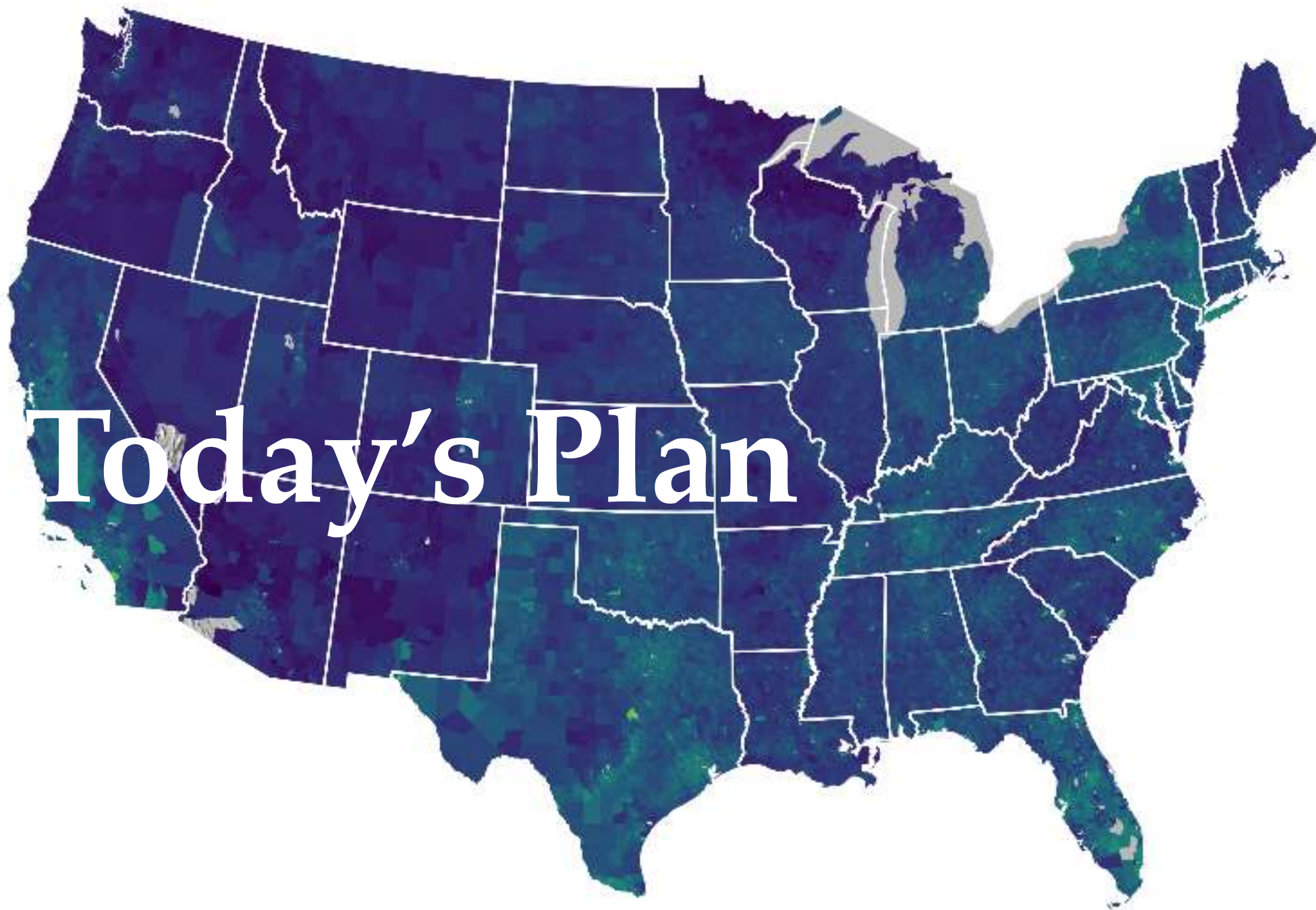


# Combining Tabular and Spatial Data

HES 505 Fall 2023: Session 15

Matt Williamson



# Objectives

By the end of today, you should be able to:

- Define *spatial analysis*
- Describe the steps in planning a spatial analysis
- Understand the structure of relational databases
- Use attributes and topology to subset data
- Generate new features using geographic data
- Join data based on attributes and location

# What is 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

# What is spatial analysis?

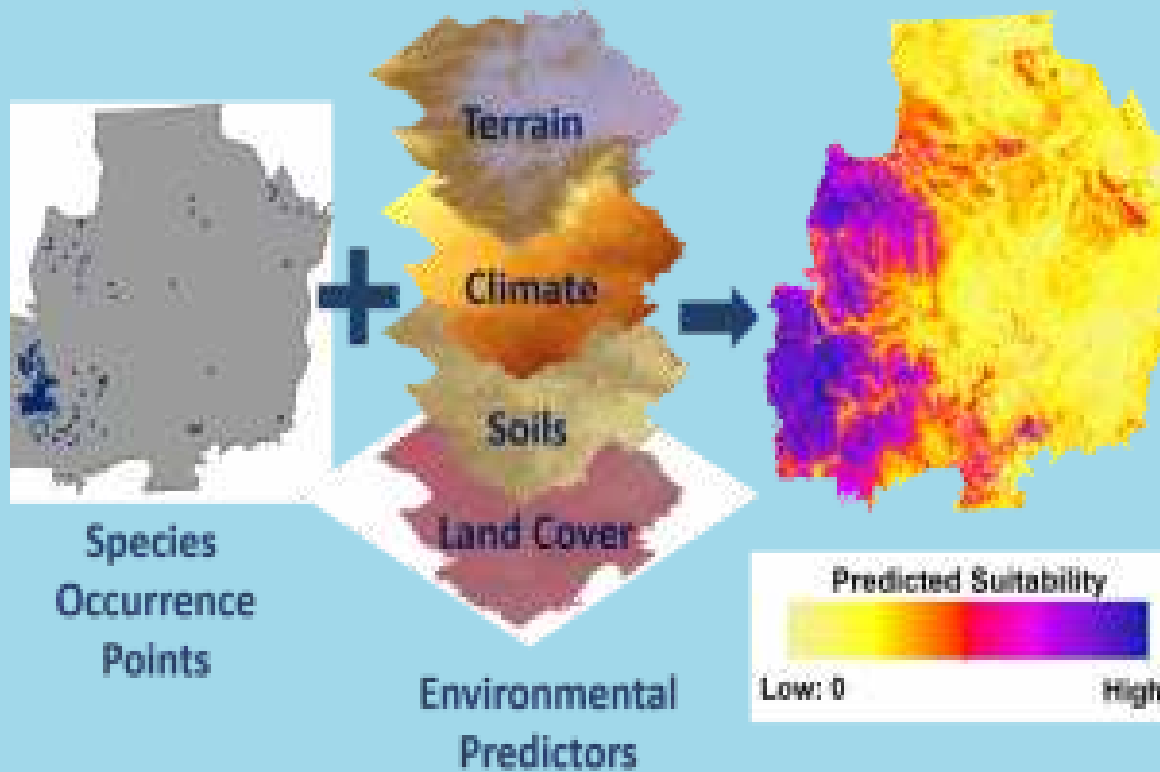
- The process of turning maps into information
- Any- or everything we do with GIS
- The use of computational and statistical algorithms to understand the relations between things that co-occur in space.



John Snow's cholera outbreak map

# Common goals for spatial analysis

## Building a Model



- Describe and visualize locations or events
- Quantify patterns
- Characterize 'suitability'
- Determine (statistical) relations

courtesy of NatureServe

# Common pitfalls of spatial analysis

- **Locational Fallacy:** Error due to the spatial characterization chosen for elements of study
- **Atomic Fallacy:** Applying conclusions from individuals to entire spatial units
- **Ecological Fallacy:** Applying conclusions from aggregated information to individuals

Spatial analysis is an inherently complex endeavor and one that is advancing rapidly. So-called “best practices” for addressing many of these issues are still being developed and debated. This doesn’t mean you shouldn’t do spatial analysis, but you should keep these things in mind as you design, implement, and interpret your analyses



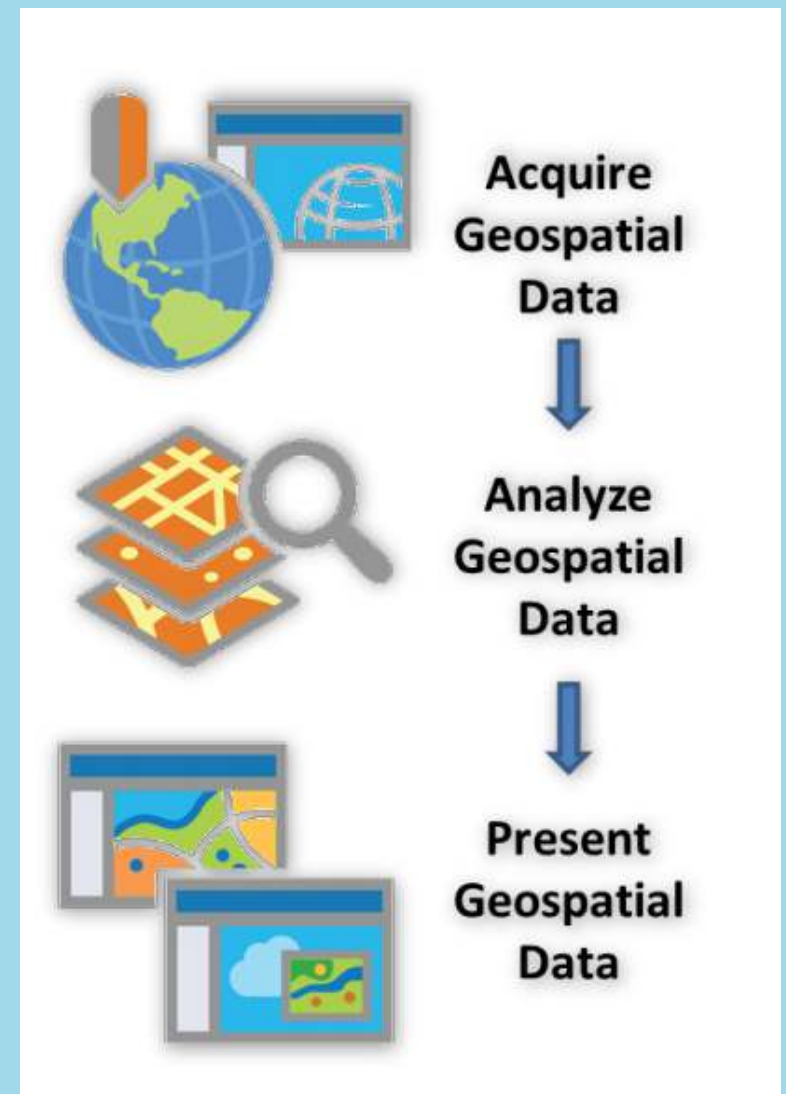
# Workflows for spatial analysis

# Workflows for spatial analysis

## Workflow for spatial analysis

## Workflow for spatial analysis

- Acquisition (not really a focus, but see **Resources**)
- Geoprocessing
- Analysis
- Visualization



courtesy of **University of Illinois**

# Geoprocessing

Manipulation of data for subsequent use

- Alignment
- Data cleaning and transformation
- Combination of multiple datasets
- Selection and subsetting

# Databases and Attributes

# Databases and Attributes

The image shows a screenshot of a GIS database table. A blue box labeled 'Field' highlights the first column, 'AREA'. A yellow box labeled 'Key ID Field' highlights the second column, 'PSIDM100'. A green box labeled 'Record' highlights the entire row for the first record. The table contains 27 rows of data with columns: AREA, PSIDM100, SPN, LANDUSE, LOT\_SIZE, and PERM100.

	A	B	C	D	E	F
	AREA	PSIDM100	SPN	LANDUSE	LOT_SIZE	PERM100
1	6474154.36276	12146.88872	20100400020000	HFAJAG	6790000.000000	M0000
2	7070794.10172	13641.47836	20100400015000	HFAJAG	6969000.000000	M0000
3	12903367.28864	16307.62117	20100300000000	HFAJAG	12209172.000000	M0000
4	2840042.70203	7686.62192	20100400030000	HFAJAG	2744200.000000	M0000
5	102726.06962	1218.30267	20100200190000	WACADA	187300.000000	M0000
6	38685.66621	1713.30267	20100200140000	WACADA	262550.000000	M0000
7	238716.26884	3728.41236	20120000000000	MRCADA	249106.800000	M0000
8	12711880.96556	18836.44816	20100200100000	HFAJAG	13009160.000000	M0000
9	9620649.10776	11660.31722	20100200162000	HFAJAG	8019167.600000	M0000
10	2634654.48219	7726.17262	20100200000000	HFAJAG	2600472.400000	M0000
11	2403632.50813	7032.64811	20100200190000	HFAJAG	2080000.800000	M0000
12	4089060.54201	28272.10462	20100200100000	WACADA	4420468.800000	M0000
13	385170.08142	13627.20329	20100100462000	WACADA	432930.000000	M0000
14	8702821.66376	13627.90196	20100100150000	WACADA	8887962.400000	M0000
15	1408916.68818	13617.67716	20100100190000	WACADA	1484100.000000	M0000
16	228970.91558	2436.00862	20100100160000	WACADA	217364.400000	M0000
17	1368014.23168	4869.26427	20100100170000	WACADA	1153468.800000	M0000
18	1815128.08861	4871.54826	20100100110000	WACADA	1954296.000000	M0000
19	32486.36388	782.44618	20100100140000	WACADA	35142.000000	M0070
20	895458.06886	3274.64752	20100510015000	ATBDA	630692.400000	M0000
21	3458710.31762	28027.24831	20101000000000	WACADA	4184392.400000	M0000
22	210796.26201	2465.23972	20100530015000	WACADA	236096.200000	M0000
23	796957.83179	11026.18576	20101000000000	WACADA	20037.800000	M0000
24	280178.57538	8774.17183	20100530020000	WACADA	235224.000000	M0000
25	37129.21791	806.95632	20100100130000	WACADA	30808.000000	M0070
26	158741.88422	1229.72911	20100530060000	ATBDA	181172.000000	M0000

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

# Databases and attributes

	A	B	C	D	E	F
	AREA	PSYCHID	APN	LANDUSE	LOT SIZE	PERMITS
1	6474154.36275	12146.88872	20100400000000	HFAJAG	679000.000000	40000
2	7070794.10172	13641.47836	20100400015000	HFAJAG	696900.000000	40000
3	1290307.20864	16307.62117	20100300000000	HFAJAG	12209172.000000	40000
4	2840042.70203	7006.62192	20100400000000	HFAJAG	2744200.000000	40000
5	102725.00962	12146.88872	20100300019000	WACADA	187300.000000	40000
6	20000.00000	12146.88872	20100300040000	WACADA	262000.000000	40000
7	238716.26884	12146.88872	20100300000000	WACADA	200000.000000	40000
8	12146.88872	12146.88872	20100300000000	HFAJAG	1000000.000000	40000
9	9630649.10776	11650.31722	20100200016000	HFAJAG	8019167.000000	40000
10	2634654.48019	7726.17282	20100200000000	HFAJAG	2000472.000000	40000
11	2408052.50813	7002.64811	20100200019000	HFAJAG	2000000.000000	40000
12	4009060.54001	2007.10462	20100200010000	WACADA	4420408.000000	40000
13	385170.08142	13627.90196	20100100460000	WACADA	422930.000000	40000
14	8702821.66378	13627.90196	20100100150000	WACADA	8887962.000000	40000
15	1408916.00818	13617.67718	20100100190000	WACADA	1494100.000000	40000
16	228970.91558	2496.00062	20100100160000	WACADA	217364.000000	40000
17	1368014.23168	4899.26427	20100100170000	WACADA	1153468.000000	40000
18	1815128.08861	4891.54836	20100100110000	WACADA	1554296.000000	40000
19	32486.36388	1702.44618	20100100140000	WACADA	35142.000000	40000
20	895458.06896	3274.24752	20100510015000	ATIDEA	630672.000000	40000
21	3458710.31782	28027.24834	20101000000000	WACADA	4184302.000000	40000
22	210796.26201	2465.23972	20100500015000	WACADA	236095.200000	20000
23	796957.83179	11004.88528	20101000000000	WACADA	20037.800000	20000
24	280178.57538	8774.17183	20100500050000	WACADA	235224.000000	20000
25	37128.21791	806.95632	20100100130000	WACADA	30008.000000	40000
26	198741.88422	1229.72111	20100500060000	ATIDEA	181172.000000	20000

- Previous focus has been largely on *location*
- Geographic data often also includes non-spatial data
- Attributes: Non-spatial information that further describes a spatial feature
- Typically stored in tables where each row represents a spatial feature
  - Wide vs. long format

courtesy of [Giscommons](#)

# Common attribute operations

- `sf` designed to work with `tidyverse`
- Allows use of `dplyr` data manipulation verbs (e.g. `filter`, `select`, `slice`)
- Can use `scales` package for units
- Also allows `%>%` to chain together multiple steps
- geometries are “sticky”



# Subsetting by Field

# Subsetting by Features

- Features refer to the individual observations in the dataset
- Selecting features

```
1 head(world)[1:3, 1:3] %>%  
2   st_drop_geometry()
```

```
# A tibble: 3 × 3  
  iso_a2 name_long      continent  
* <chr>  <chr>         <chr>  
1 FJ     Fiji         Oceania  
2 TZ     Tanzania       Africa  
3 EH     Western Sahara Africa
```

```
1 world %>%  
2   filter(continent == "Asia") %>%  
3     dplyr::select(name_long, conti  
4   st_drop_geometry() %>%  
5   head(.)
```

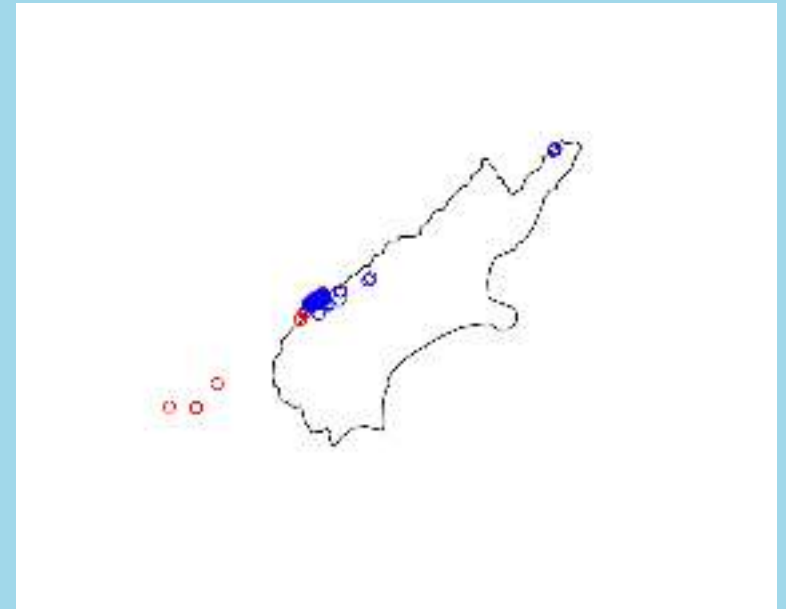
```
# A tibble: 6 × 2  
  name_long      continent  
  <chr>         <chr>  
1 Kazakhstan Asia  
2 Uzbekistan Asia  
3 Indonesia Asia  
4 Timor-Leste Asia  
5 Israel Asia  
6 Lebanon Asia
```

# Spatial 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 canterbury = nz %>% filter(Name == "Canterbury")
2 canterbury_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 co = filter(nz, grepl("Canter|Otago", nz$name))
2 st_intersects(nz_height, co,
3               sparse = TRUE)[1:3]
```

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

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

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

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

```
[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
```



# New Attributes from Existing Fields



# Revisiting the tidyverse

- Creating new fields

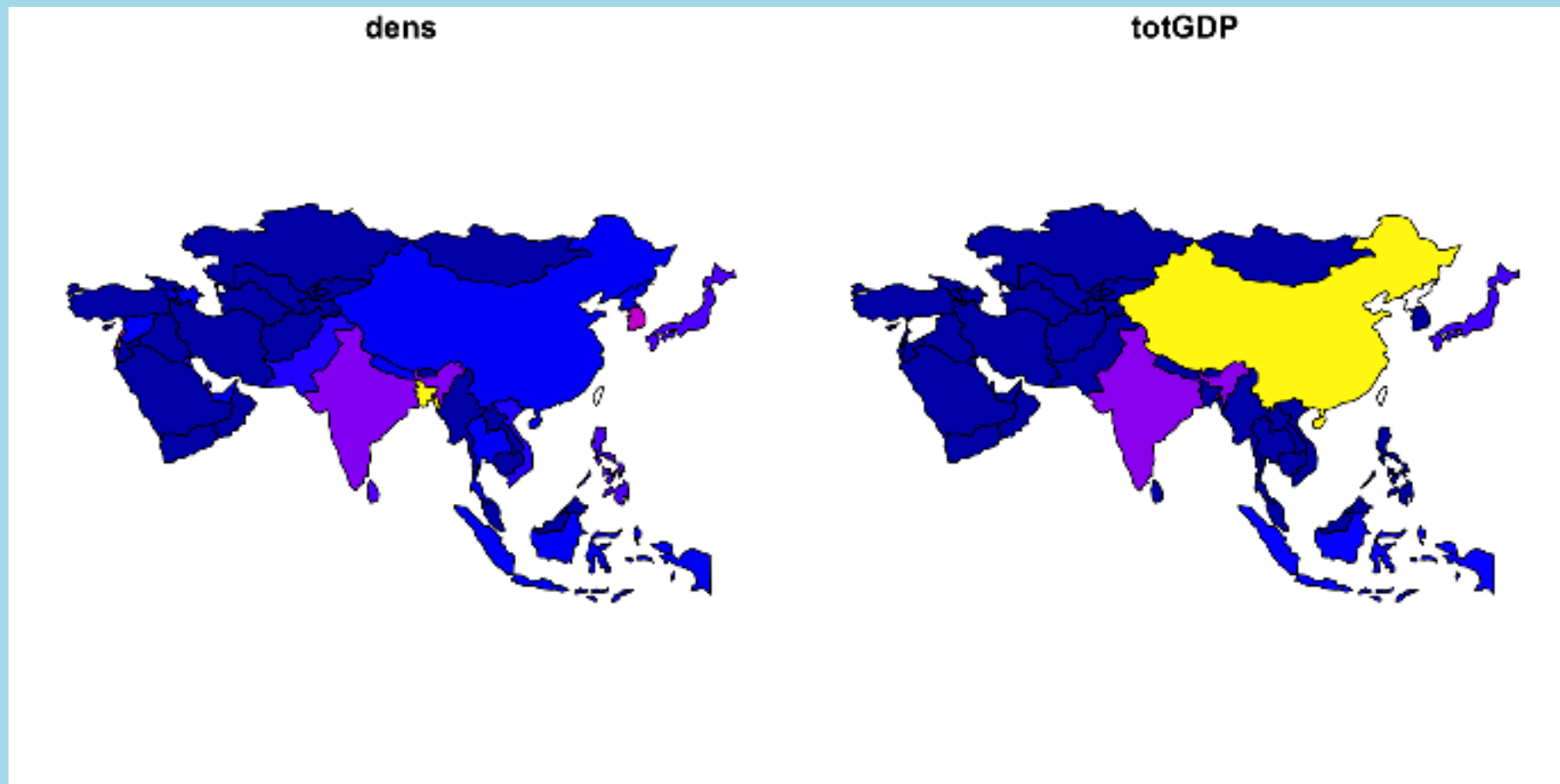
```
1 world %>%
2   filter(continent == "Asia") %>%
3   dplyr::select(name_long, continent, pop, gdpPercap ,area_km2) %>%
4   mutate(., dens = pop/area_km2,
5           totGDP = gdpPercap * pop) %>%
6   st_drop_geometry() %>%
7   head(.)
```

# A tibble: 6 × 7

	name_long	continent	pop	gdpPercap	area_km2	dens	totGDP
	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	Kazakhstan	Asia	17288285	23587.	2729811.	6.33	4.08e11
2	Uzbekistan	Asia	30757700	5371.	461410.	66.7	1.65e11
3	Indonesia	Asia	255131116	10003.	1819251.	140.	2.55e12
4	Timor-Leste	Asia	1212814	6263.	14715.	82.4	7.60e 9
5	Israel	Asia	8215700	31702.	22991.	357.	2.60e11
6	Lebanon	Asia	5603279	13831.	10099.	555.	7.75e10

# Revisiting the tidyverse

- Creating new fields



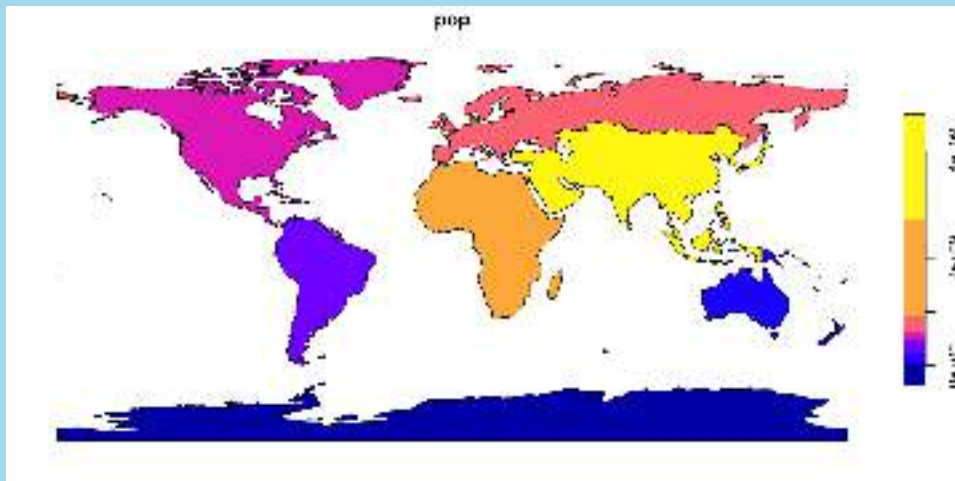
# Revisiting the tidyverse

- Aggregating data

```
1 world %>%  
2   st_drop_geometry(.) %>%  
3   group_by(continent) %>%  
4   summarize(pop = sum(pop, na.rm =
```

```
# A tibble: 8 × 2
```

	continent	pop
	<chr>	<dbl>
1	Africa	1154946633
2	Antarctica	0
3	Asia	4311408059
4	Europe	669036256
5	North America	565028684
6	Oceania	37757833
7	Seven seas (open ocean)	0
8	South America	412060811



# New Attributes from Topology

# 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

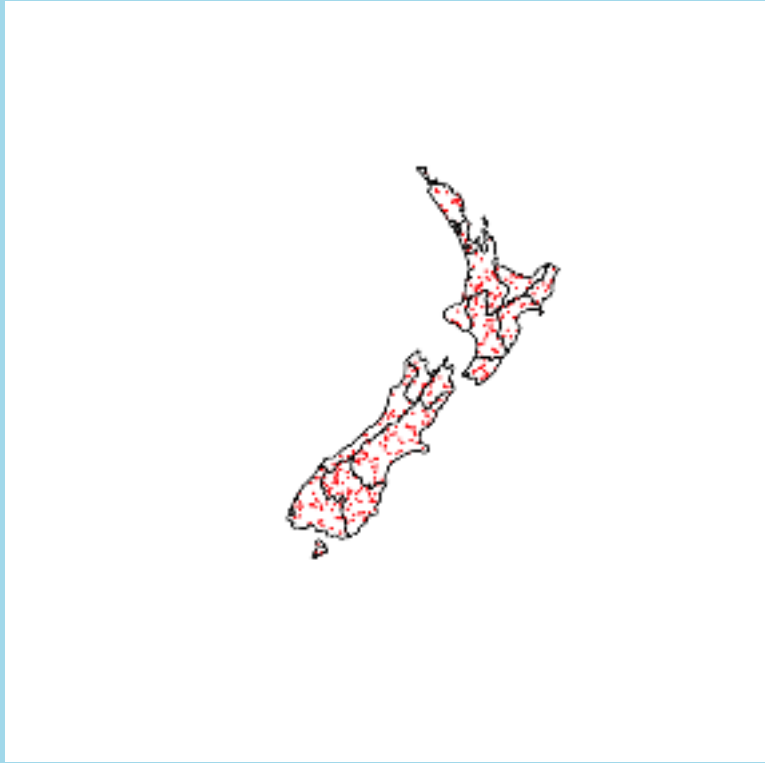
```
1 nz.sf <- nz %>%  
2   mutate(area = st_area(nz  
3 head(nz.sf$area, 3)
```

```
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



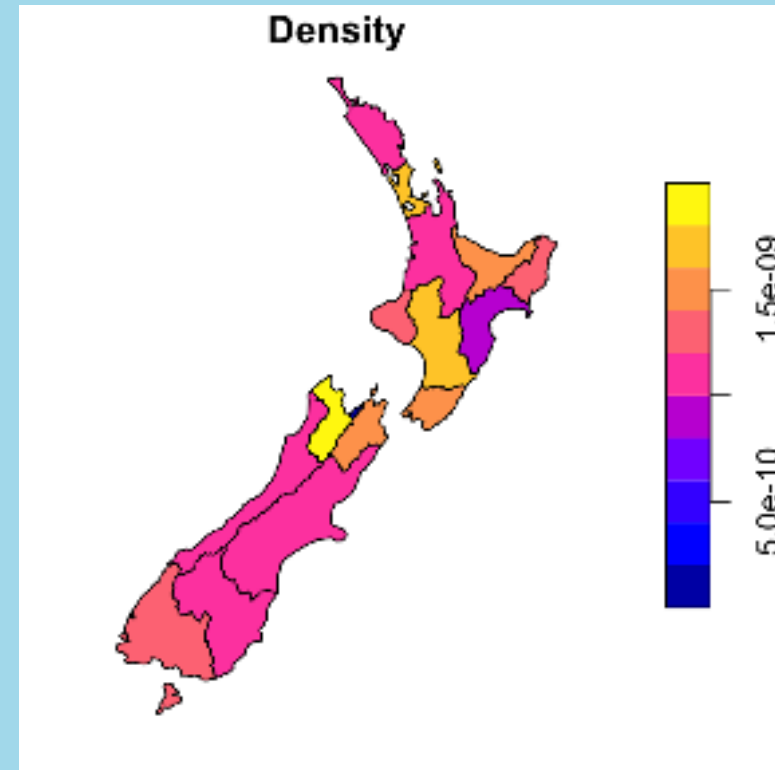
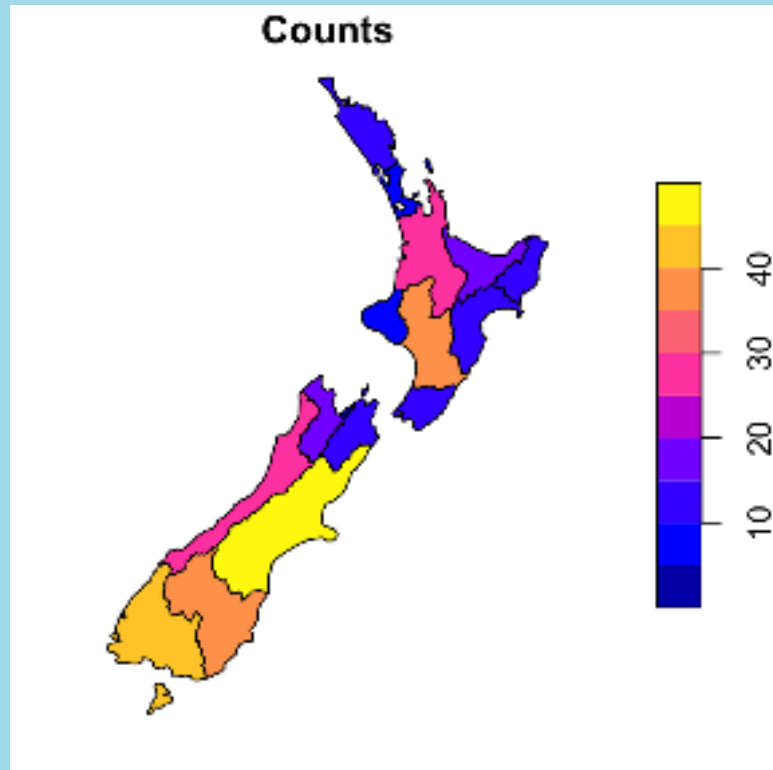
# Estimating Density in Polygons



```
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	14 12890576439 [m^2]	1.086065e-09
	[1/m^2]	
2	8 4911565037 [m^2]	1.628809e-09
	[1/m^2]	
3	28 24588819863 [m^2]	1.138729e-09
	[1/m^2]	
4	18 12271015945 [m^2]	1.466871e-09
	[1/m^2]	
5	11 8364554416 [m^2]	1.315073e-09
	[1/m^2]	
6	14 14242517871 [m^2]	9.829723e-10
	[1/m^2]	

# Estimating Density in Polygons

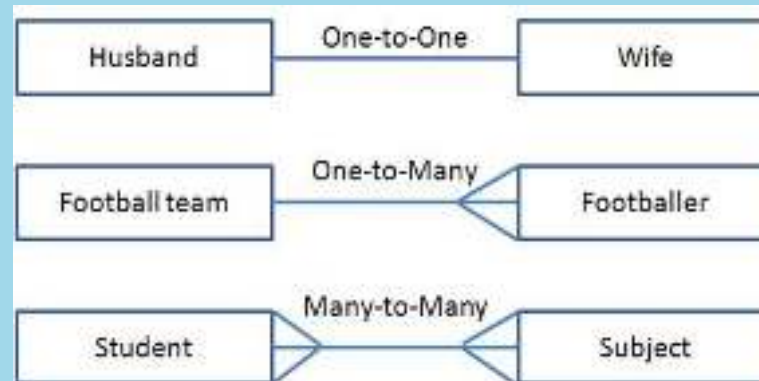


# Estimating Distance

- 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



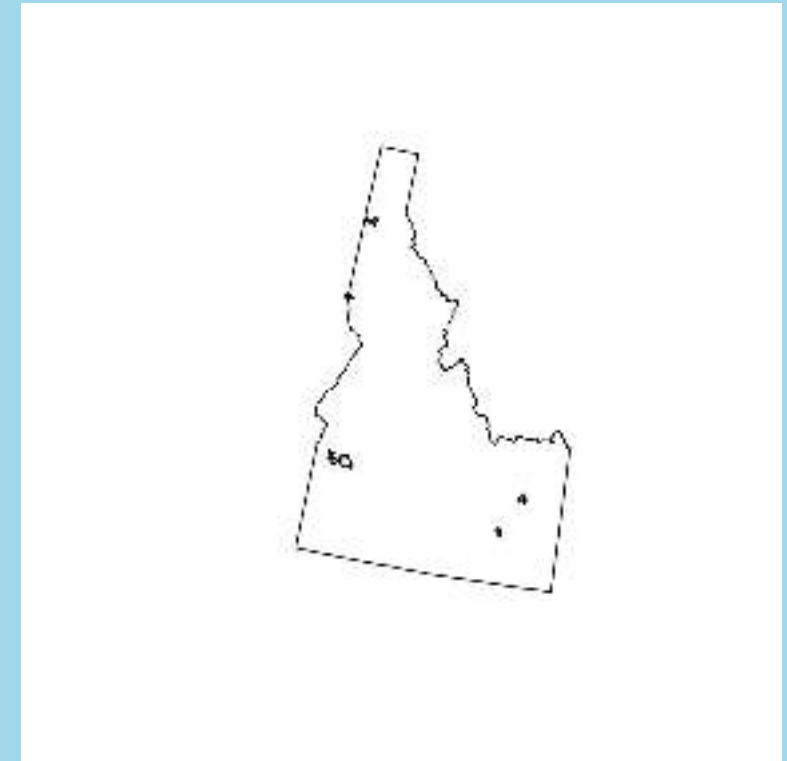
# Estimating Single Point Distance

- 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]

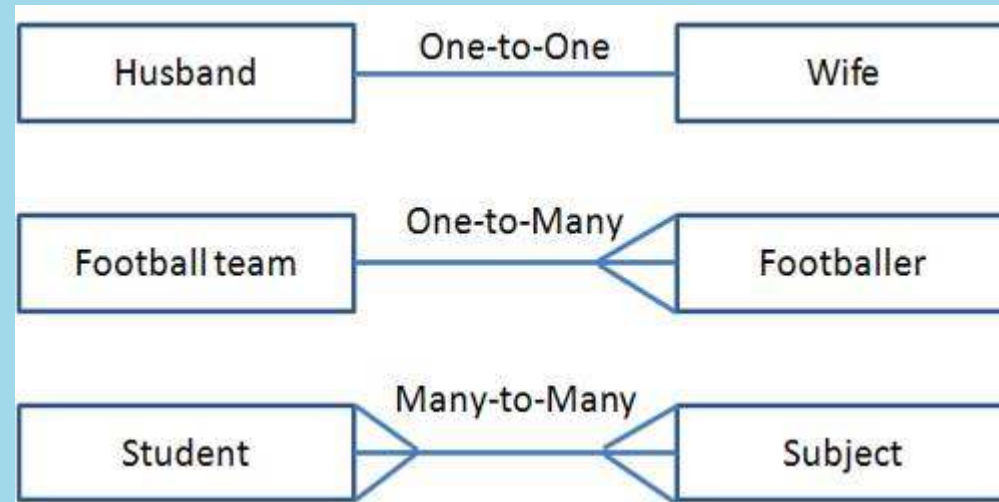
```
[1] 61386.444 61386.444 1646.182
1646.182 136908.183 136908.183
```



# Joining (a)spatial data

# Joining (a)spatial data

- Requires a “key” field
- Multiple outcomes possible
- Think about your final data form



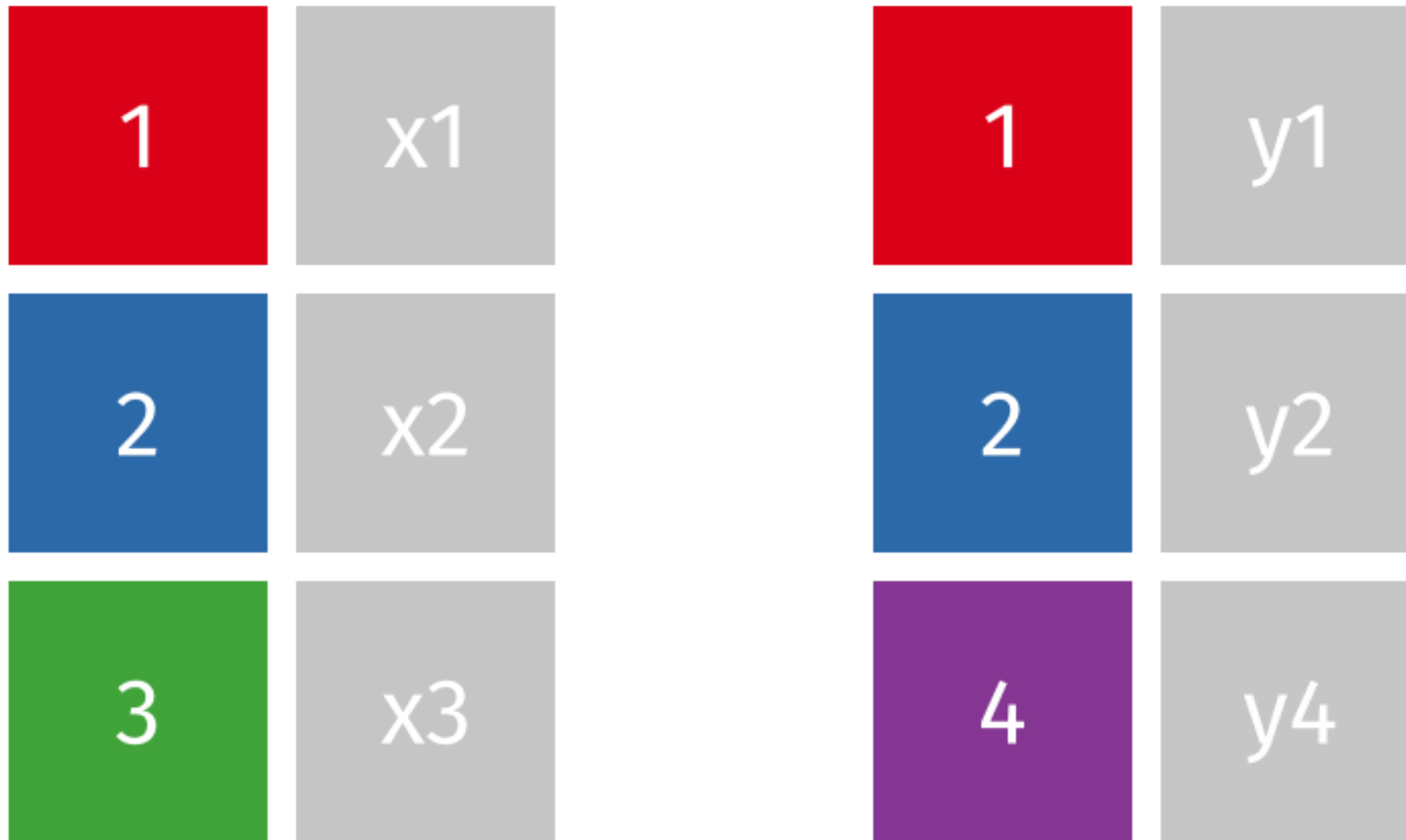


# Left Join

- Useful for adding other attributes not in your spatial data
- Returns all of the records in **x** attributed with **y**
- Pay attention to the number of rows!

# Left Join

`left_join(x, y)`



# Left Join

```
1 head(coffee_data)
```

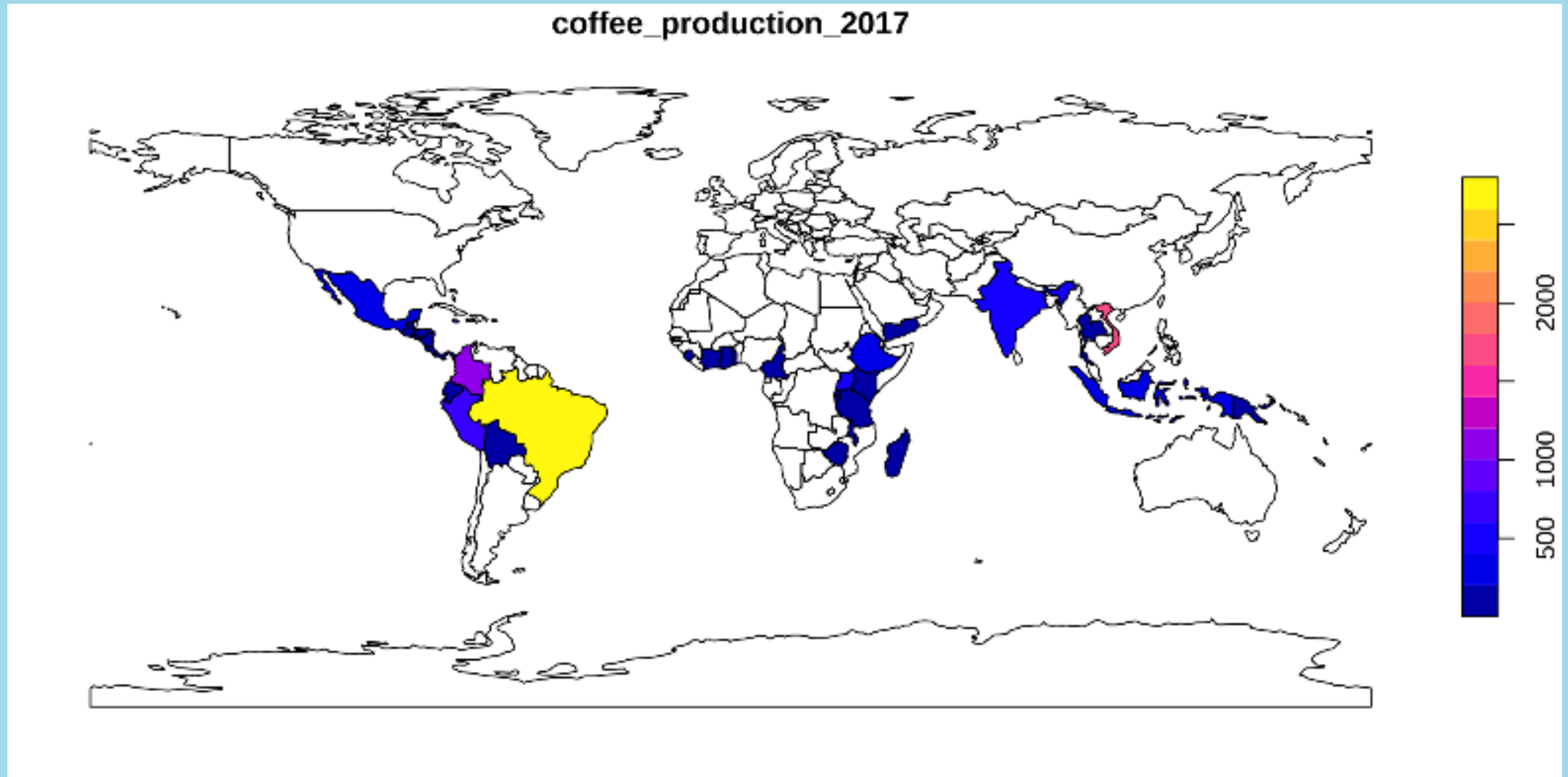
```
# A tibble: 6 × 3
```

name_long	coffee_production_2016	coffee_production_2017
<chr>	<int>	<int>
1 Angola		
NA		NA
2 Bolivia		
3		4
3 Brazil		
3277		2786
4 Burundi		
37		38
5 Cameroon		
~		~

```
1 world_coffee = left_join(world, coffee_data)
2 nrow(world_coffee)
```

```
[1] 177
```

# Left Join



# Inner Join

- Useful for subsetting to “complete” records
- Returns all of the records in **x** with matching **y**
- Pay attention to the number of rows!

# Inner Join

`inner_join(x, y)`

1	x1	1	y1
2	x2	2	y2
3	x3	4	y4

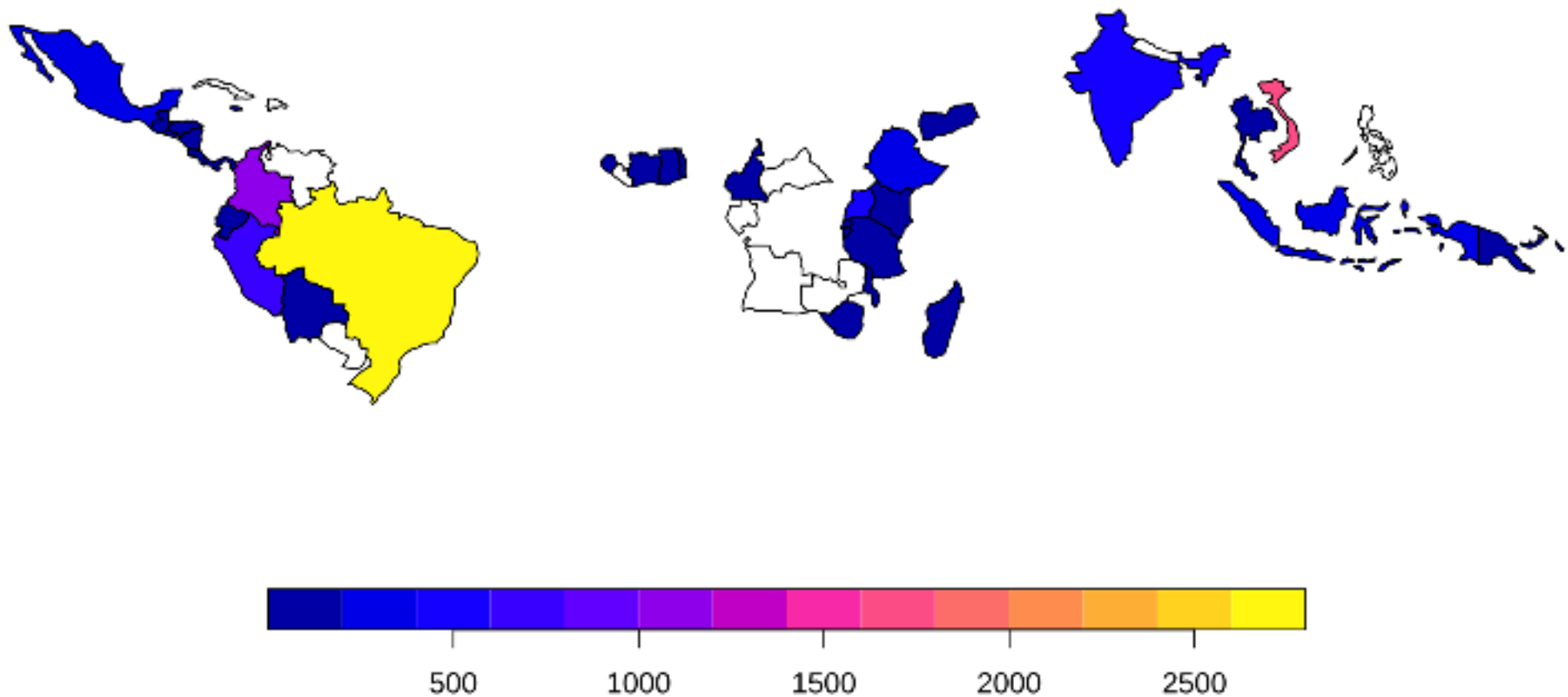
# Inner Join

```
1 world_coffee_inner = inner_join(wc
2 nrow(world_coffee_inner)
[1] 45
```

```
1 setdiff(coffee_data$name_long, wor
[1] "Congo, Dem. Rep. of" "Others"
```

# Inner Join

coffee\_production\_2017





# Spatial Joins

# Spatial Joins

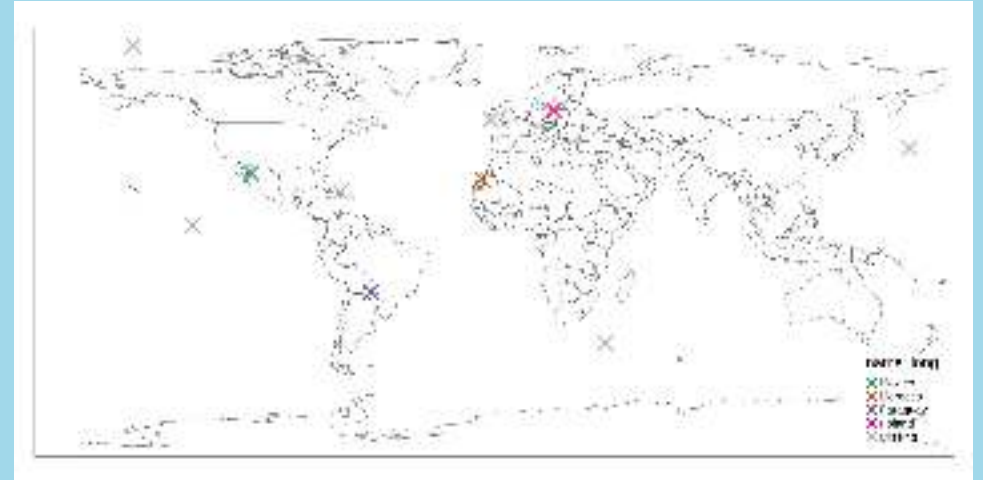
- `sf` package provides `st_join` for vectors
- Allows joins based on the predicates (`st_intersects`, `st_touches`, `st_within_distance`, etc.)
- Default is a left join

# 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
```

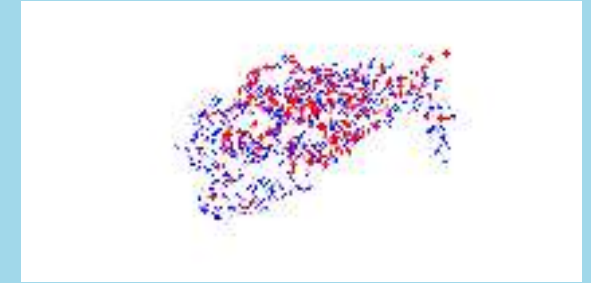
```
1 z = st_join(cycle_hire, cycle_hire_osm, st_is_within
```

```
2 nrow(cycle_hire)
```

```
[1] 742
```

```
1 nrow(z)
```

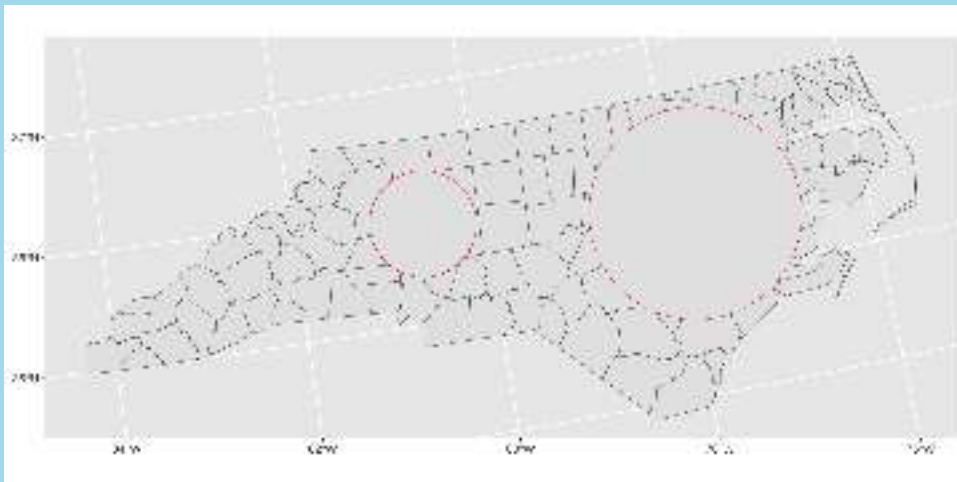
```
[1] 762
```



# 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, 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

