

Building Spatial Databases based on Location

HES 505 Fall 2024: Session 15

Carolyn Koehn

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

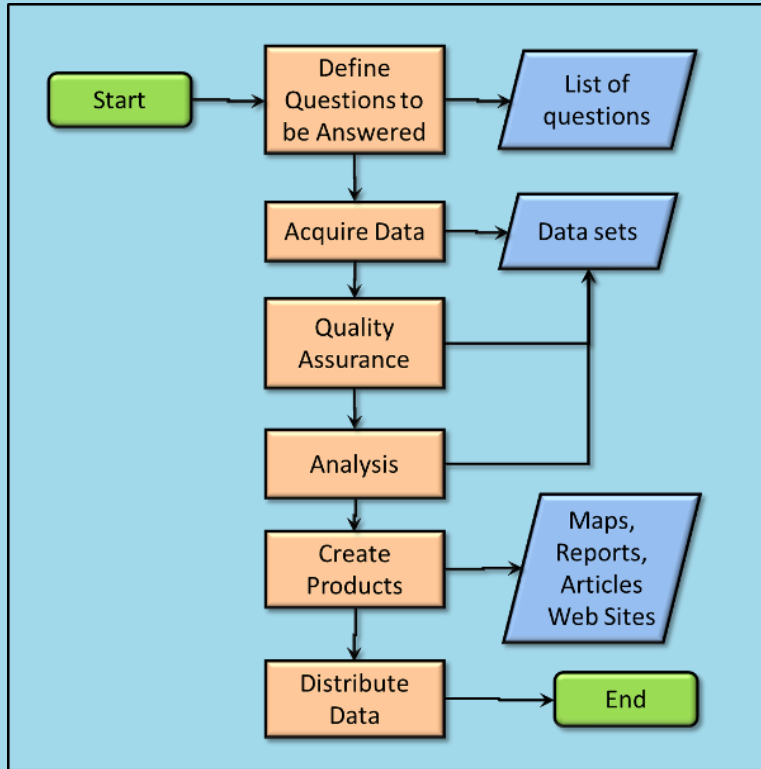
Revisiting Spatial Analysis

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	NEBRHC
1						
2	6474154.35276	10145.96973	20100400020000	HMAJCG	6795360.000000	M0000
3	7076794.10172	10644.47635	20100400010000	HFAJAG	6969600.000000	M0000
4	12993367.28984	15307.50117	20100300200000	HFAJAG	13229172.000000	M0000
5	2942042.70203	7688.52193	20100400030000	HPAJAG	2744260.000000	M0000
6	102725.86950	5216.30257	20100300190000	WBAC0A	187308.000000	M0000
7	78659.06631	3743.20269	20101000140000	WCAC0A	262656.000000	M0000
8	208715.26064	5238.44336	20101000140000	MROADA	219106.800000	M0000
9	12711289.20939	15000.44010	20100300100000	HFAJAG	13009700.000000	M0000
10	8530649.18776	11583.31722	20100200150000	HFAJAG	8819157.600000	M0000
11	2534604.48019	7728.17055	20100200200000	HFAJAG	2800472.400000	M0000
12	2459663.50513	7083.54911	20100200190000	HFAJAG	2090008.800000	M0000
13	4389060.54201	8923.10466	20100200180000	WCAC0A	4420468.800000	M0000
14	385170.08143	13827.90196	20100100450000	WGAC0A	402930.000000	M0000
15	8702821.65378	5361.67716	20100100150000	WCAC0A	8887962.400000	M0000
16	1488916.88618	2496.00860	20100100190000	WCAC0A	1494108.000000	M0000
17	229970.91558	4569.26427	20100100170000	WCAC0A	1153468.800000	M0000
18	1368014.23169	5911.54636	20100100110000	WCAC0A	1594296.000000	M0000
19	1615128.08861	752.44578	20100100140000	WCAC0A	35142.000000	M0070
20	32486.36388	3274.84752	20100510010000	A1E00A	600692.400000	M0000
21	595458.06886	28027.24631	20101000060000	WGAC0A	4194392.400000	M0000
22	3450710.31760	2466.20972	20100520010000	WGAC0A	236095.200000	E0000
23	210706.26281	15248.85526	20101000080000	WHAC0A	20037.600000	E0000
24	796557.93179	3775.17959	20100530060000	IAGAAB	235224.000000	E0000
25	269178.57938	808.95637	20100100130000	WCAC0A	30608.000000	M0070
26	37129.21791	2205.72911	20100530060000	A1D00A	161172.000000	E0000
27	158741.85422					

- Attributes: Information that further describes a spatial feature
- Attributes → predictors for analysis
- Monday's 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

courtesy of [Giscommons](#)

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

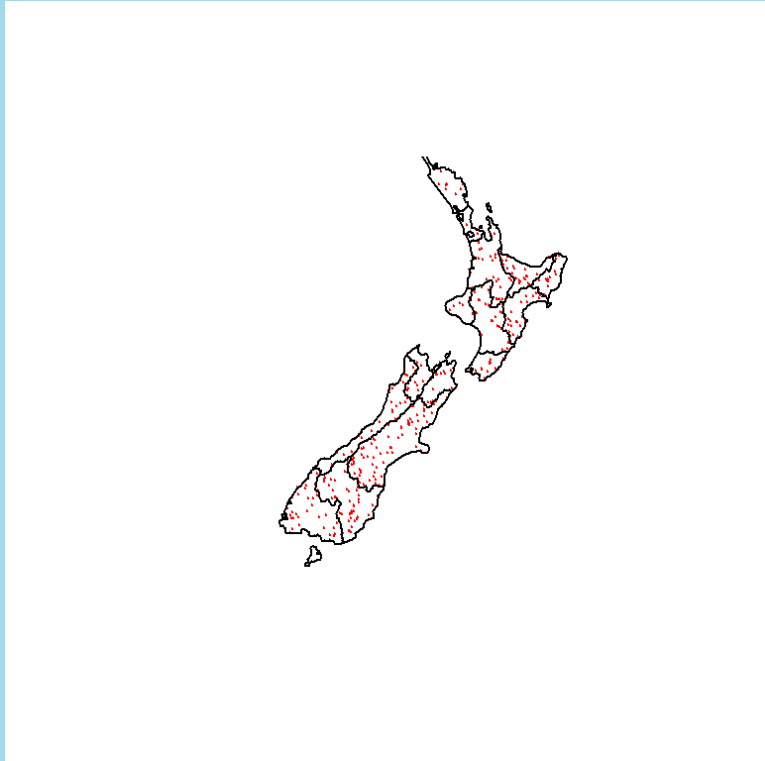
```
1 nz.sf <- nz %>%  
2   mutate(area = st_area(nz  
3   head(nz.sf$area, 3)
```

```
Units: [m^2]  
[1] 12890576439  4911565037  
24588819863
```

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

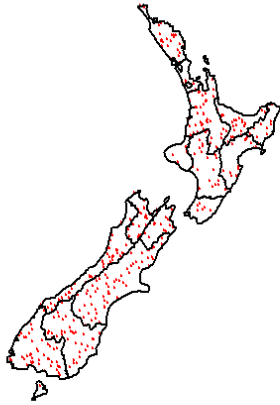
```
Units: [km^2]  
[1] 12890.576  4911.565  
24588.820
```

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

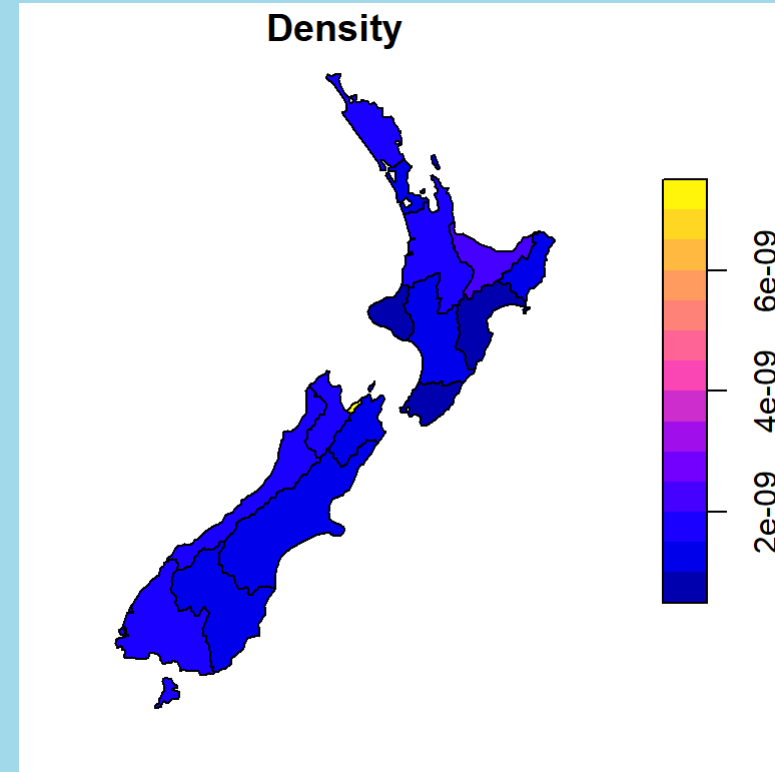
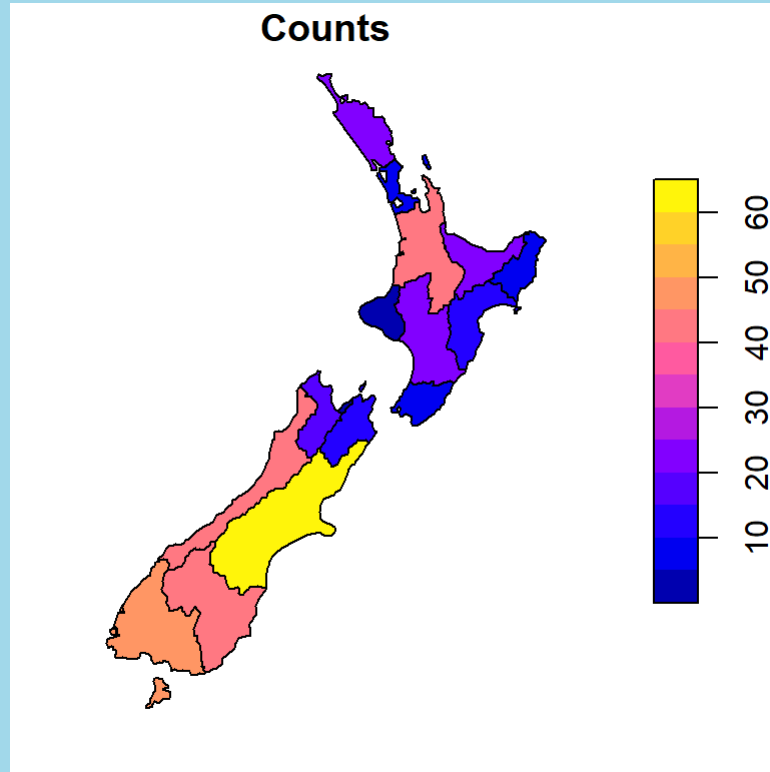
Estimating Density in Polygons



```
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]))
```

	counts	area
density		
1	24 12890576439 [m^2]	1.861825e-09
	[1/m^2]	
2	7 4911565037 [m^2]	1.425208e-09
	[1/m^2]	
3	42 24588819863 [m^2]	1.708093e-09
	[1/m^2]	
4	25 12271015945 [m^2]	2.037321e-09
	[1/m^2]	
5	10 8364554416 [m^2]	1.195521e-09
	[1/m^2]	
6	14 14242517871 [m^2]	9.829723e-10
	[1/m^2]	

Estimating Density in Polygons

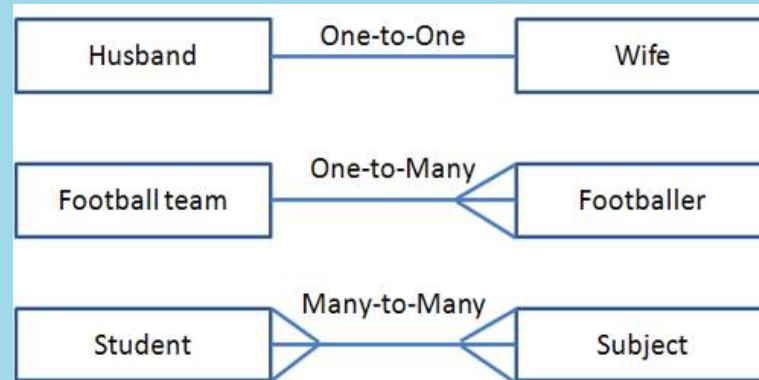


Estimating Distance

- 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**



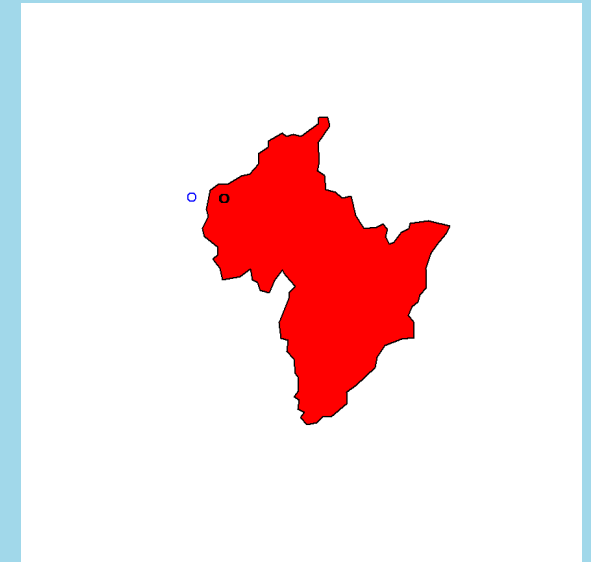
Estimating Single Point Distance

- 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)
```

Units: [m]

	[,1]	[,2]
[1,]	123537.16	15497.72
[2,]	94282.77	0.00
[3,]	93018.56	0.00



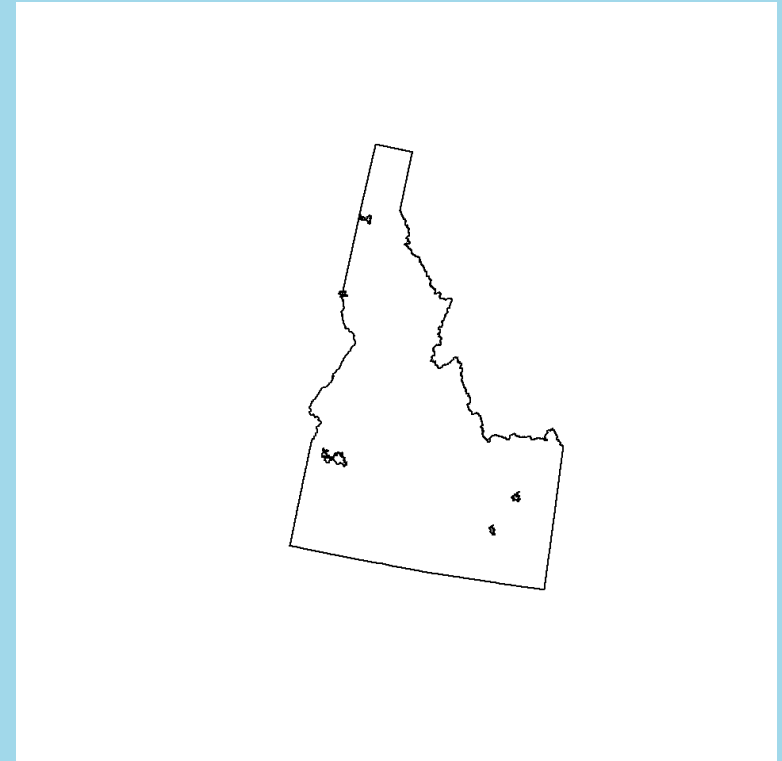
Estimating Single Point Distance

- 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
```

Units: [m]

```
[1] 61373.575 61373.575 1647.128
1647.128 136917.546 136917.546
```

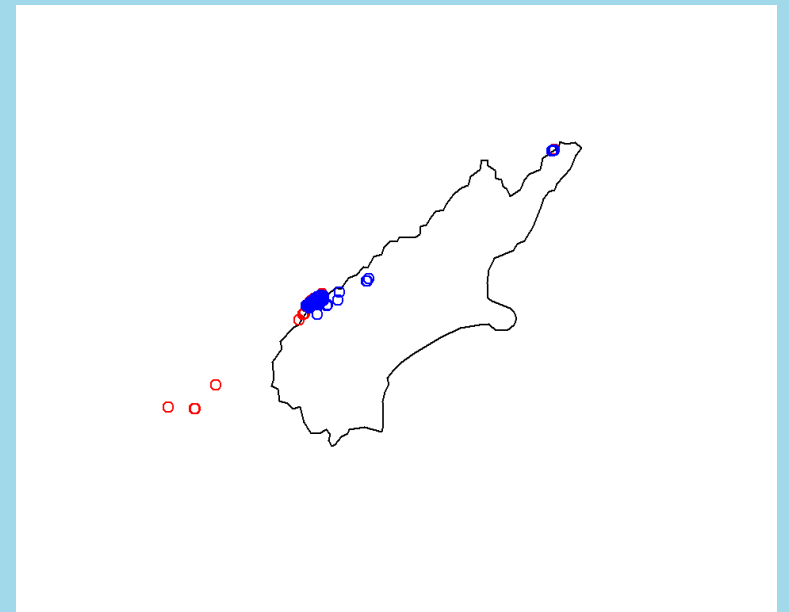


Topological Subsetting

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]
```

```
[[1]]  
integer(0)
```

```
[[2]]  
[1] 2
```

```
[[3]]  
[1] 2
```

```
1 lengths(st_intersects(nz_height,  
2                      co, sparse =
```

```
[1] FALSE TRUE TRUE
```

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,]
```

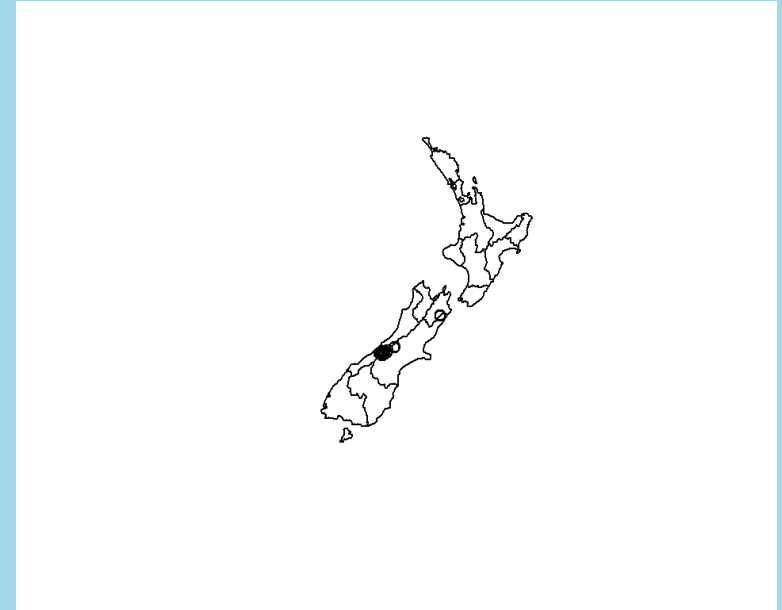
```
      [,1] [,2]  
[1,] FALSE FALSE  
[2,] FALSE  TRUE  
[3,] FALSE  TRUE
```

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

```
[1] FALSE  TRUE  TRUE
```

Topological Subsetting

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



Spatial Joins

Spatial Joins

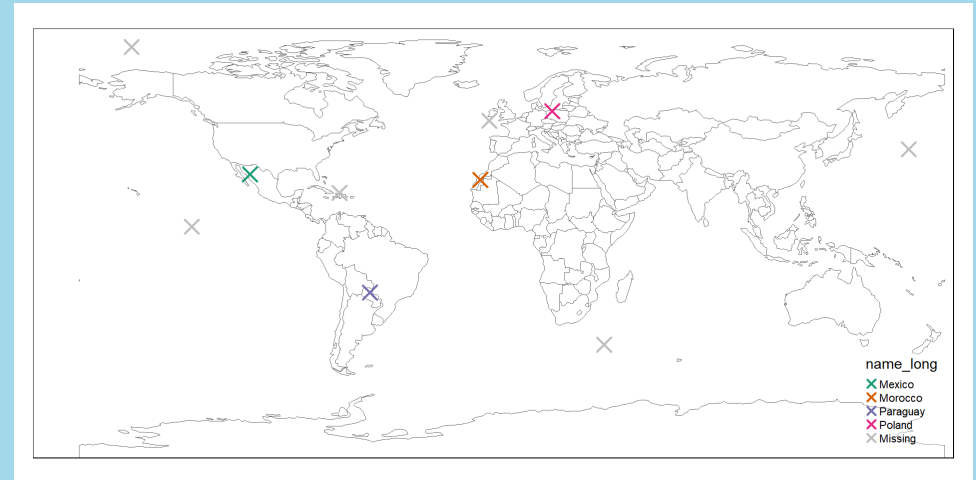
- `sf` package provides `st_join` for vectors
- Allows joins based on the predicates (`st_intersects`, `st_touches`, `st_within_distance`, etc.)
- Default is a left join

Spatial Joins

```
1 set.seed(2018)
2 (bb = st_bbox(world)) # the world'
```

```
      xmin      ymin      xmax
ymax
-180.00000 -89.90000  179.99999
83.64513
```

```
1 #>      xmin      ymin      xmax      ymax
2 #> -180.0   -89.9   180.0    83.6
3 random_df = data.frame(
4   x = runif(n = 10, min = bb[1], m
5   y = runif(n = 10, min = bb[2], m
6 )
7 random_points <- random_df %>%
8   st_as_sf(coords = c("x", "y")) %
9   st_set_crs("EPSG:4326") # set ge
10
11 random_joined = st_join(random_poi
```



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
```

```
[1] FALSE
```

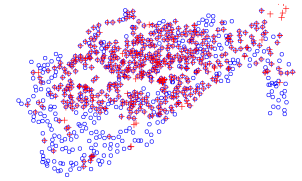
```
1 z = st_join(cycle_hire, cycle_hire_osm, st_is_within
```

```
2 nrow(cycle_hire)
```

```
[1] 742
```

```
1 nrow(z)
```

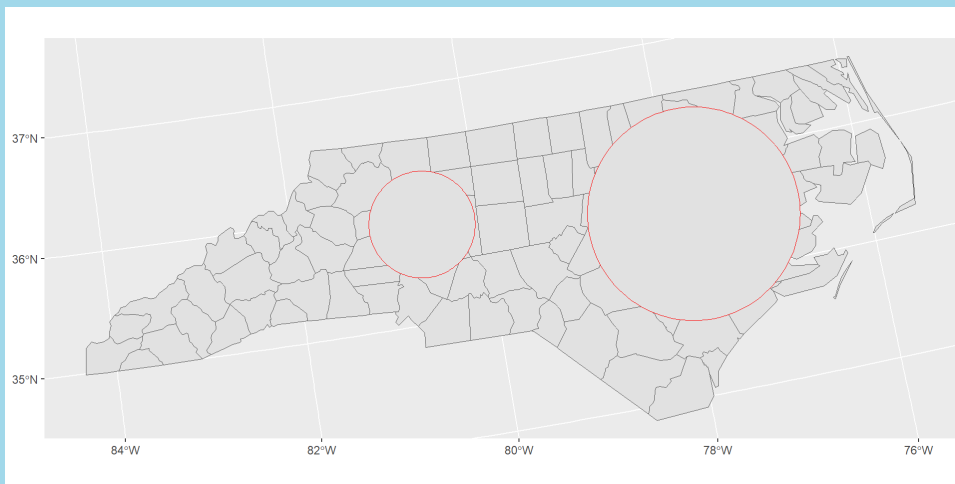
```
[1] 762
```



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, home_range)
2   mutate(intersect_area = st_area(intersect_pct))
3   dplyr::select(NAME, intersect_area)
4   st_drop_geometry()
5
6 nc <- mutate(nc, county_area = st_area(county_geometry))
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

