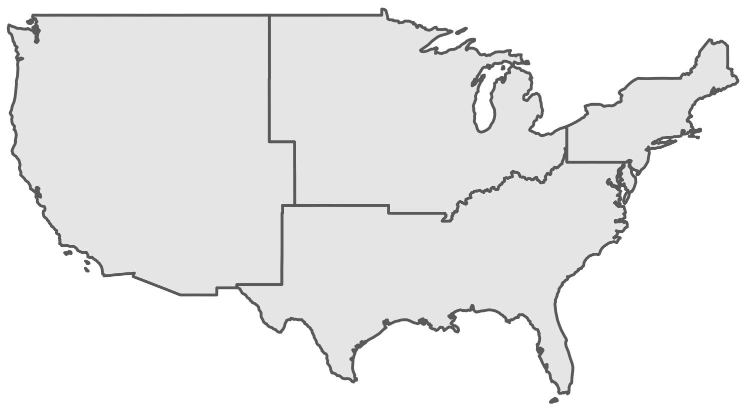
As you see from our initial non-valid polygon, a second was created which resolves the geometry problem. Now our feature is represented by two polygons. From our earlier lecture please recall that the start and end points composing the line and polygon simple feature geometries are always the same coordinate 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.

3.7 Aggregate Simple Features via 'group_by' & 'summarize'

- Can be used to combine the geometries of features

NAME	REGION	geometry
Alabama	South	MULTIPOLYGON (((-88.20006 3...
Arizona	West	MULTIPOLYGON (((-114.7196 3...
Colorado	West	MULTIPOLYGON (((-109.0501 4...
Connecticut	Norteast	MULTIPOLYGON (((-73.48731 4...
Florida	South	MULTIPOLYGON (((-81.81169 2...
Georgia	South	MULTIPOLYGON (((-85.60516 3...

```r
regions <- us_states %>%
  group_by(REGION) %>%
  summarize(geometry = st_union(geometry))
```



REGION	geometry
Norteast	MULTIPOLYGON (((-70.11867 4...
Midwest	MULTIPOLYGON (((-85.48703 4...
South	MULTIPOLYGON (((-81.68524 2...
West	MULTIPOLYGON (((-118.4887 3...

4 Area Calculations

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

```
st_area(chi_neighb)
```

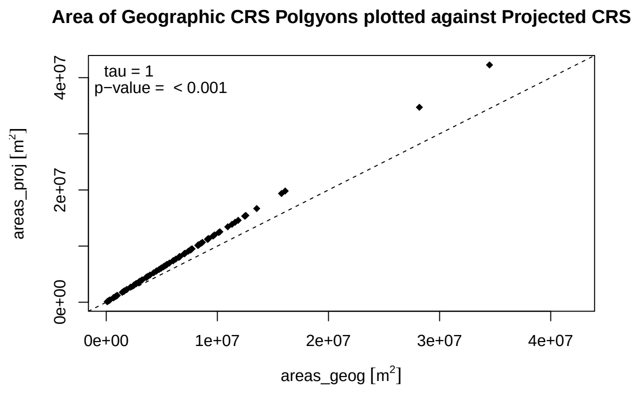
```
Units: [m^2]
[1]  4498293.6   200563.5  3016684.2   972375.8 11596322.7
```

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)
```

Table 4: Size of the Largest and Smallest Neighborhoods in Chicago

Smallest		Largest	
Neighborhood	Area (m^2)	Neighborhood	Area (m^2)
Magnificent Mile	81975.56	O'Hare	34545390
Greektown	153346.06	South Deering	28222390
Printers Row	200863.15	Englewood	16127577
Millenium Park	257818.86	Austin	15796991
Boystown	312683.10	Hegewisch	13559961



```
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).

```r
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.

```r
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
```

4.1 Length calculations

- Calculate the length of a linestring

```
st_length(NU_campuses)
```

```
18578.52 [m]
```

4.2 Boundary calculations

- 'convert' the exterior of a polygon to a linestring (more on this later…)
- apply st_length
- Edzar makes you do spatial problem solving **quite often**

```
rogers_park <- chi_neighb %>%
  filter(pri_neigh == 'Rogers Park')

rogers_park <- st_cast(rogers_park, to = 'MULTILINESTRING')
rogers_park <- st_cast(rogers_park, to = 'LINESTRING')
```

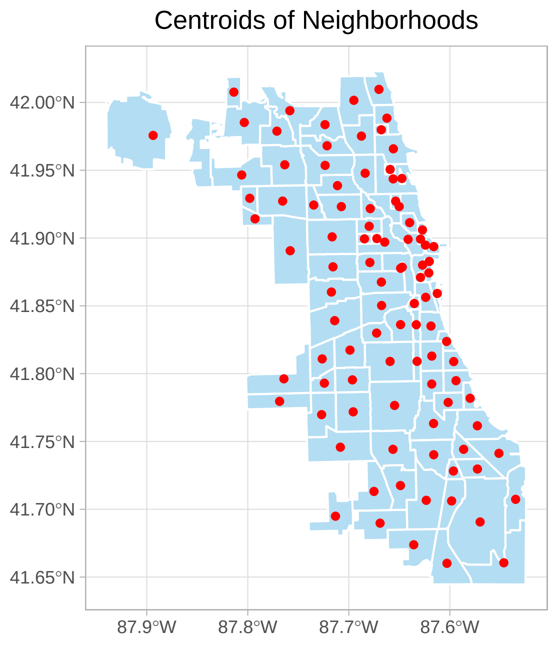
```
st_length(rogers_park)
```

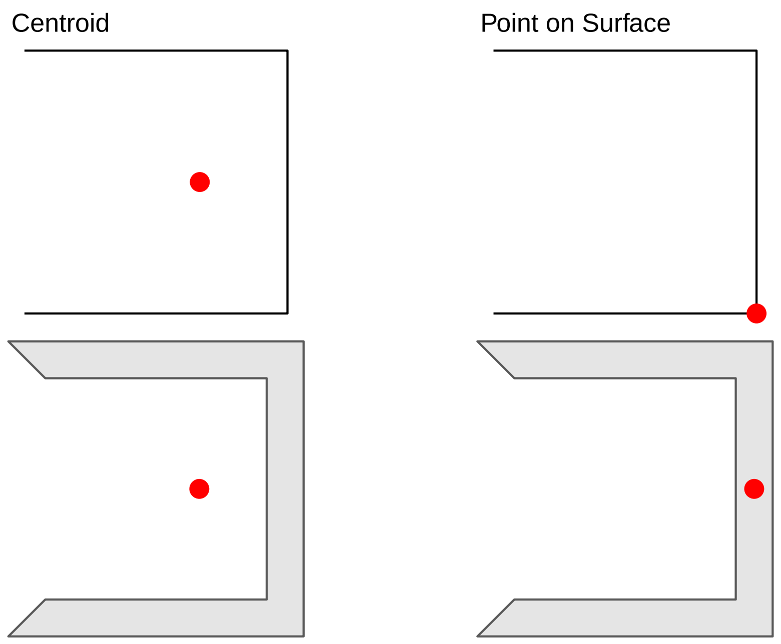
```
10370.89 [m]
```

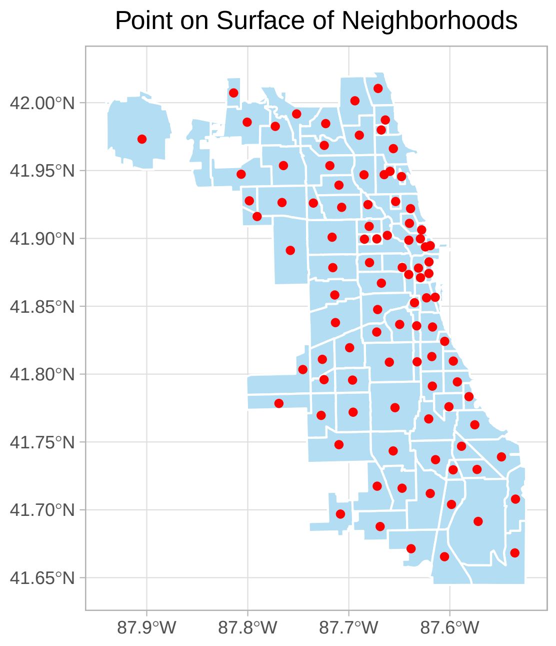
5 Centroids & Point on Surface

- 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)
```





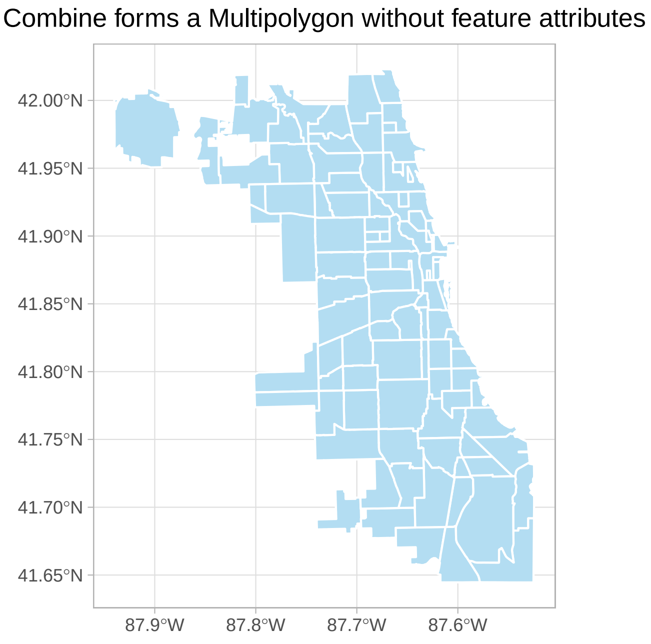


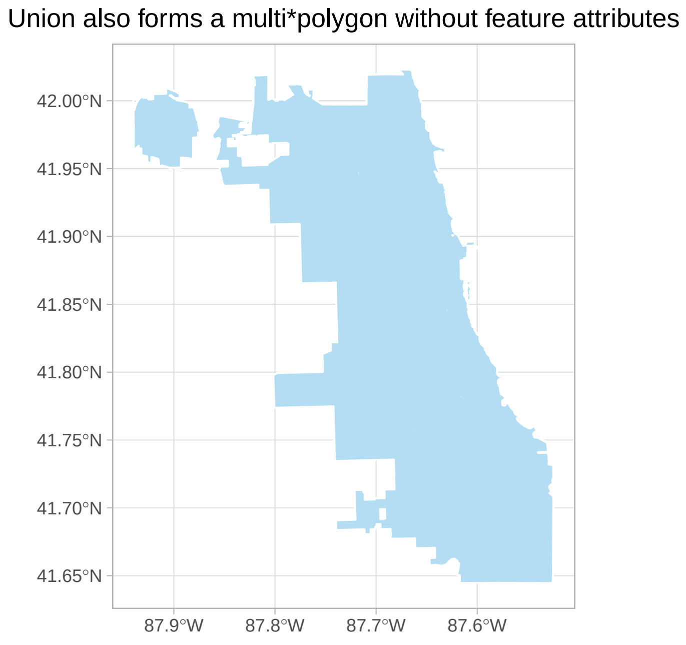
```
## Warning in rm(chi_pos, chi_cent, a, b, c, d, c_line, c_polygon, c_line_cen, :
## object 'c_line_pos' not found
```

6 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

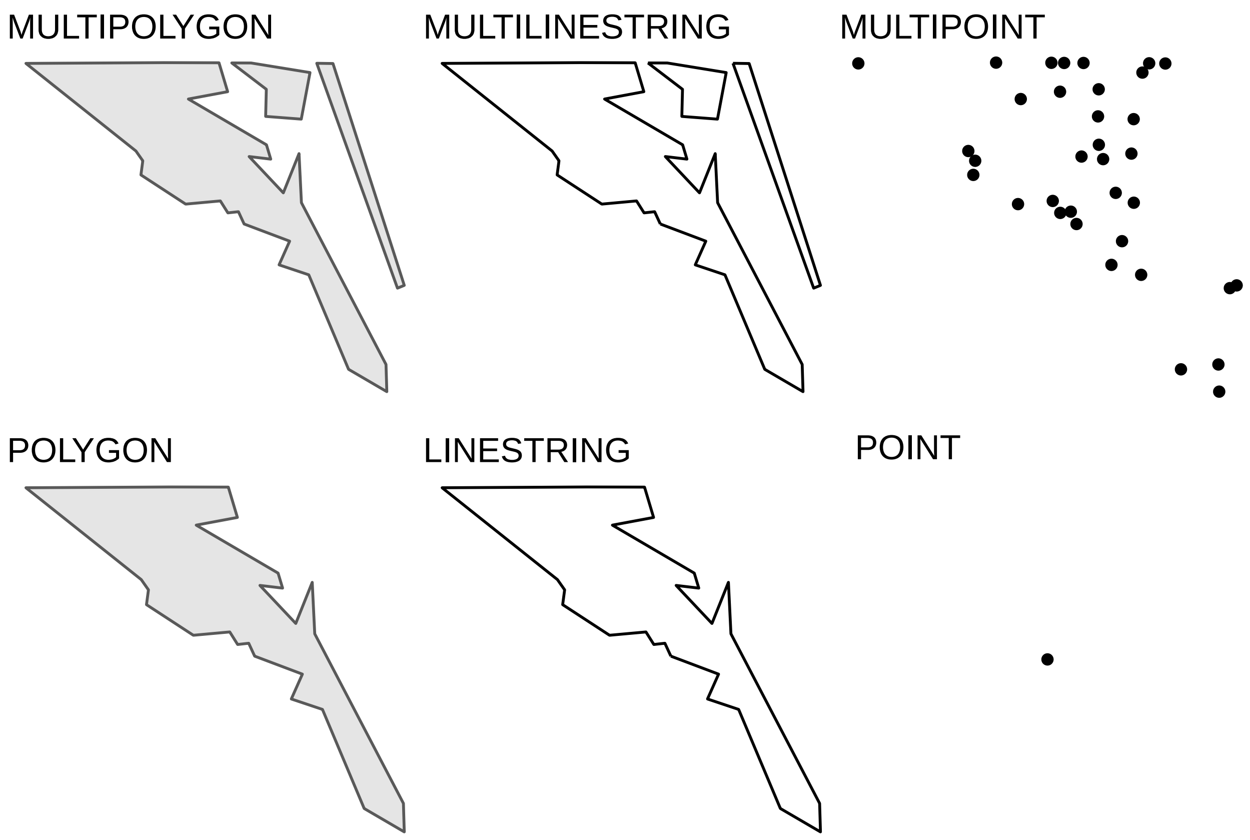
```
chi_combine <- st_combine(chi_neighb)
chi_union <- st_union(chi_neighb)
```





6.1 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")
```

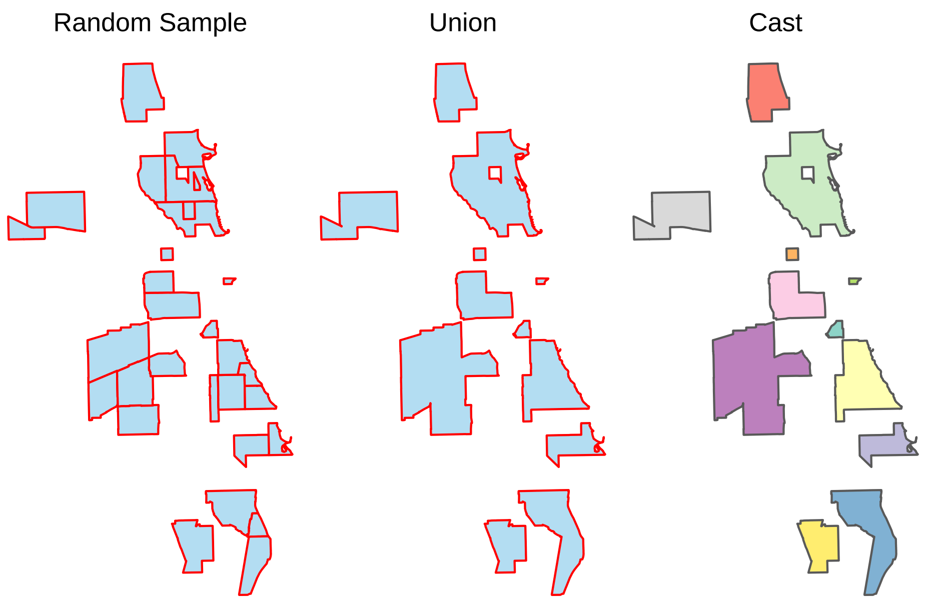


Table 5: Original data

Neighborhood	geometry
Uptown	MULTIPOLYGON (((-87.67398 4...
Fuller Park	MULTIPOLYGON (((-87.6288 41...
United Center	MULTIPOLYGON (((-87.66707 4...
Chatham	MULTIPOLYGON (((-87.59535 4...
Grand Boulevard	MULTIPOLYGON (((-87.60671 4...
Chinatown	MULTIPOLYGON (((-87.63022 4...

Table 6: Unioned Data

geometry
MULTIPOLYGON (((-87.63023 4...

7 Buffers & Convex Hulls

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

```r
chi_buffer_3k <- st_buffer(chi_union, dist = 3000)
chi_union_sp <- as(chi_union, 'Spatial')

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))
```

8 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

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

Table 7: Data Cast to Polygons

geometry
POLYGON ((-87.63023 41.8577...
POLYGON ((-87.59215 41.8169...
POLYGON ((-87.5863 41.77334...
POLYGON ((-87.68485 42.0194...
POLYGON ((-87.60157 41.6862...
POLYGON ((-87.67669 41.8959...

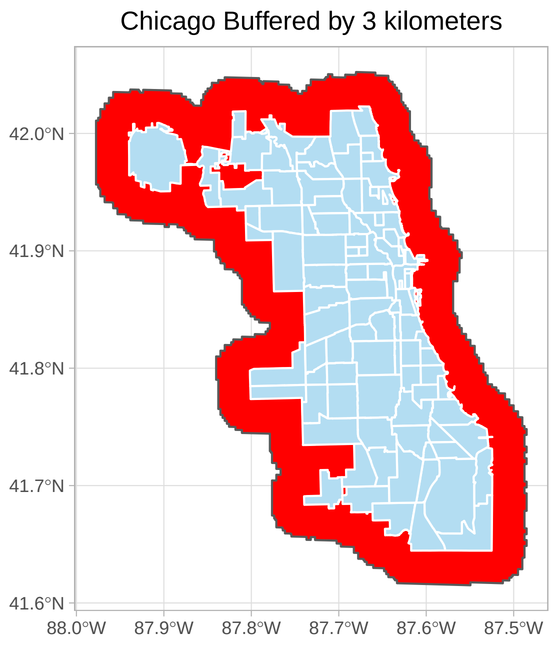


Figure 3: Buffer

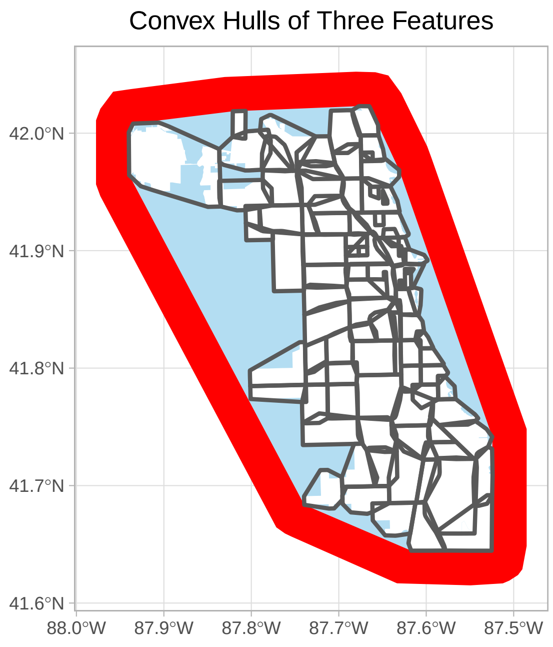


Figure 4: Convex Hull of Chicago Neighborhoods, The City, and a buffer of 3km

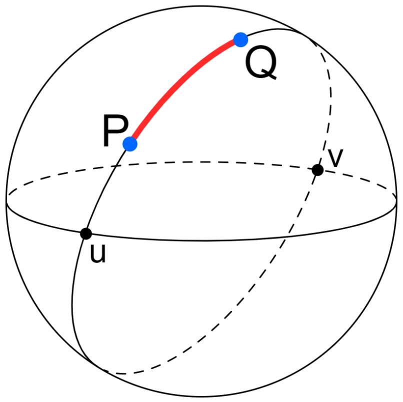
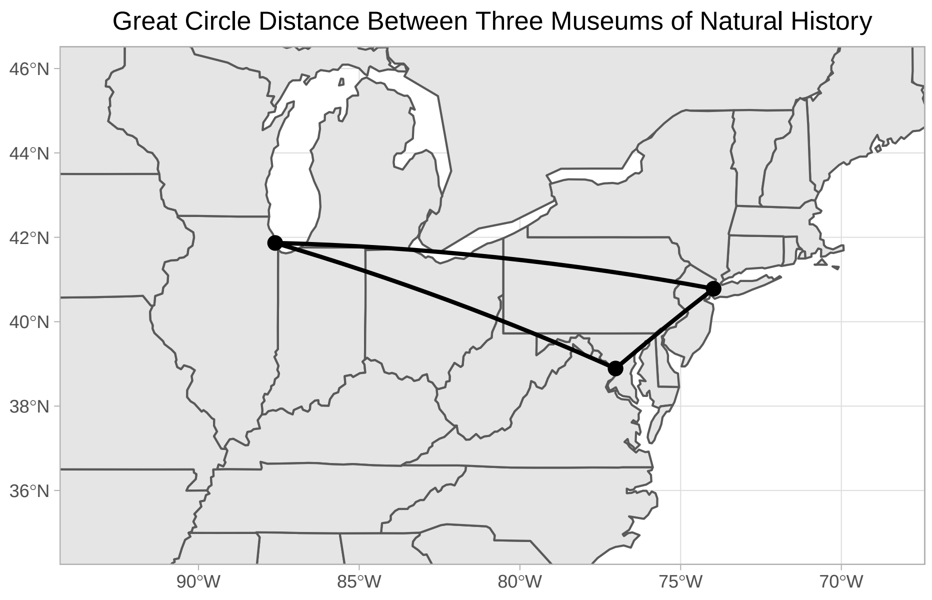


Figure 5: Great Circle Distance, by: ChecCheDaWaff

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


Great Circle Distance Between Three Museums of Natural History
46°N
44°N
42°N
40°N
38°N
36°N
90°W
85°W
80°W
75°W
70°W

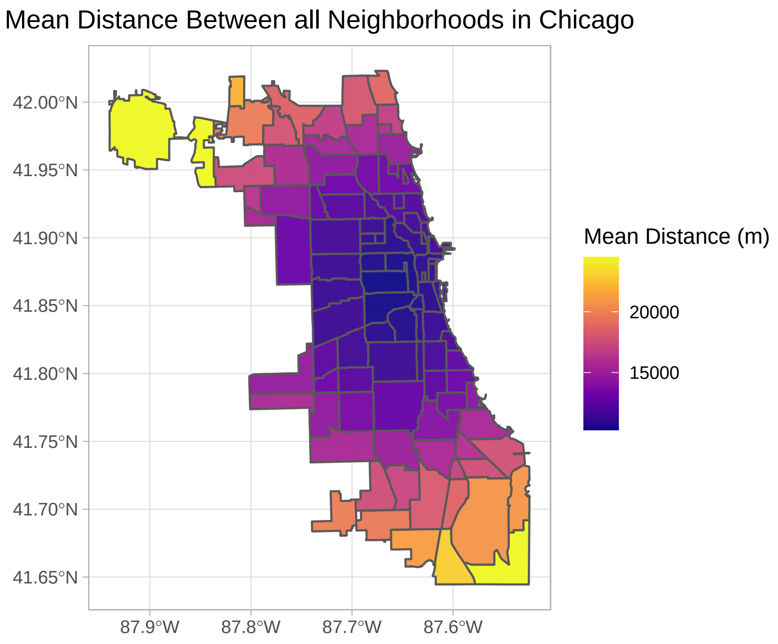
```
gcr_distance <- st_length(great_circle_routes)
```

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.

Table 8: 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

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



9 An introduction to some types of Spatial Analyses

- Two very popular and simple linear models
- **S**patial **A**utoregressive **M**odels (SAR), and **C**onditional **A**utoregressive **M**odels (CAR)
- Both require knowledge of the distance between observations to weigh values

9.1 Identifying Distance Based Neighbors

- One can find neighboring objects via distance thresholds using the spdep package
- these neighbors can be used in many types of linear regression for spatial analyses

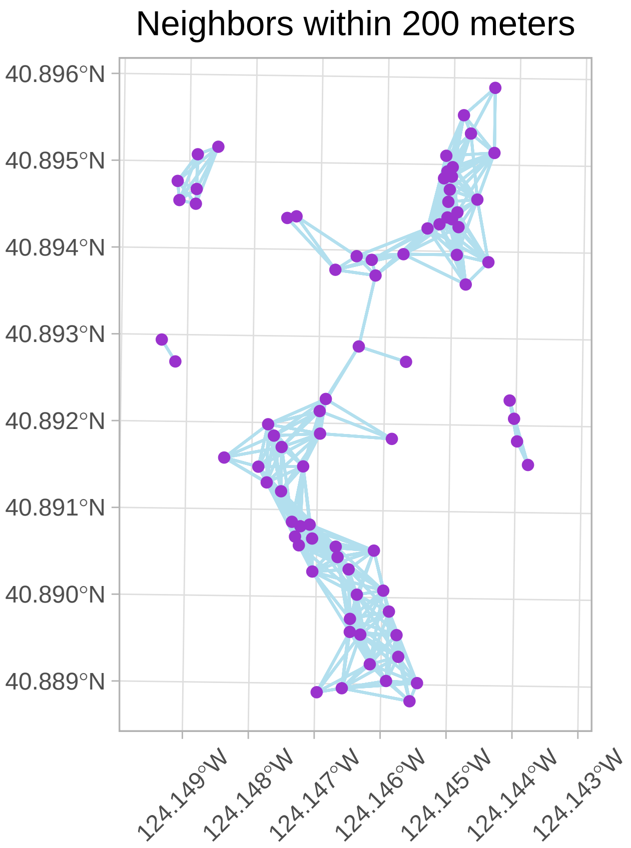
Earlier in the lecture we saw that we could develop a list of neighbors from polygon geometries based on adjacency and whether the polygons touched. We can also identify the neighbors of geometries based on the distance between them. Here we will use a set of Points to do this.

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
```

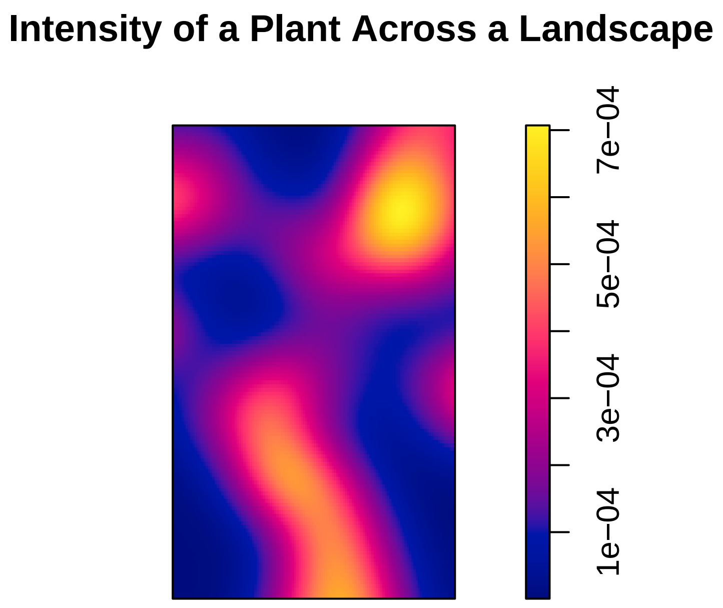


From a distance neighbor based linkage analysis, we can for, example determine how many individual plants of this species are located in a close enough proximity that a pollinator may cross pollinate individuals. In the readout from this analysis we see that each plant has the potential to

9.2 Kernel Density Estimation of a Point Pattern

- estimate the *intensity* in a sense, the density of a process in space
- more widely used for building models than estimating densities via projections

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.

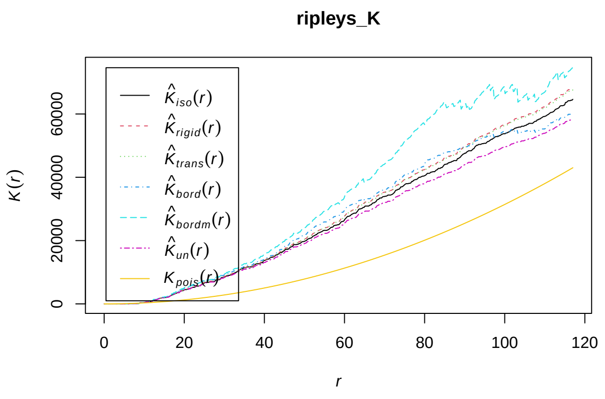
9.3 Cluster a Point Pattern

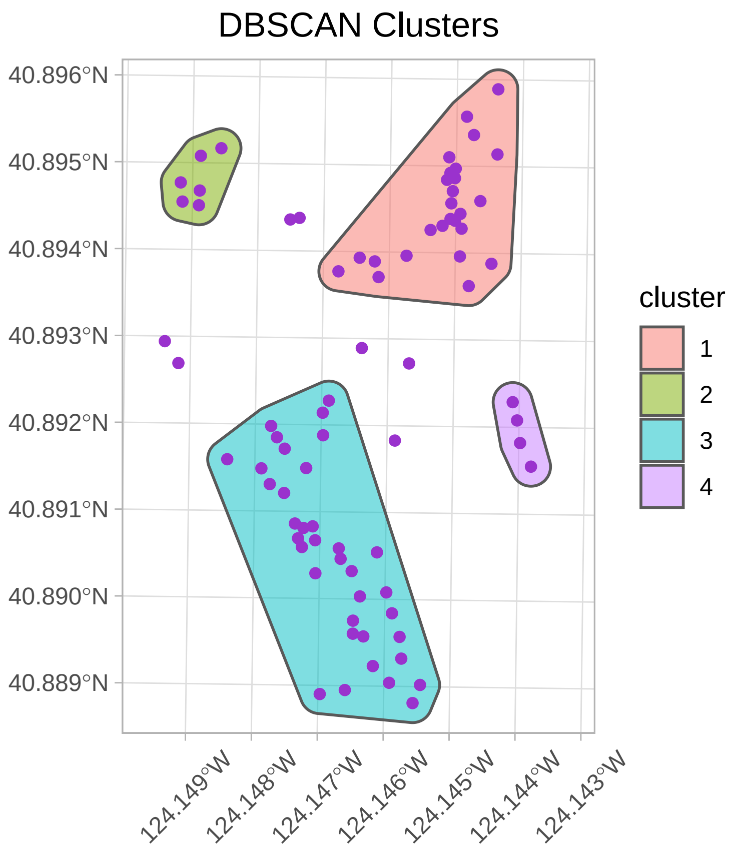
- **D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise (DBSCAN).
- Classification algorithm for grouping objects in space
- Developed in the mid 90's, still used and highly cited!
- Requires two terms
  - minium points: the number of points required to form a cluster
  - eps: loosely related to distance for which neighbor searches can occur
- Ripley's K
  - Determine whether a point process is not randomly distributed
  - Points can be fit with models of distributions
  - segments of a models fit above the poisson curve have clustered distributions
  - a process which can be worth modeling.

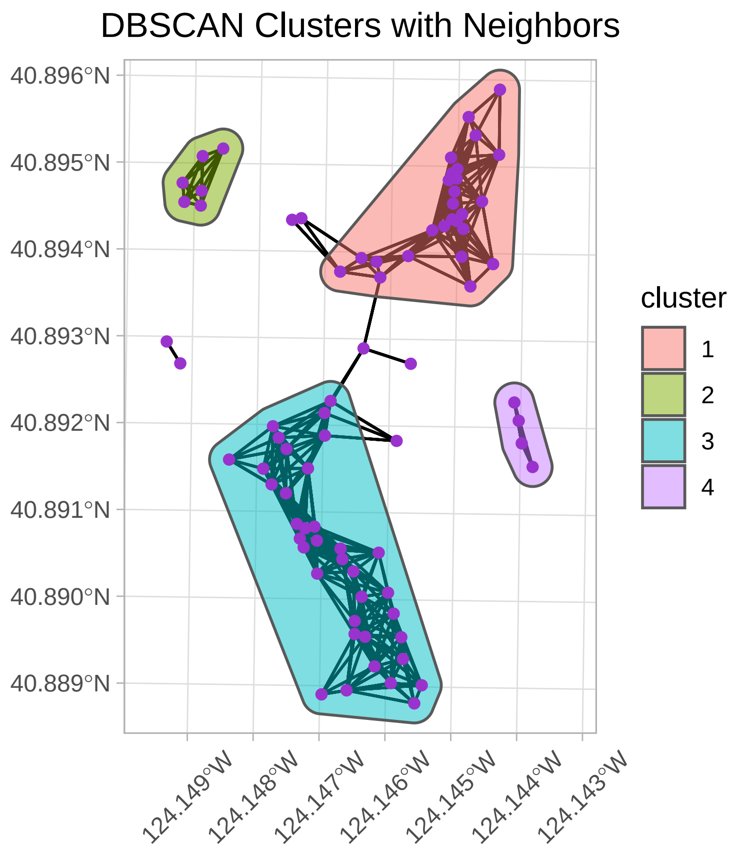
If we combine the ideas of aggregating individuals in space based upon their proximity, with the estimation of the intensity of points, we come out with the ability to aggregate spatial objects into groups based upon nothing more than their relationships in space. Here I showcase the ability to use the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







10 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).

11 Works Cited

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

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.

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

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

11.1 Packages cited:

```r
c("raster", "sp", "sf", "tidyverse", "terra", 'spData', 'spdep', 'gstat', 'spatstat', 'fields' ) %>%
  map(citation) %>%
  print(style = "text", na.print = '')
```

```
[[1]]
Hijmans R (2022). _raster: Geographic Data Analysis and Modeling_. R
package version 3.5-15, <URL:
https://CRAN.R-project.org/package=raster>.

[[2]]
Pebesma EJ, Bivand RS (2005). "Classes and methods for spatial data in
R." _R News_, *5*(2), 9-13. <URL:
https://CRAN.R-project.org/doc/Rnews/>.

Bivand RS, Pebesma E, Gomez-Rubio V (2013). _Applied spatial data
analysis with R, Second edition_. Springer, NY. <URL:
https://asdar-book.org/>.

[[3]]
Pebesma E (2018). "Simple Features for R: Standardized Support for
Spatial Vector Data." _The R Journal_, *10*(1), 439-446. doi:
10.32614/RJ-2018-009 (URL: https://doi.org/10.32614/RJ-2018-009), <URL:
https://doi.org/10.32614/RJ-2018-009>.

[[4]]
Wickham H, Averick M, Bryan J, Chang W, McGowan LD, François R,
Grolemund G, Hayes A, Henry L, Hester J, Kuhn M, Pedersen TL, Miller E,
Bache SM, Müller K, Ooms J, Robinson D, Seidel DP, Spinu V, Takahashi
K, Vaughan D, Wilke C, Woo K, Yutani H (2019). "Welcome to the
tidyverse." _Journal of Open Source Software_, *4*(43), 1686. doi:
10.21105/joss.01686 (URL: https://doi.org/10.21105/joss.01686).

[[5]]
Hijmans R (2022). _terra: Spatial Data Analysis_. R package version
1.5-18, <URL: https://rspatial.org/terra/>.

[[6]]
Bivand R, Nowosad J, Lovelace R (2021). _spData: Datasets for Spatial
Analysis_. R package version 2.0.1, <URL:
https://CRAN.R-project.org/package=spData>.

[[7]]
Bivand R, Wong DWS (2018). "Comparing implementations of global and
```

local indicators of spatial association." _TEST_, *27*(3), 716-748.
<URL: https://doi.org/10.1007/s11749-018-0599-x>.

Bivand RS, Pebesma E, Gomez-Rubio V (2013). _Applied spatial data
analysis with R, Second edition_. Springer, NY. <URL:
https://asdar-book.org/>.

[[8]]
Pebesma EJ (2004). "Multivariable geostatistics in S: the gstat
package." _Computers & Geosciences_, *30*, 683-691.

Gräler B, Pebesma E, Heuvelink G (2016). "Spatio-Temporal Interpolation
using gstat." _The R Journal_, *8*, 204-218. <URL:
https://journal.r-project.org/archive/2016/RJ-2016-014/index.html>.

[[9]]
Baddeley A, Rubak E, Turner R (2015). _Spatial Point Patterns:
Methodology and Applications with R_. Chapman and Hall/CRC Press,
London. <URL:
https://www.routledge.com/Spatial-Point-Patterns-Methodology-and-Applications-with-R/Baddeley-Rubak-Tur

Baddeley A, Turner R, Mateu J, Bevan A (2013). "Hybrids of Gibbs Point
Process Models and Their Implementation." _Journal of Statistical
Software_, *55*(11), 1-43. doi: 10.18637/jss.v055.i11 (URL:
https://doi.org/10.18637/jss.v055.i11).

Baddeley A, Turner R (2005). "spatstat: An R Package for Analyzing
Spatial Point Patterns." _Journal of Statistical Software_, *12*(6),
1-42. doi: 10.18637/jss.v012.i06 (URL:
https://doi.org/10.18637/jss.v012.i06).

[[10]]
Douglas Nychka, Reinhard Furrer, John Paige, Stephan Sain (2021).
"fields: Tools for spatial data." R package version 13.3, <URL:
https://github.com/dnychka/fieldsRPackage>.