

Building Spatial Databases based on Location

HES 505 Fall 2022: Session 15

Matt Williamson

Outline for today

- Update on assignments
- Refresher: Building a spatial analysis workflow
- Building a database for an analysis (part 2) based on location

Update on assignments

- Assignment 2 due by 14 Oct
- Self-assessment 2 due 21 Oct
- Resubmits
- Final Project

Objectives

By the end of today you should be able to:

- Create new features based on topological relationships
- Use topological subsetting to reduce features
- Use spatial joins to add attributes based on location

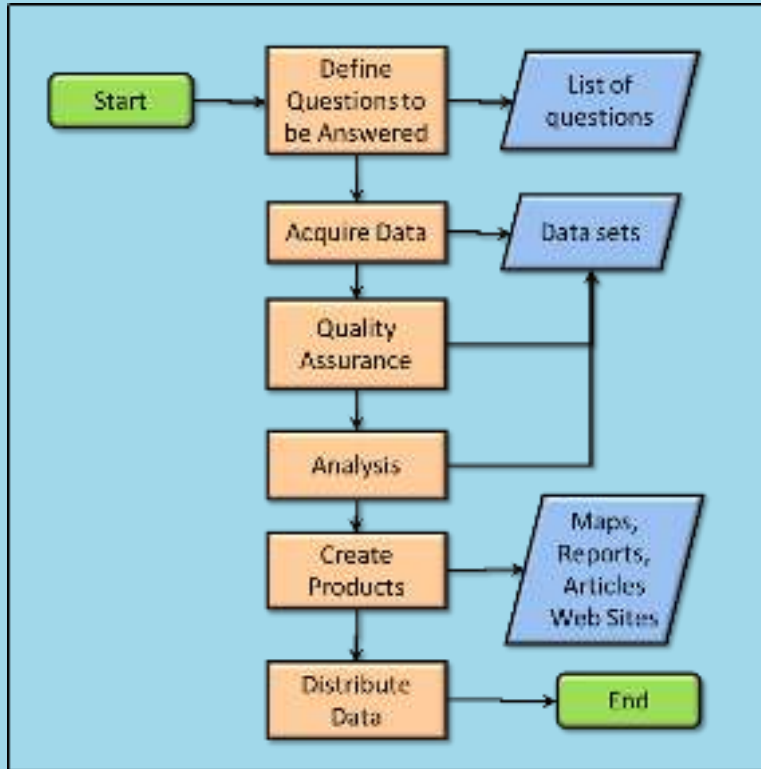
Revisiting Spatial Analysis

What is spatial analysis?

“The process of examining the locations, attributes, and relationships of features in spatial data through overlay and other analytical techniques in order to address a question or gain useful knowledge. Spatial analysis extracts or creates new information from spatial data”.

— ESRI Dictionary

Workflows for spatial analysis



- Align processing with objectives
- Imagining the visualizations and analysis clarifies file formats and variables
- Helps build reproducibility

courtesy of Humboldt State
University

Databases and Attributes

	A	B	C	D	E	F
	AREA	PERIMETER	APN	LANDUSE	LOT SIZE	HECTARES
1	6474154.36275	11146.88972	00100400000000	HFAJAG	8796360.000000	40000
2	7070794.15172	15644.47836	00100400015000	HFAJAG	9969000.000000	40000
3	12991087.28984	18307.82117	00100200000000	HFAJAG	12235172.000000	40000
4	2843043.70203	7886.52182	00100400030000	HFAJAG	2741200.000000	40000
5	102725.06962	5215.30267	00100200090000	WCAACA	187300.000000	40000
6	38655.06531	3719.30211	00100200040000	WCAACA	26255.888889	40000
7	238716.26834	3578.14239	00100200000000	WCAACA	259106.800000	40000
8	1271388.36556	15888.48816	00100200000000	HFAJAG	13069100.000000	40000
9	9530649.13776	11620.31722	00100200000000	HFAJAG	80159157.600000	40000
10	2534654.40519	7726.17855	00100200000000	HFAJAG	2800472.400000	40000
11	2403652.50513	7031.64816	00100200090000	HFAJAG	2080000.900000	40000
12	4080660.54001	2997.10462	00100200000000	WCAACA	4420488.800000	40000
13	385170.08140	13027.90196	00100100450000	WCAACA	432930.000000	40000
14	8702821.66378	13027.90196	00100100150000	WCAACA	8887982.400000	40000
15	1408916.88818	8367.67718	00100100090000	WCAACA	1484100.000000	40000
16	228970.51558	2488.00862	00100100160000	WCAACA	217364.400000	40000
17	1368014.22168	4888.26427	00100100170000	WCAACA	1152488.800000	40000
18	1815128.08861	5071.54836	00100100110000	WCAACA	1504296.000000	40000
19	32486.36388	783.48578	00100100140000	WCAACA	35142.000000	40000
20	885458.08888	3274.84752	00100510010000	ATBDBA	630092.400000	40000
21	3458710.31762	28027.24830	00101000000000	WCAACA	4154392.400000	40000
22	210706.26281	2465.20722	00100500010000	WCAACA	236095.200000	40000
23	798957.83178	11028.88528	00101000000000	WCAACA	20337.800000	40000
24	280178.57538	8774.17883	00100530050000	JAQAAA	235224.000000	40000
25	37128.21791	808.85834	00100100130000	WCAACA	36808.000000	40000
26	198741.88422	3205.72915	00100530080000	ATBDBA	181172.000000	40000

courtesy of [Giscommons](#)

- Attributes: Information that further describes a spatial feature
- Attributes → predictors for analysis
- Last week focus on thematic relations between datasets
 - Shared 'keys' help define linkages between objects
- Sometimes we are interested in attributes that describe location (overlaps, contains, distance)
- Sometimes we want to join based on location rather than thematic connections
 - Must have the same CRS

Calculating New Attributes

Attributes based on geometry and location (**measures**)

- Attributes like area and length can be useful for a number of analyses
 - Estimates of 'effort' in sampling designs
 - Offsets for modeling rates (e.g., Poisson regression)
- Need to assign the result of the function to a column in data frame (e.g., **\$**, **mutate**, and **summarize**)
- Often useful to test before assigning

Estimating area

- **sf** bases area (and length) calculations on the map units of the CRS
- the **units** library allows conversion into a variety of units

```
1 nz.sf <- nz %>%  
2   mutate(area = st_area(nz  
3 head(nz.sf$area, 3)
```

```
1 nz.sf$areakm <- units::set  
2 head(nz.sf$areakm, 3)
```

Estimating Density in Polygons

- Creating new features based on the frequency of occurrence
- Clarifying graphics
- Underlies quadrat sampling for point patterns
- Two steps: count and area

Estimating Density in Polygons

```
1 nz.df <- nz %>%  
2 mutate(counts = lengths(st_intersects(., r  
3         area = st_area(nz),  
4         density = counts/area)  
5 head(st_drop_geometry(nz.df[,7:10]))
```

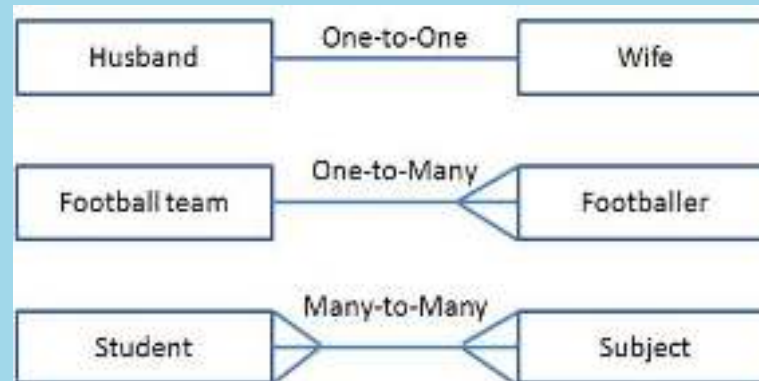
Estimating Density in Polygons

Estimating Distance

- As a covariate
- For use in covariance matrices
- As a means of assigning connections in networks

Estimating Single Point Distance

- **st_distance**
returns distances
between all features
in **x** and all features
in **y**
- One-to-One
relationship requires
choosing a single
point for **y**



Estimating Single Point Distance

- Subsetting **y** into a single feature

```
1 canterbury = nz %>% filter(Name == "Canterbury")
2 canterbury_height = nz_height[canterbury, ]
3 co = filter(nz, grepl("Canter|Otag", Name))
4 st_distance(nz_height[1:3, ], co)
```

Estimating Single Point Distance

- Using nearest neighbor distances

```
1 ua <- urban_areas(cb = FALSE, progress_bar
2   filter(., UATYP10 == "U") %>%
3   filter(., str_detect(NAME10, "ID")) %>%
4   st_transform(., crs=2163)
5
6 #get index of nearest ID city
7 nearest <- st_nearest_feature(ua)
8 #estimate distance
9 (dist = st_distance(ua, ua[nearest,], by_e
```

Topological Subsetting

Topological Subsetting

- Topological relations describe the spatial relationships between objects
- We can use the overlap (or not) of vector data to subset the data based on topology
- Need *valid* geometries
- Easiest way is to use `[` notation, but also most restrictive

```
1 ctby_height <- nz_height[canterbury, ]
```

Topological Subsetting

- Lots of verbs in **sf** for doing this (e.g., **st_intersects**, **st_contains**, **st_touches**)
- see **?geos_binary_pred** for a full list
- Creates an **implicit** attribute (the *records* in **x** that are “in” **y**)

Using **sparse=TRUE**

```
1 st_intersects(nz_height, co,  
2               sparse = TRUE)[1:3]  
3  
4 lengths(st_intersects(nz_height,  
5                       co, sparse =
```

Topological Subsetting

- The **sparse** option controls how the results are returned
- We can then find out if one or more elements satisfies the criteria

Using **sparse=FALSE**

```
1 st_intersects(nz_height, co, sparse = FALSE)[1:3,]  
2  
3 apply(st_intersects(nz_height, co, sparse = FALSE), 1, any)[1:3]
```

Topological Subsetting

```
1 canterbury_height3 = nz_height %>%  
2   filter(st_intersects(x = ., y = canterbu
```

Spatial Joins

Spatial Joins

- `sf` package provides `st_join` for vectors
- Allows joins based on the predicates (`st_intersects`, `st_touches`, `st_within_distance`, etc.)
- Default is a left join

Spatial Joins

```
1 set.seed(2018)
2 (bb = st_bbox(world)) # the world's
3 #>      xmin      ymin      xmax      ymax
4 #> -180.0   -89.9   180.0    83.6
5 random_df = data.frame(
6   x = runif(n = 10, min = bb[1], max = bb[3]),
7   y = runif(n = 10, min = bb[2], max = bb[4]),
8 )
9 random_points = random_df |>
10   st_as_sf(coords = c("x", "y")) |>
11   st_set_crs("EPSG:4326") # set geographic
12
13 random_joined = st_join(random_points, world)
```

Spatial Joins

- Sometimes we may want to be less restrictive
- Just because objects don't touch doesn't mean they don't relate to each other
- Can use `predicates` in `st_join`
- Remember that default is `left_join` (so the number of records can grow if multiple matches)

Spatial Joins

```
1 any(st_touches(cycle_hire, cycle_hire_osm, sparse
2 z = st_join(cycle_hire, cycle_hire_osm, st_is_with
3 nrow(cycle_hire)
4 nrow(z)
```

Extending Joins

Extending Joins

- Sometimes we are interested in analyzing locations that contain the overlap between two vectors
 - How much of home range a occurs on soil type b
 - How much of each Census tract is contained within a service provision area?
- `st_intersection`, `st_union`, and `st_difference` return new geometries that we can use as records in our spatial database

```
1 intersect_pct <- st_intersection(nc, st_area)
2   mutate(intersect_area = st_area)
3   dplyr::select(NAME, intersect_area)
4   st_drop_geometry()
5
6 nc <- mutate(nc, county_area = st_area)
7
8 # Merge by county name
9 nc <- merge(nc, intersect_pct, by = "NAME")
10
11 # Calculate coverage
12 nc <- nc %>%
13   mutate(coverage = as.numeric(intersect_area / county_area))
```

Extending Joins

