Combining Tabular and Spatial Data

HES 505 Fall 2023: Session 15

Matt Williamson

Today's Plan

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

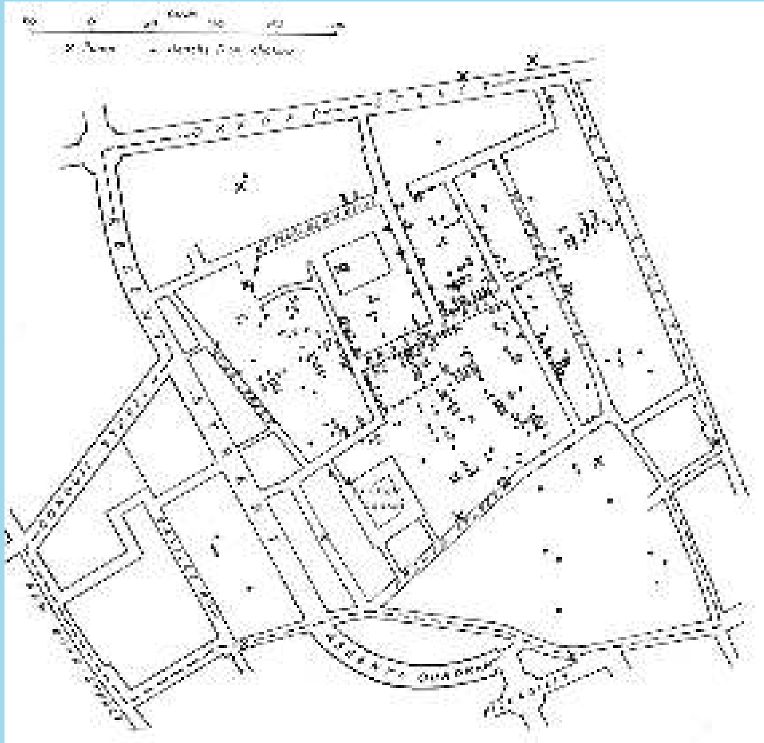
What is spatial analysis?

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

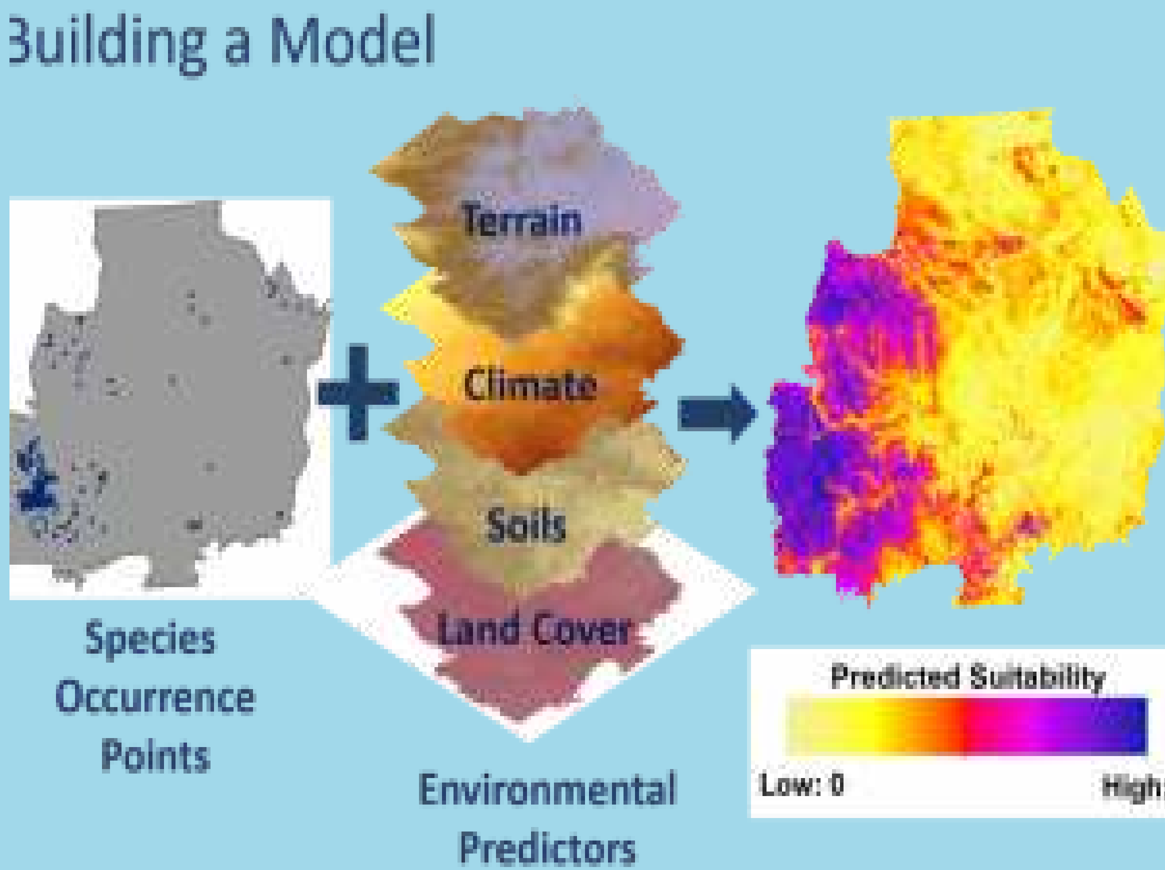
What is spatial analysis?

- 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



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

courtesy of NatureServe

Common pitfalls of spatial analysis

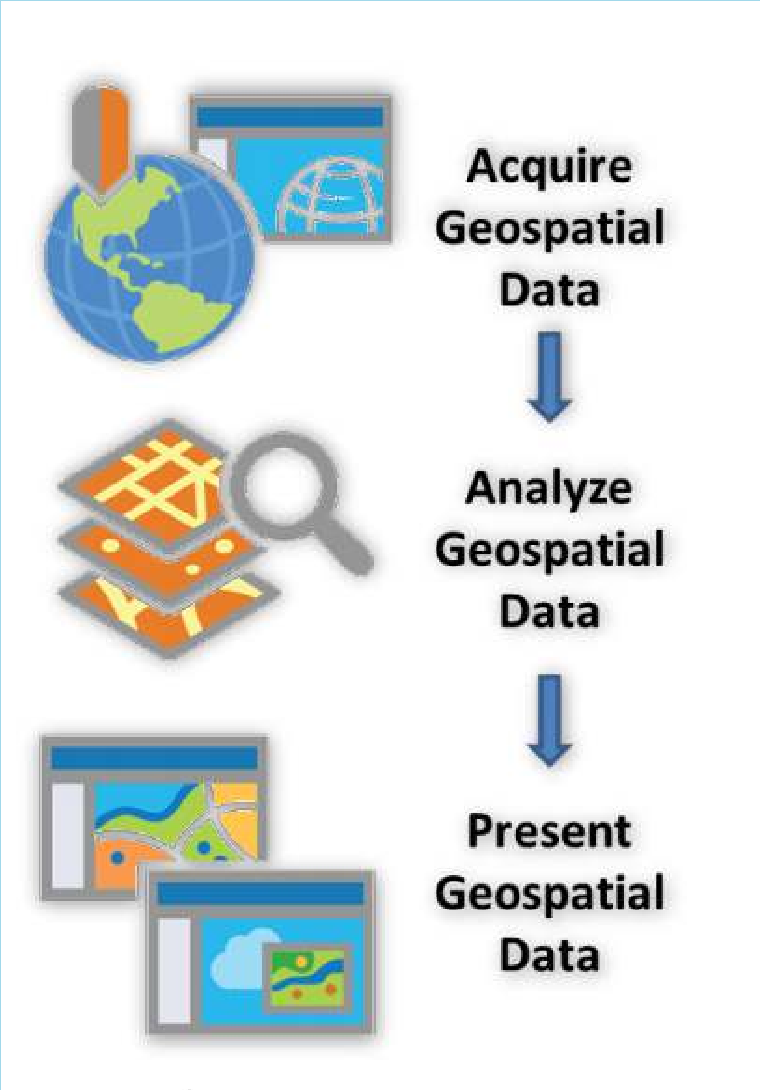
- **Locational Fallacy:** Error due to the spatial characterization chosen for elements of study
- **Atomic Fallacy:** Applying conclusions from individuals to entire spatial units
- **Ecological Fallacy:** Applying conclusions from aggregated information to individuals

> 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

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



courtesy of University of Illinois

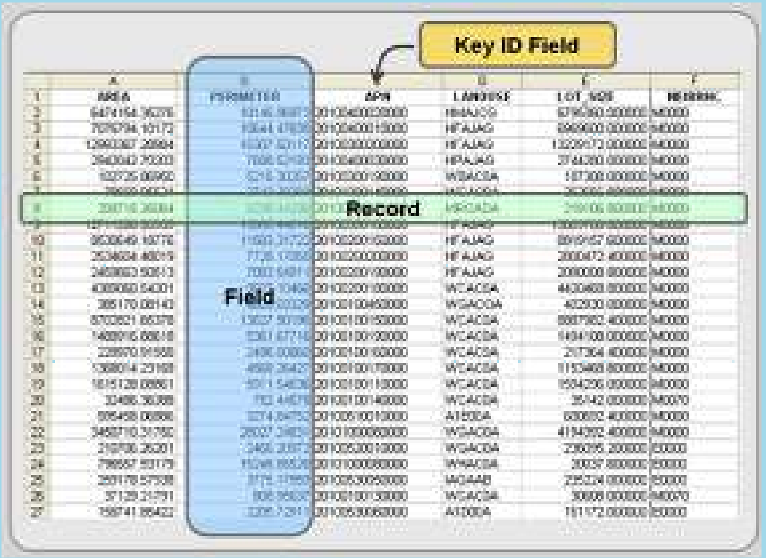
Geoprocessing

Manipulation of data for subsequent use

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

Databases and Attributes

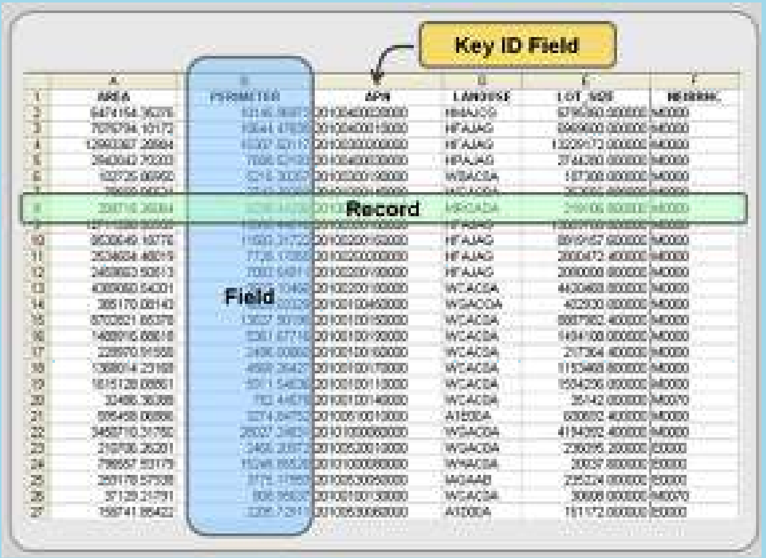
Databases and Attributes



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



courtesy of Giscommons

- 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

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

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

```r
world %>%
  filter(continent == "Asia") %>%
    dplyr::select(name_long, conti
  st_drop_geometry() %>%
  head(.)
```

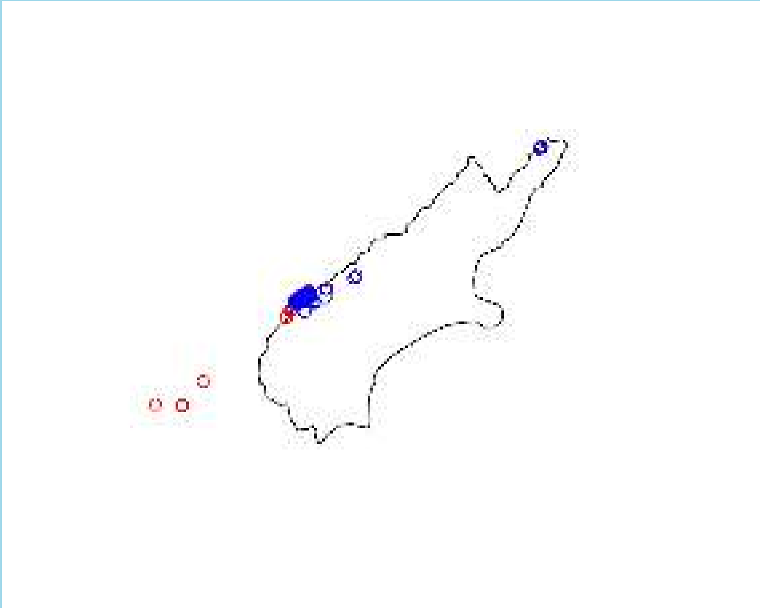
```
# A tibble: 6 × 2
  name_long   continent
  <chr>       <chr>
1 Kazakhstan  Asia
2 Uzbekistan  Asia
3 Indonesia   Asia
4 Timor-Leste Asia
5 Israel      Asia
6 Lebanon     Asia
```

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

```
canterbury = nz  %>% filter(Name == "Cante
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`

```
co = filter(nz, grepl("Canter|Otag
st_intersects(nz_height, co,
                   sparse = TRUE)[1:3]
```

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

[[2]]
[1] 2

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

```
lengths(st_intersects(nz_height,
                      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`

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

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

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

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

Topological Subsetting

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

New Attributes from Existing Fields

Revisiting the `tidyverse`

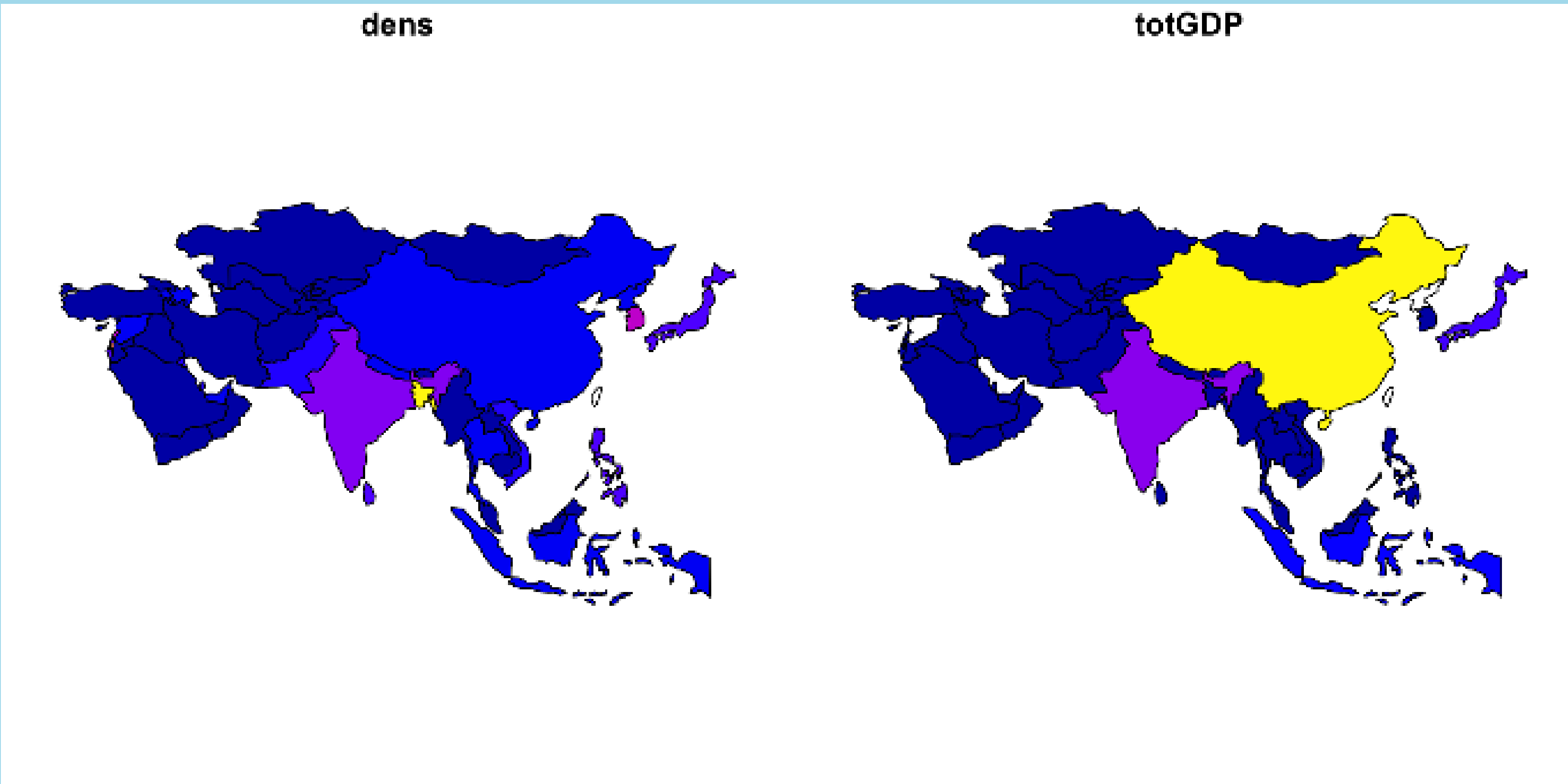
- Creating new fields

```
world %>%
  filter(continent == "Asia") %>%
    dplyr::select(name_long, continent, pop, gdpPercap ,area_km2) %>%
  mutate(., dens = pop/area_km2,
         totGDP = gdpPercap * pop) %>%
  st_drop_geometry() %>%
  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

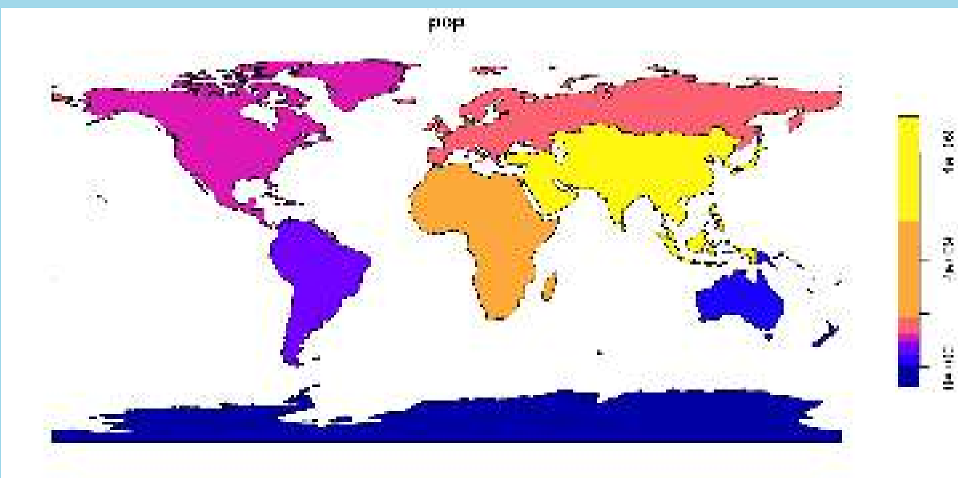


Revisiting the `tidyverse`

- Aggregating data

```r
world %>%
  st_drop_geometry(.) %>%
  group_by(continent) %>%
  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

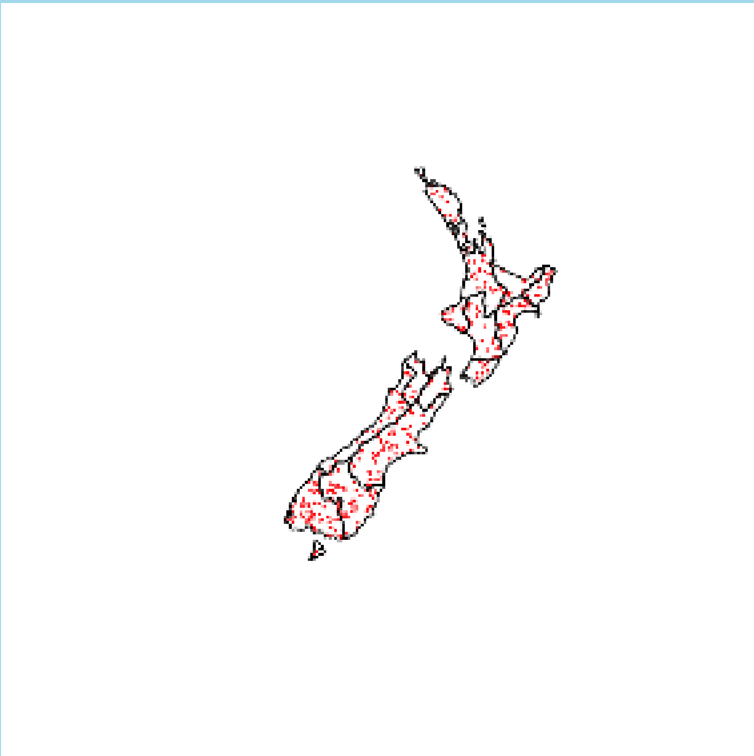
```
nz.sf <- nz %>%
  mutate(area = st_area(nz
head(nz.sf$area, 3)
```

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

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

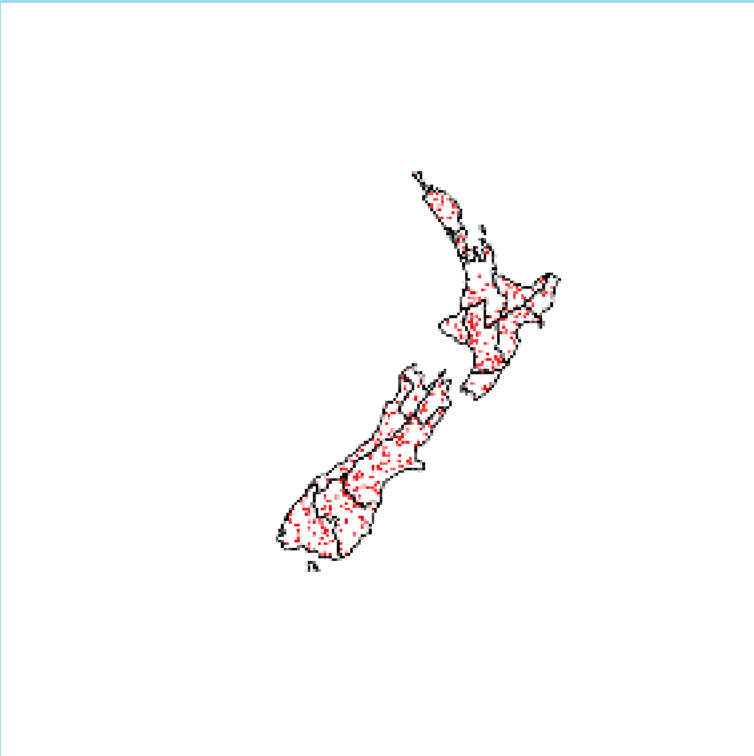
```
Units: [km^2]
[1] 12890.576  4911.565
24588.820
```

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

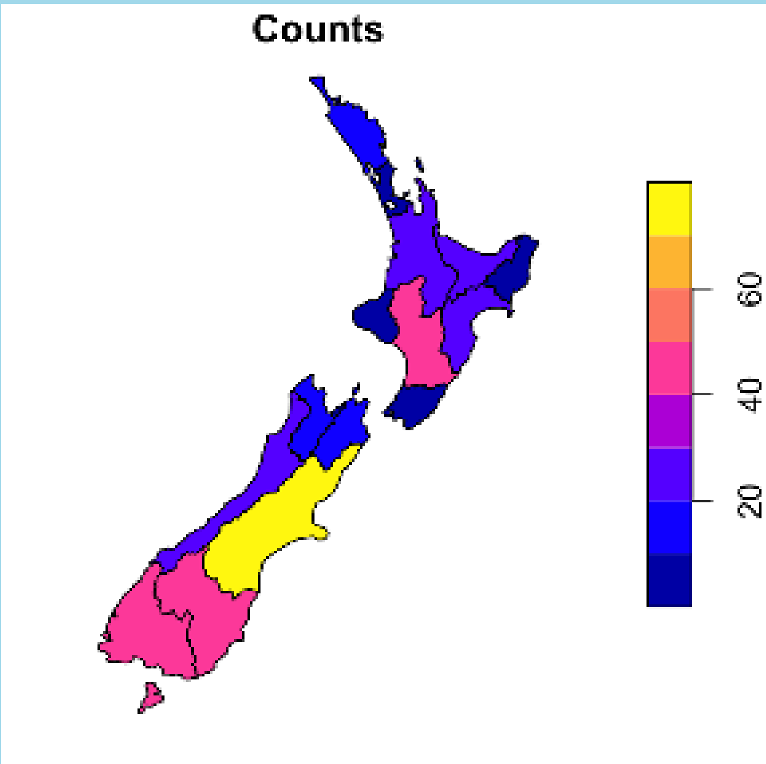
Estimating Density in Polygons

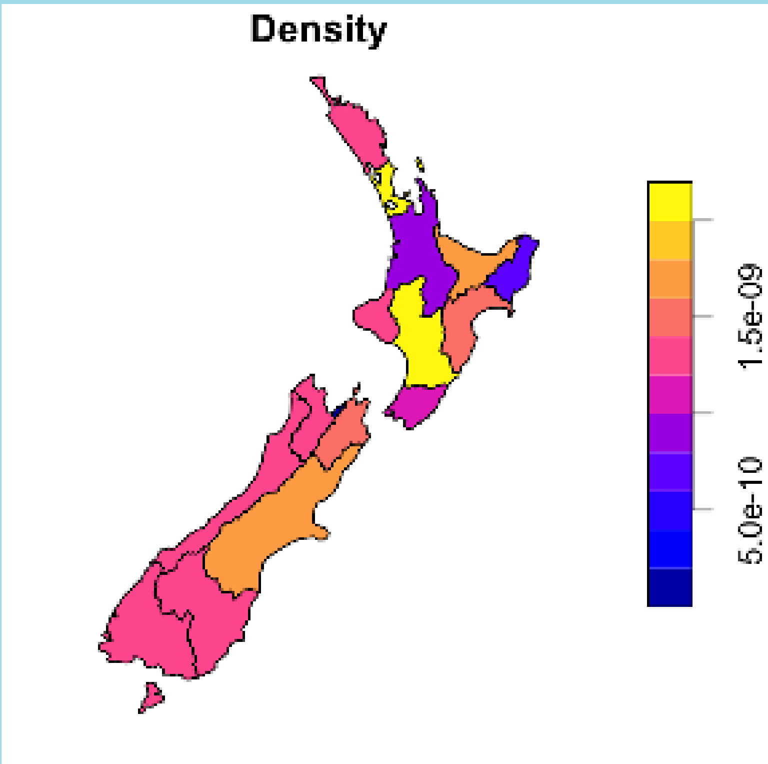


```
nz.df <- nz %>%
mutate(counts = lengths(st_intersects(., r
       area = st_area(nz),
       density = counts/area)
head(st_drop_geometry(nz.df[,7:10]))
```

```
  counts                area
density
1     18 12890576439 [m^2] 1.396369e-09
[1/m^2]
2     10  4911565037 [m^2] 2.036011e-09
[1/m^2]
3     24 24588819863 [m^2] 9.760534e-10
[1/m^2]
4     22 12271015945 [m^2] 1.792843e-09
[1/m^2]
5      6  8364554416 [m^2] 7.173126e-10
[1/m^2]
6     21 14242517871 [m^2] 1.474458e-09
[1/m^2]
```

Estimating Density in Polygons



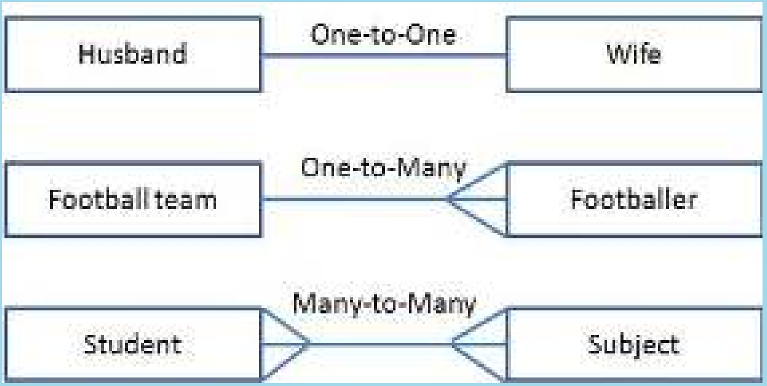


Estimating Distance

- 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

```
canterbury = nz %>% filter(Name == "Canterbury")
canterbury_height = nz_height[canterbury, ]
co = filter(nz, grepl("Canter|Otag", Name))
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
```

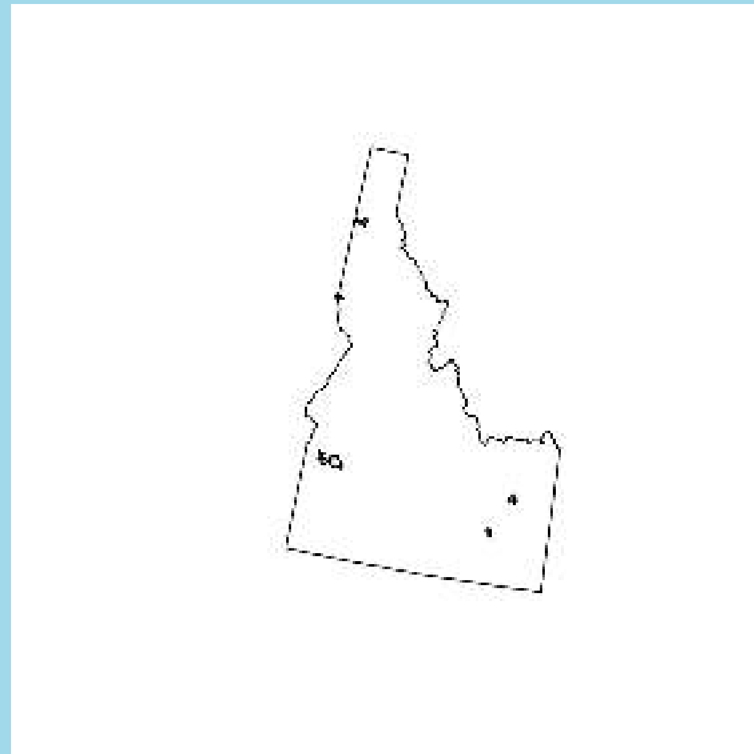
Estimating Single Point Distance

- Using nearest neighbor distances

```
ua <- urban_areas(cb = FALSE, progress_bar
  filter(., UATYP10 == "U") %>%
  filter(., str_detect(NAME10, "ID")) %>%
  st_transform(., crs=2163)

#get index of nearest ID city
nearest <-  st_nearest_feature(ua)
#estimate distance
(dist = st_distance(ua, ua[nearest,], by_e
```

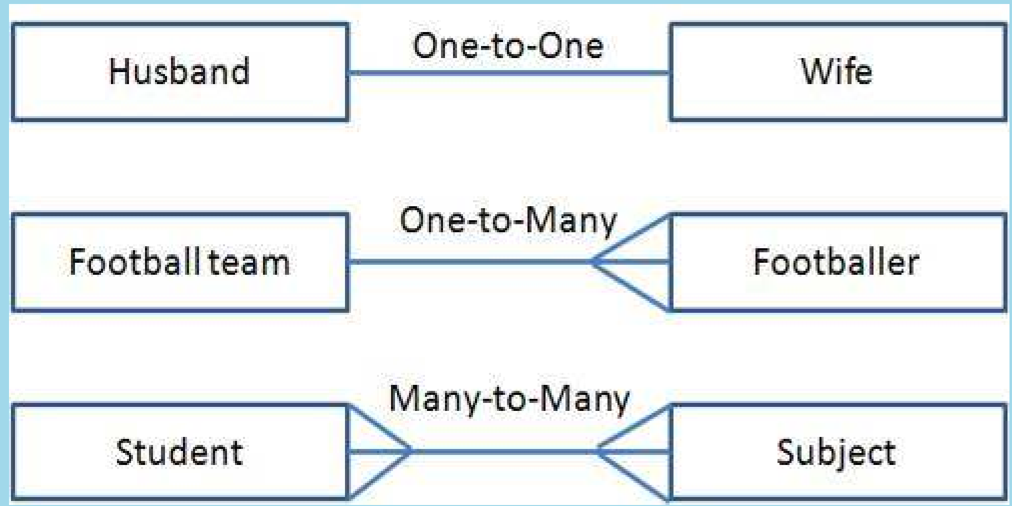
```
Units: [m]
[1]  61386.444  61386.444   1646.182
1646.182 136908.183 136908.183
```



Joining (a)spatial data

Joining (a)spatial data

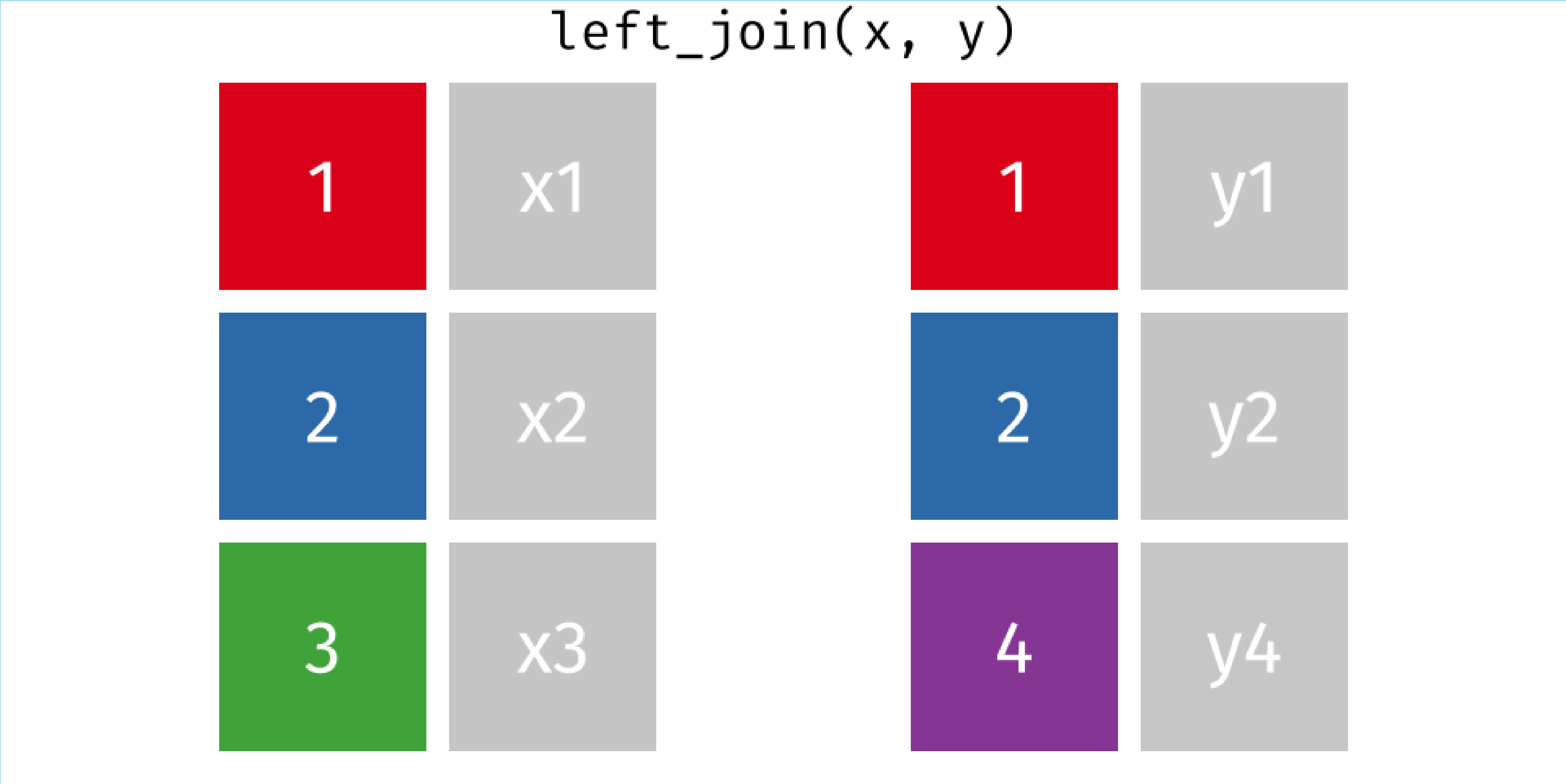
- Requires a “key” field
- Multiple outcomes possible
- Think about your final data form



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

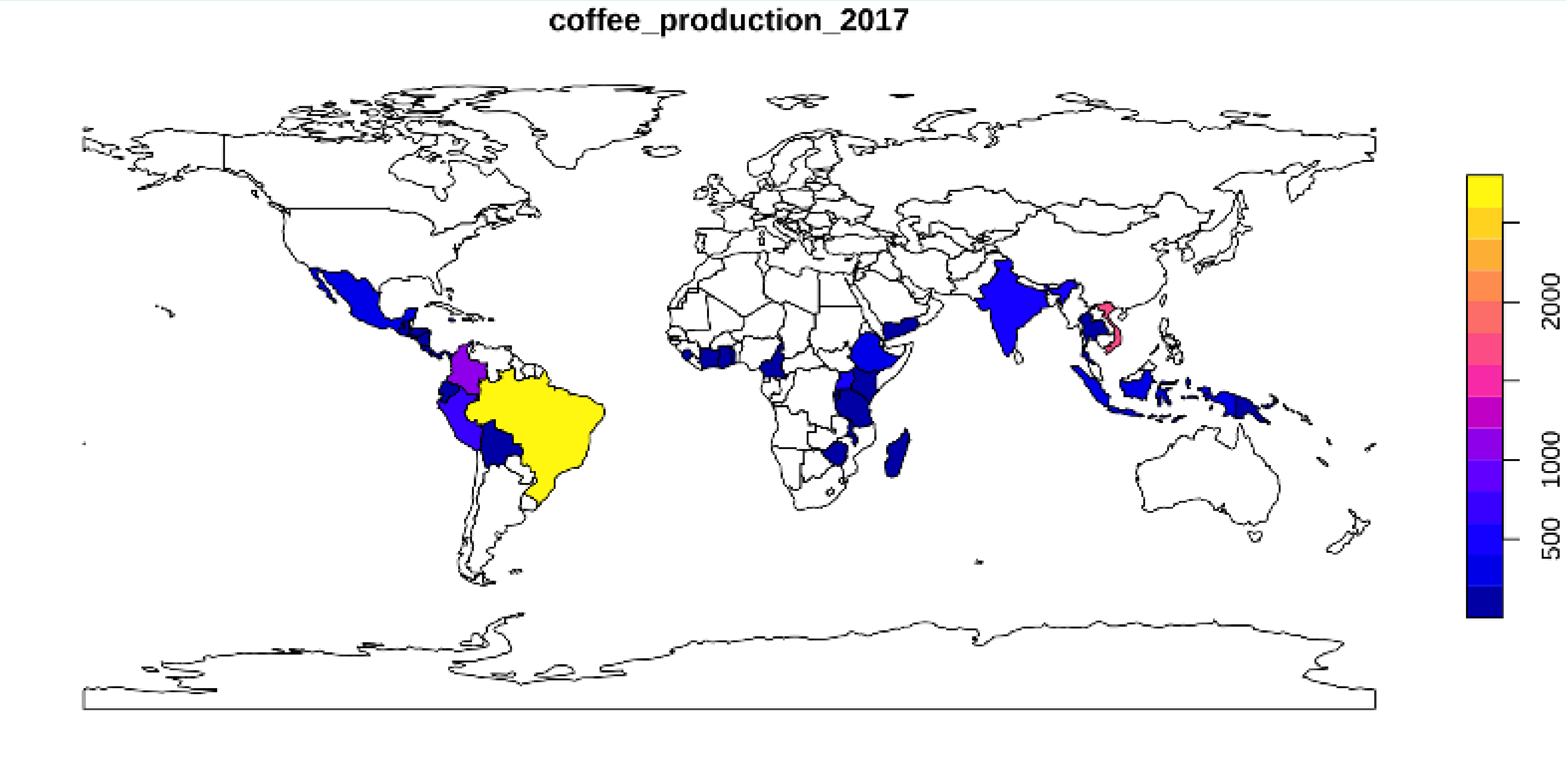
```
head(coffee_data)
```

```
# 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
```

```
world_coffee = left_join(world, co
nrow(world_coffee)
```

```
[1] 177
```

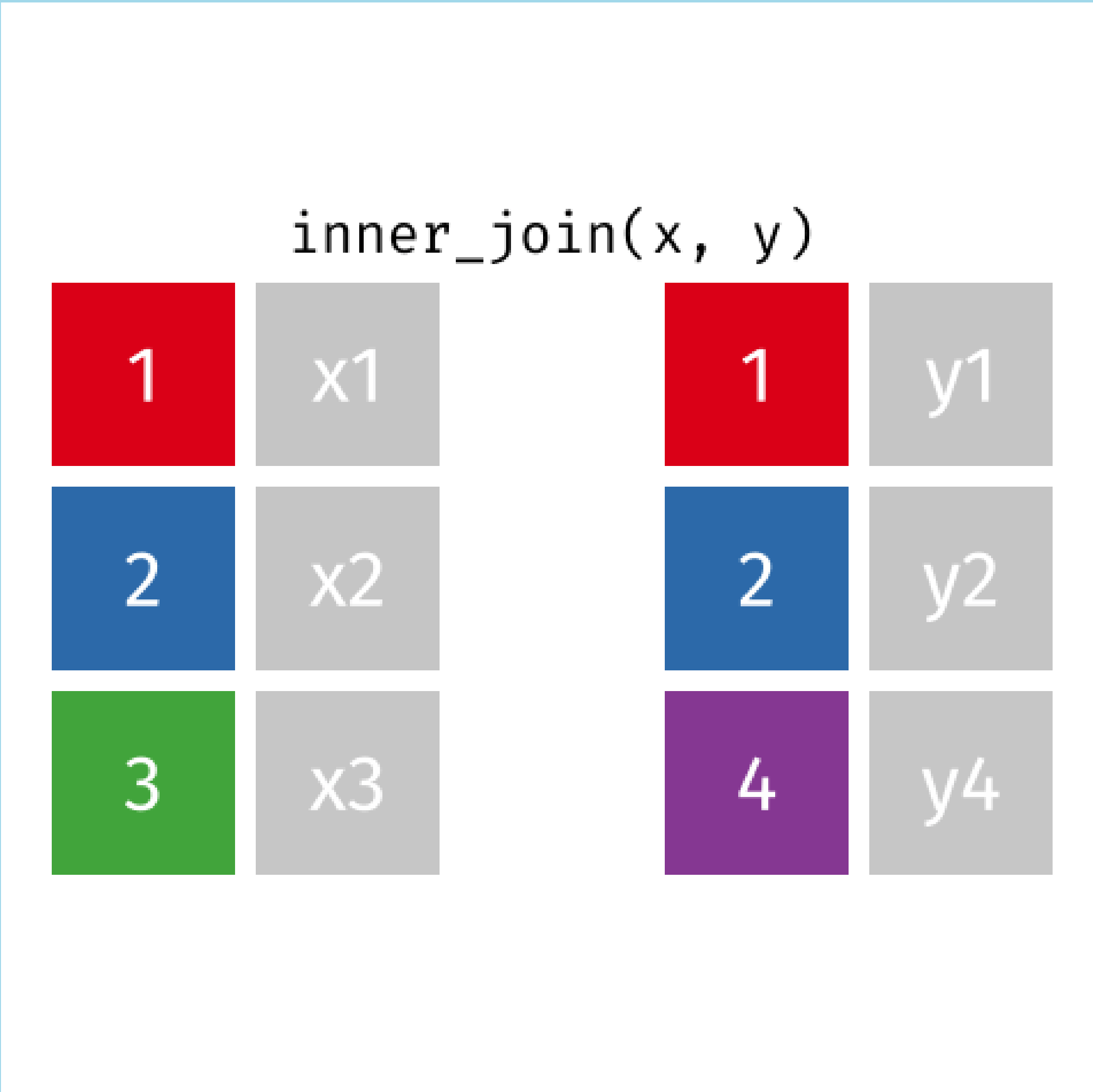
Left Join



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

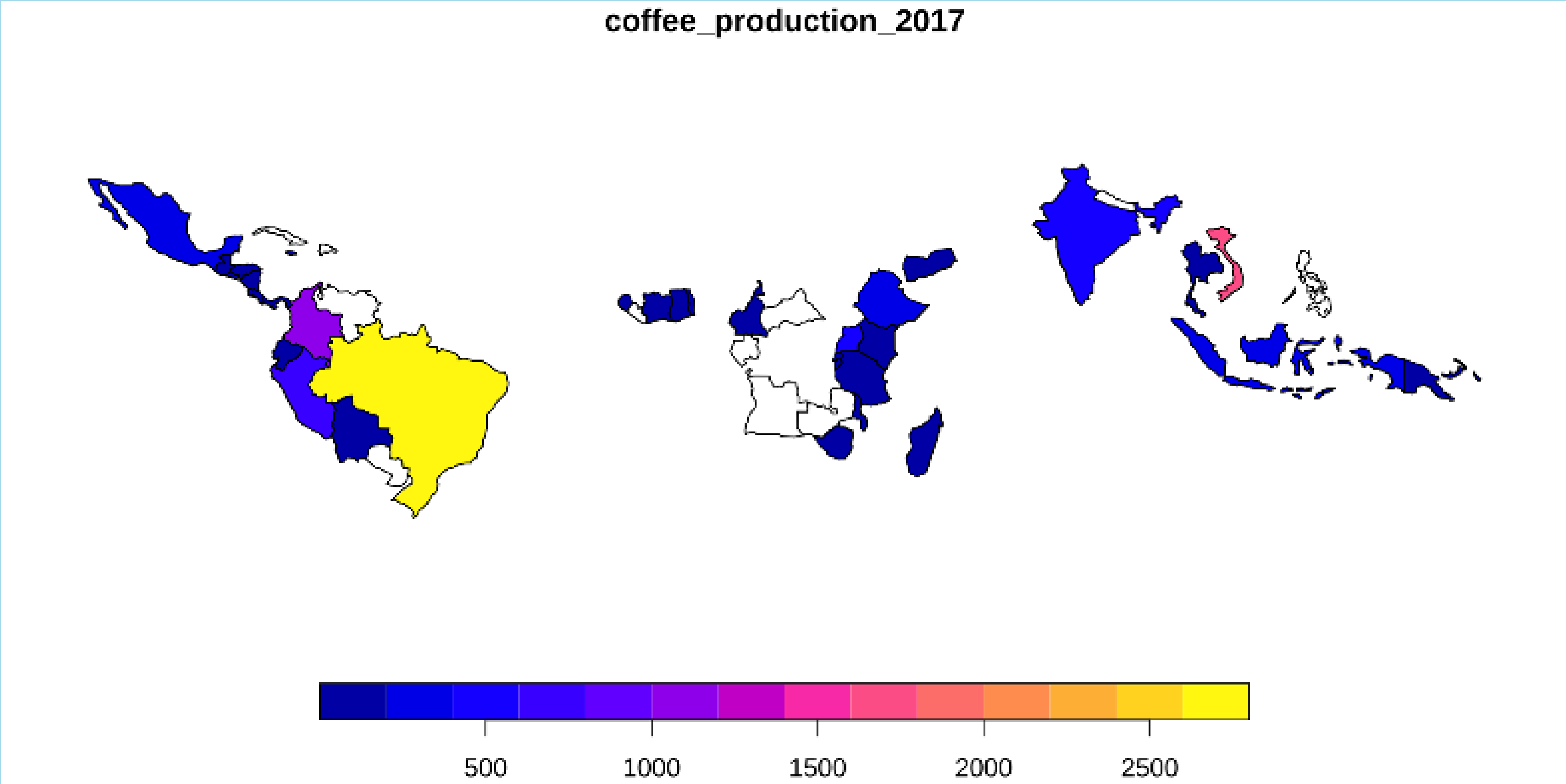
```
world_coffee_inner = inner_join(wo
nrow(world_coffee_inner)
```

```
[1] 45
```

```
setdiff(coffee_data$name_long, wor
```

```
[1] "Congo, Dem. Rep. of" "Others"
```

Inner Join



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

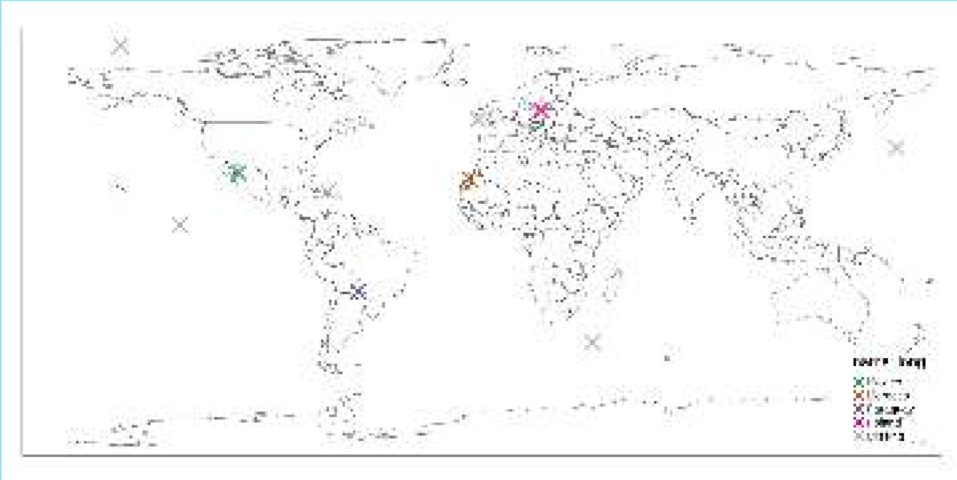
Spatial Joins

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

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

```
#>    xmin    ymin    xmax    ymax
#> -180.0   -89.9   180.0    83.6
random_df = data.frame(
  x = runif(n = 10, min = bb[1], m
  y = runif(n = 10, min = bb[2], m
)
random_points = random_df |>
  st_as_sf(coords = c("x", "y")) |
  st_set_crs("EPSG:4326") # set ge

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

```
any(st_touches(cycle_hire, cycle_hire_osm, sparse
```

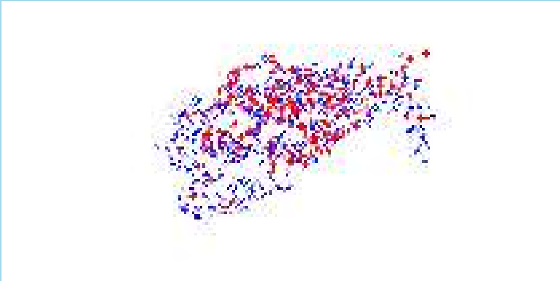
```
[1] FALSE
```

```
z = st_join(cycle_hire, cycle_hire_osm, st_is_with
nrow(cycle_hire)
```

```
[1] 742
```

```
nrow(z)
```

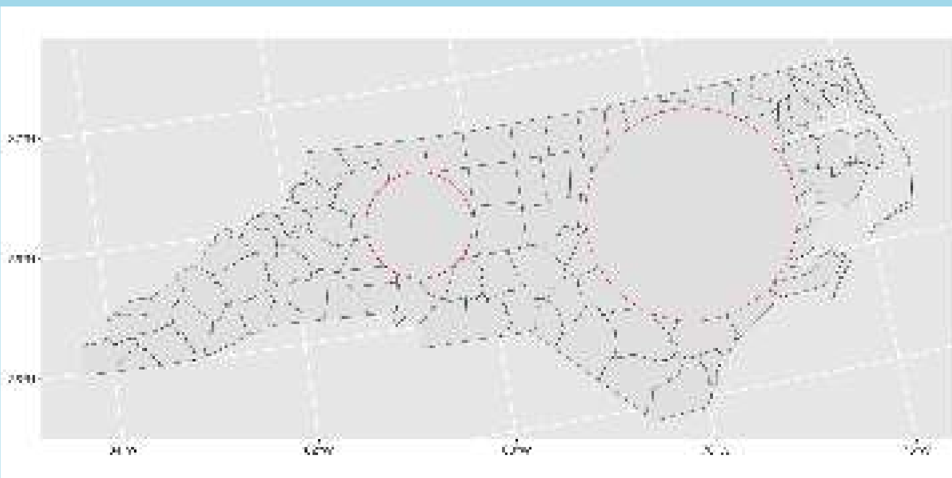
```
[1] 762
```



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 with a service provision area?
- `st_intersection`, `st_union`, and `st_difference` return new geometries that we can use as records in our spatial database



```
intersect_pct <- st_intersection(n
   mutate(intersect_area = st_area
   dplyr::select(NAME, intersect_a
   st_drop_geometry()

nc <- mutate(nc, county_area = st_

# Merge by county name
nc <- merge(nc, intersect_pct, by

# Calculate coverage
nc <- nc %>%
   mutate(coverage = as.numeric(in
```

Extending Joins