Introduction to Spatial Data Programming - R

Lesson 08
Geometric operations with vector layers

Michael Dorman
Geography and Environmental Development
dorman@post.bgu.ac.il

Last updated: 2019-01-14 08:33:04

Contents

- Join -
  - By attribute
  - By location
- Spatial operators -
  - Area
  - Distance
  - Centroid
  - Union
  - Buffer
  - Intersection
  - Aggregation
  - Convex hull

Aim

- Learn to join by location and by attributes
- Learn to make geometric calculations between vector layers

Preparation

```
library(sf)
library(units)
library(dplyr)

setwd("~/Dropbox/Data2")
```

US counties and three airports

- Let's load the US **counties** and three **airports** layers (see **Lesson 07**) -

```
airports = st_read(
  "airports.geojson",
  stringsAsFactors = FALSE
)
county = st_read(
  "USA_2_GADM_fips.shp",
  stringsAsFactors = FALSE
)
```

- And **transform** them to the US National Atlas projection -

```
airports = st_transform(airports, 2163)
county = st_transform(county, 2163)
```

Join by location

- Join by spatial location, or **spatial join**, is one of the most common operations in spatial analysis
- In a spatial join we are "attaching" attributes from one layer to another based on their **spatial relations**
- In ArcGIS this is done using "Join data from another layer based on spatial location"

Join by location

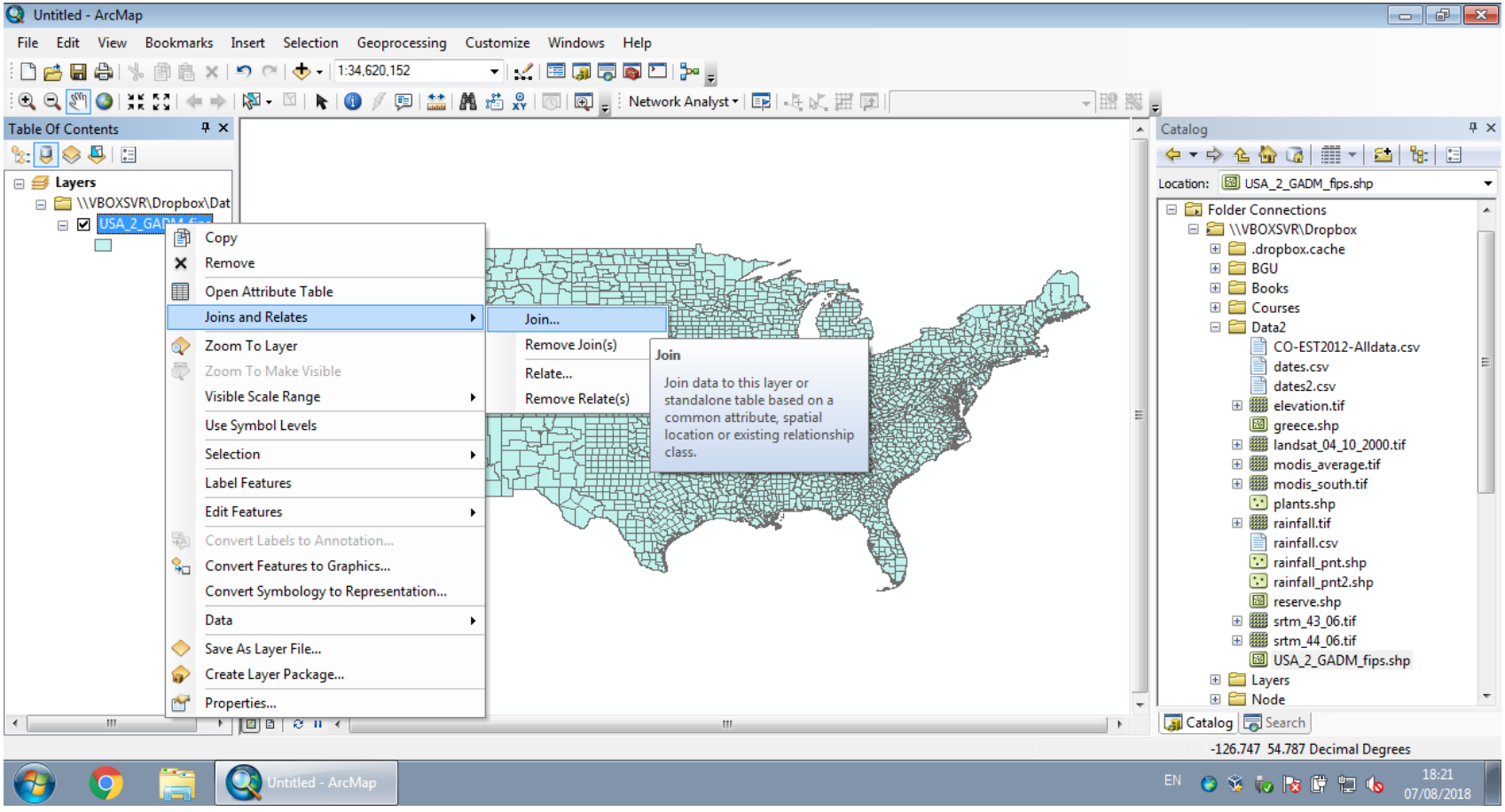

Figure 1: Join by location in ArcGIS: step 1

Join by location

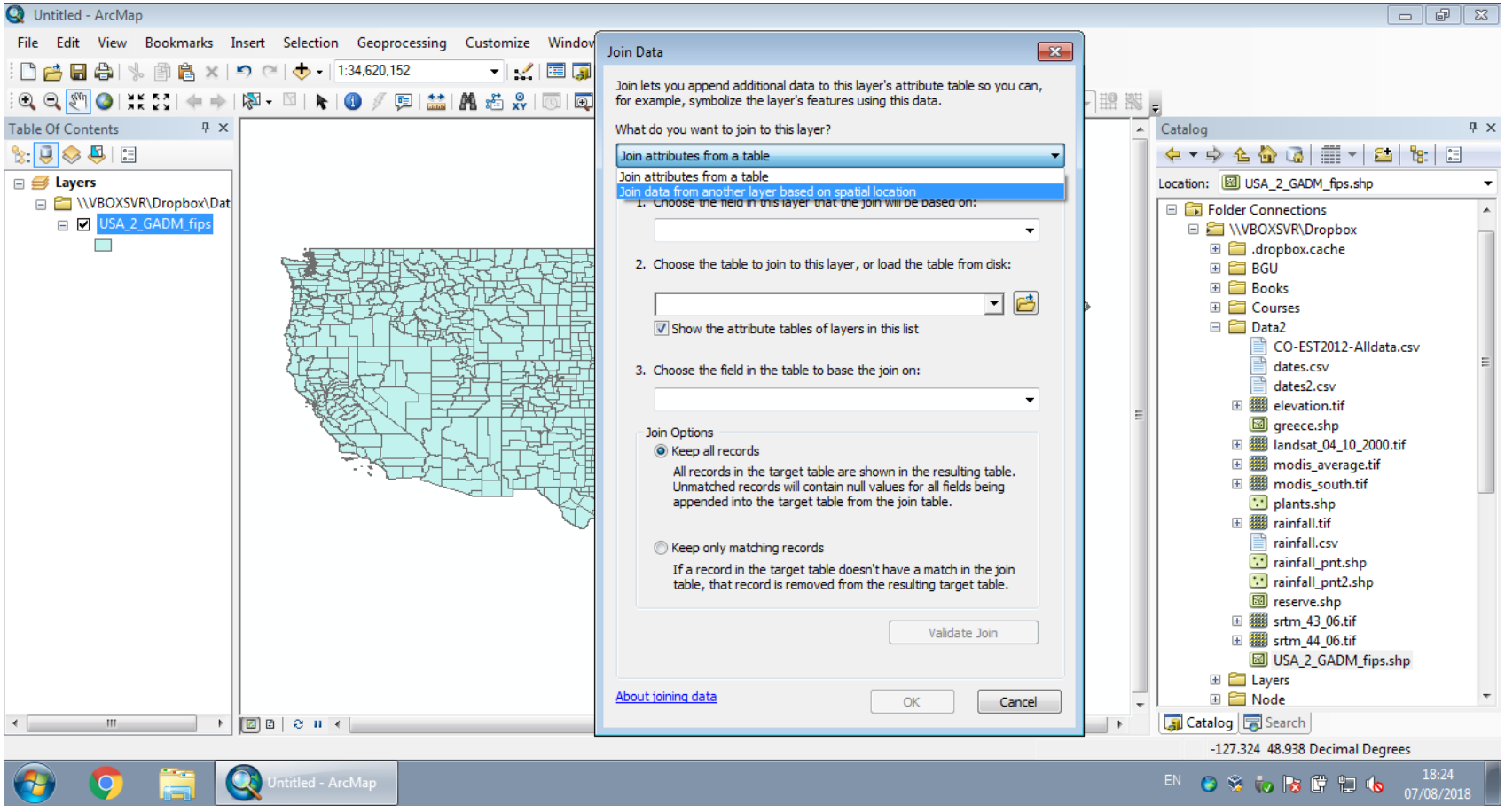

Figure 2: Join by location in ArcGIS: step 2

Join by location

- For simplicity, we will create a subset of the counties of New-Mexico only -

```
nm = county[county$NAME_1 == "New Mexico", ]
```

- Plot -

```
plot(st_geometry(nm))
plot(
  st_geometry(airports),
  col = "red",
  pch = 16,
  cex = 2,
  add = TRUE
)
```

- Note: `pch` = point shape, `cex` = point size (see `?points`)

Join by location

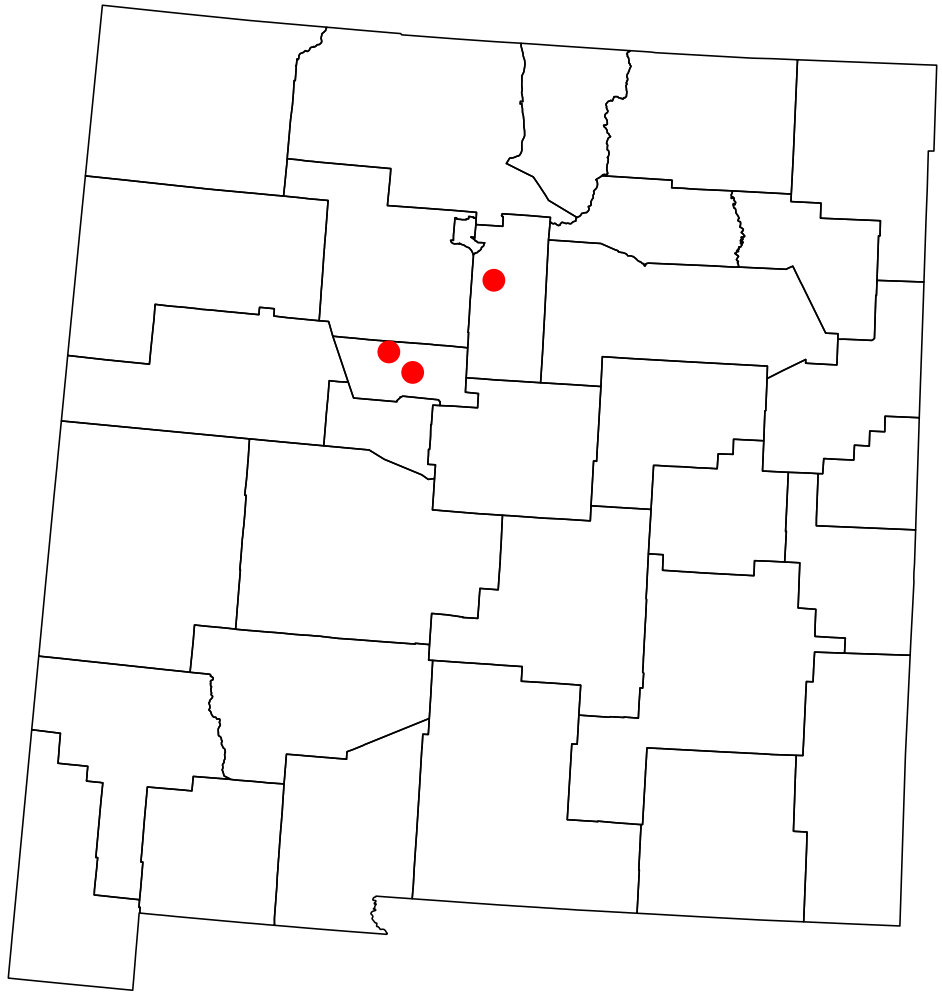

Figure 3: The `nm` and `airports` layers

Join by location

- Given two layers `x` and `y`, a function call of the form `st_join(x, y)` returns the `x` layer along with **matching attributes** from `y`
- Note: the default is join by **intersection**, other options possible (see `join` parameter in `?st_join`)
- For example, here is how we join the `airports` layer with the matching **county attributes** -

```
st_join(airports, nm)
```

```
## Simple feature collection with 3 features and 5 fields
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: -619184.5 ymin: -1081605 xmax: -551689.8 ymax: -102236
## epsg (SRID):    2163
## proj4string:    +proj=laea +lat_0=45 +lon_0=-100 +x_0=0 +y_0=0 +a=6370997 +b
##                          name     NAME_1     NAME_2 TYPE_2  FIPS
## 1 Albuquerque International New Mexico Bernalillo County 35001 POINT (-60386
## 2           Double Eagle II New Mexico Bernalillo County 35001 POINT (-61918
## 3        Santa Fe Municipal New Mexico   Santa Fe County 35049 POINT (-55168
```

Join by location

- Question: what do you think happens in the following case? How many features does the result have, and why?

```
st_join(nm, airports)
```

```
## Simple feature collection with 34 features and 5 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -863763.9 ymin: -1478601 xmax: -267074.1 ymax: -845634
## epsg (SRID):    2163
## proj4string:    +proj=laea +lat_0=45 +lon_0=-100 +x_0=0 +y_0=0 +a=6370997 +b
## First 10 features:
##          NAME_1     NAME_2 TYPE_2  FIPS               name
## 1800 New Mexico      Union County 35059               <NA> MULTIPOLYGON (((-
## 1801 New Mexico   San Juan County 35045               <NA> MULTIPOLYGON (((-
## 1802 New Mexico Rio Arriba County 35039               <NA> MULTIPOLYGON (((-
## 1803 New Mexico       Taos County 35055               <NA> MULTIPOLYGON (((-
## 1804 New Mexico     Colfax County 35007               <NA> MULTIPOLYGON (((-
## 1826 New Mexico       Mora County 35033               <NA> MULTIPOLYGON (((-
## 1832 New Mexico    Harding County 35021               <NA> MULTIPOLYGON (((-
## 1833 New Mexico   Sandoval County 35043               <NA> MULTIPOLYGON (((-
## 1839 New Mexico   Santa Fe County 35049 Santa Fe Municipal MULTIPOLYGON (((-
## 1840 New Mexico   McKinley County 35031               <NA> MULTIPOLYGON (((-
```

Subsetting by location

- We can create **subsets** based on intersection with **another layer** using the `[` operator
- An expression of the form `x[y, ]` returns a subset of `x` features that **intersect** `y`
- For example -

```
nm1 = nm[airports, ]
```

- Plot -

```
plot(st_geometry(nm))
plot(st_geometry(nm1), col = "lightblue", add = TRUE)
plot(st_geometry(airports),
  col = "red", pch = 16, cex = 2, add = TRUE
)
```

- Question 1: what will be the result of `airports[nm, ]`?
- Question 2: what will be the result of `nm[nm[20, ], ]`?

Subsetting by location

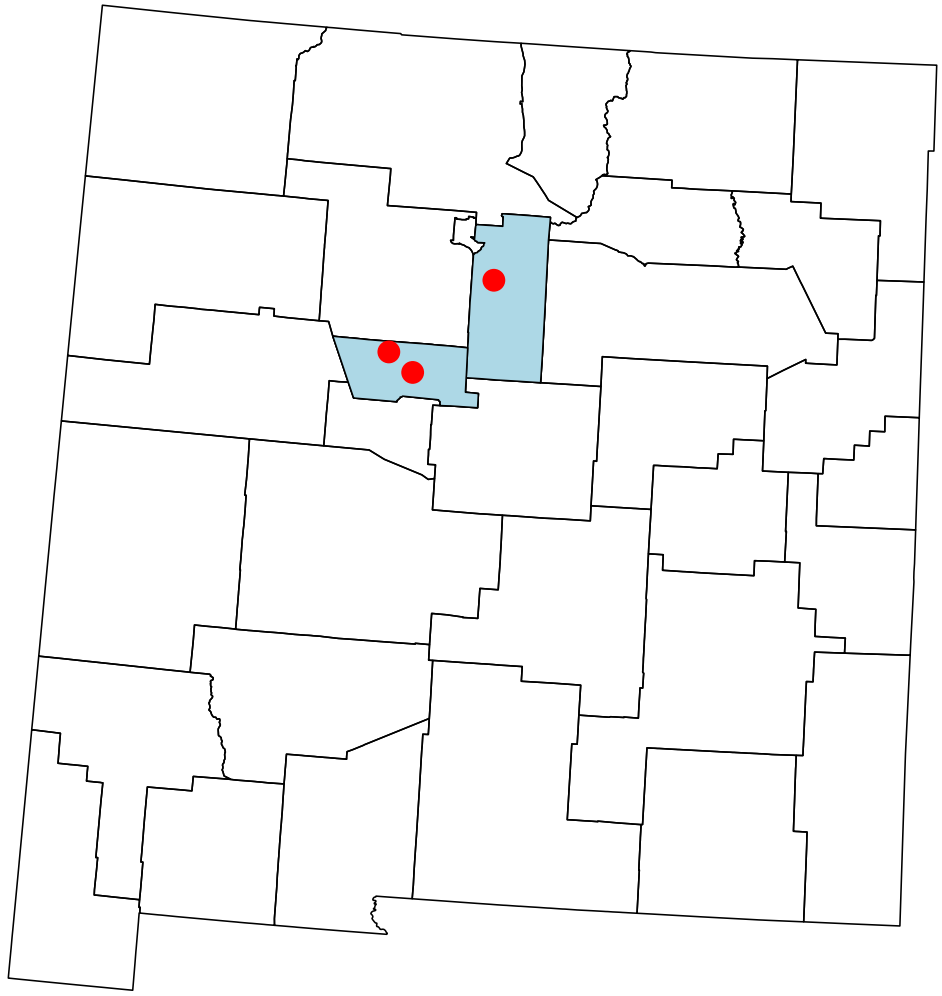

Figure 4: Subset of the `nm` layer based on intersection with `airports`

Geometric calculations

Geometric operations on vector layers can conceptually be divided into **three groups** according to their output -

- **Numeric** values: Functions that summarize geometrical properties of -
  - A **single layer** (e.g. area, length)
  - A **pair of layers** (e.g. distance)
- **Logical** values: Functions that evaluate whether a certain condition holds true, regarding -
  - A **single layer** (e.g. geometry is valid)
  - A **pair of layers** (e.g. feature A intersects feature B)
- **Spatial** layers: Functions that create a new layer based on -
  - A **single layer** (e.g. centroids)
  - A **pair of layers** (e.g. intersection area)

Numeric

- There are several functions to calculate **numeric geometric properties** of vector layers in package `sf` -
  - `st_length`
  - `st_area`
  - `st_distance`
  - `st_bbox`
  - `st_dimension`

Numeric

- For example, we can calculate the **area** of each feature in the `states` layer (i.e. each state) using `st_area` -

```
county$area = st_area(county)
county$area[1:3]
## Units: m^2
## [1] 2451875694 1941109814 1077788608
```

- The **result** is an object of class `units` -

```
class(county$area)
## [1] "units"
```

- **CRS units** (e.g. meters) are used by default

Numeric

- We can convert measurements an **ordinary** `numeric` vector with `as.numeric` -

```
as.numeric(county$area[1:3])
## [1] 2451875694 1941109814 1077788608
```

- We can convert measurements to **different units** with `set_units` from package `units` (see `?valid_udunits`) -

```
county$area = set_units(county$area, "km^2")
county$area[1:3]
## Units: km^2
## [1] 2451.876 1941.110 1077.789
```

- Inspecting the new `"area"` column -

```
plot(county[, "area"])
```

Numeric

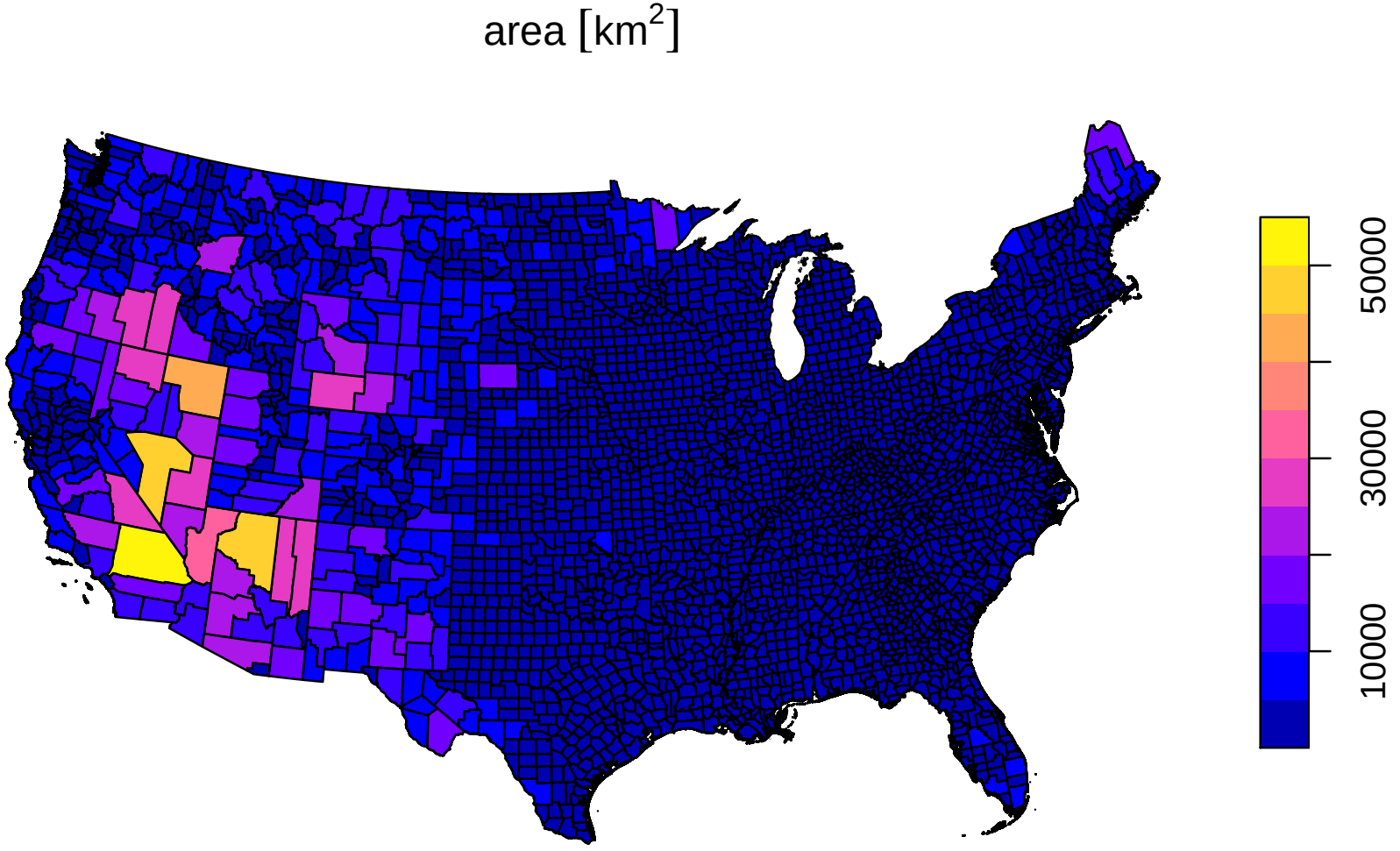

Figure 5: Calculated `area` attribute

Numeric

- An example of a numeric operator on a **pair** of geometries is **geographical distance**
- Distances can be calculated using function `st_distance` -

```
d = st_distance(airports, nm)
```

Numeric

- The result is a **matrix** of `units` values -

```
d[, 1:4]
## Units: m
##             [,1]      [,2]      [,3]      [,4]
## [1,] 266778.9 140470.5 103328.22 140913.62
## [2,] 275925.0 120879.9  93753.25 141511.80
## [3,] 197540.4 145580.9  34956.53  62231.01
```

- In the **distance matrix** -
  - **rows** refer to features of `x`
  - **columns** refer to features of `y`

```
dim(d)
## [1]  3 33
```

Numeric

- Just like areas, distances can always be converted to different **units** with `set_units` -

```
d = set_units(d, "km")
```

```
d[, 1:4]
## Units: km
##           [,1]     [,2]      [,3]      [,4]
## [1,] 266.7789 140.4705 103.32822 140.91362
## [2,] 275.9250 120.8799  93.75325 141.51180
## [3,] 197.5404 145.5809  34.95653  62.23101
```

Numeric

- To work with the distance `matrix`, it can be convenient to set row and column **names** -

```
rownames(d) = airports$name
colnames(d) = nm$NAME_2
```

```
d[1:3, 1:2]
## Units: km
##                                  Union San Juan
## Albuquerque International 266.7789 140.4705
## Double Eagle II           275.9250 120.8799
## Santa Fe Municipal        197.5404 145.5809
```

Numeric

- When row and column names are set, it is more convenient to find out the distance between a **specific** airport and a **specific** county -

```
d["Santa Fe Municipal", "Santa Fe", drop = FALSE]
## Units: km
##                          Santa Fe
## Santa Fe Municipal             0
```

Logical

- Given two layers, `x` and `y`, the following **logical geometric functions** check whether each feature in `x` maintains the specified **relation** with each feature in `y` -
  - `st_intersects`
  - `st_disjoint`
  - `st_touches`
  - `st_crosses`
  - `st_within`
  - `st_contains`
  - `st_overlaps`
  - `st_covers`
  - `st_covered_by`
  - `st_equals`
  - `st_equals_exact`

Logical

- When specifying `sparse=FALSE` the functions return a **logical** `matrix`
- Each **element** `i,j` in the matrix is `TRUE` when `f(x[i], y[j])` is `TRUE`
- For example, this creates a matrix of **intersection** relations between counties -

```
int = st_intersects(nm, nm, sparse = FALSE)
```

```
int[1:4, 1:4]
##       [,1]  [,2]  [,3]  [,4]
## [1,]  TRUE FALSE FALSE FALSE
## [2,] FALSE  TRUE  TRUE FALSE
## [3,] FALSE  TRUE  TRUE  TRUE
## [4,] FALSE FALSE  TRUE  TRUE
```

Logical

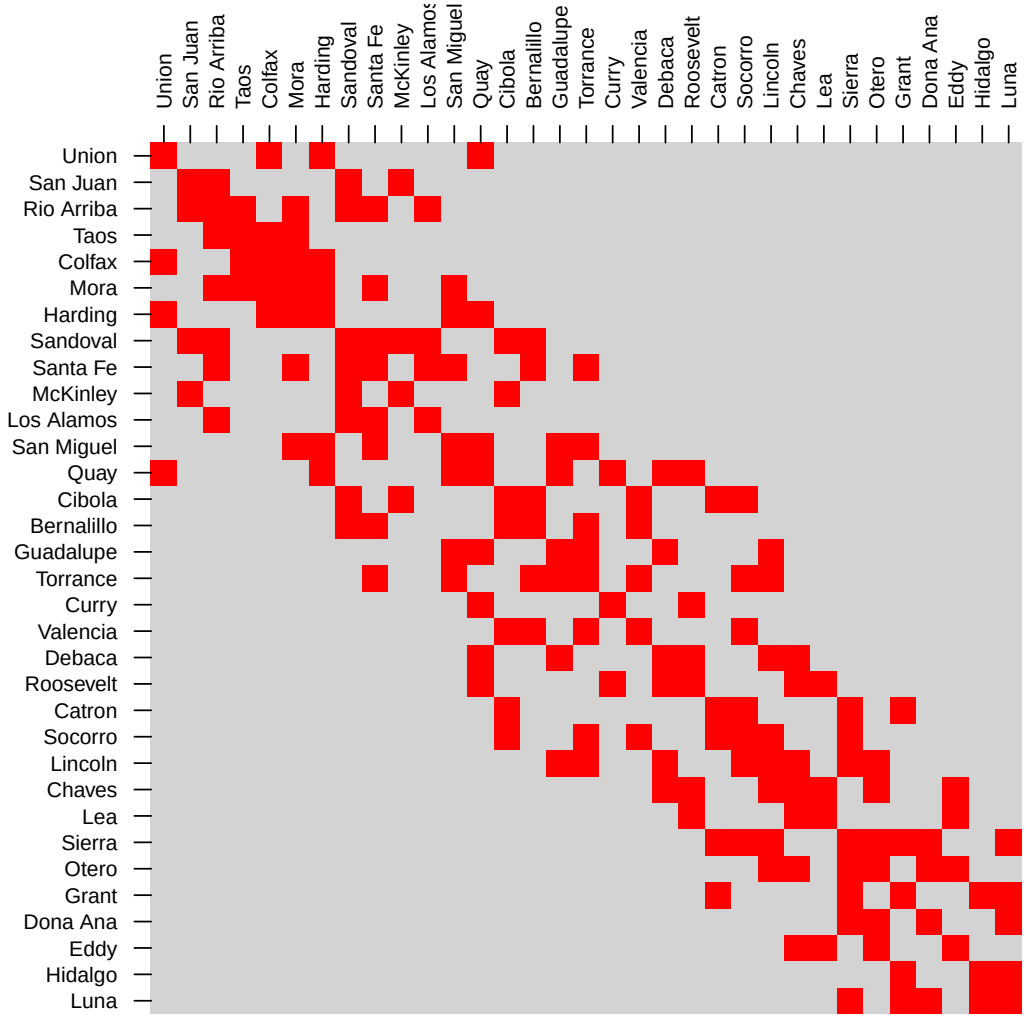

Figure 6: Intersection between counties in New Mexico (`nm`)

Logical

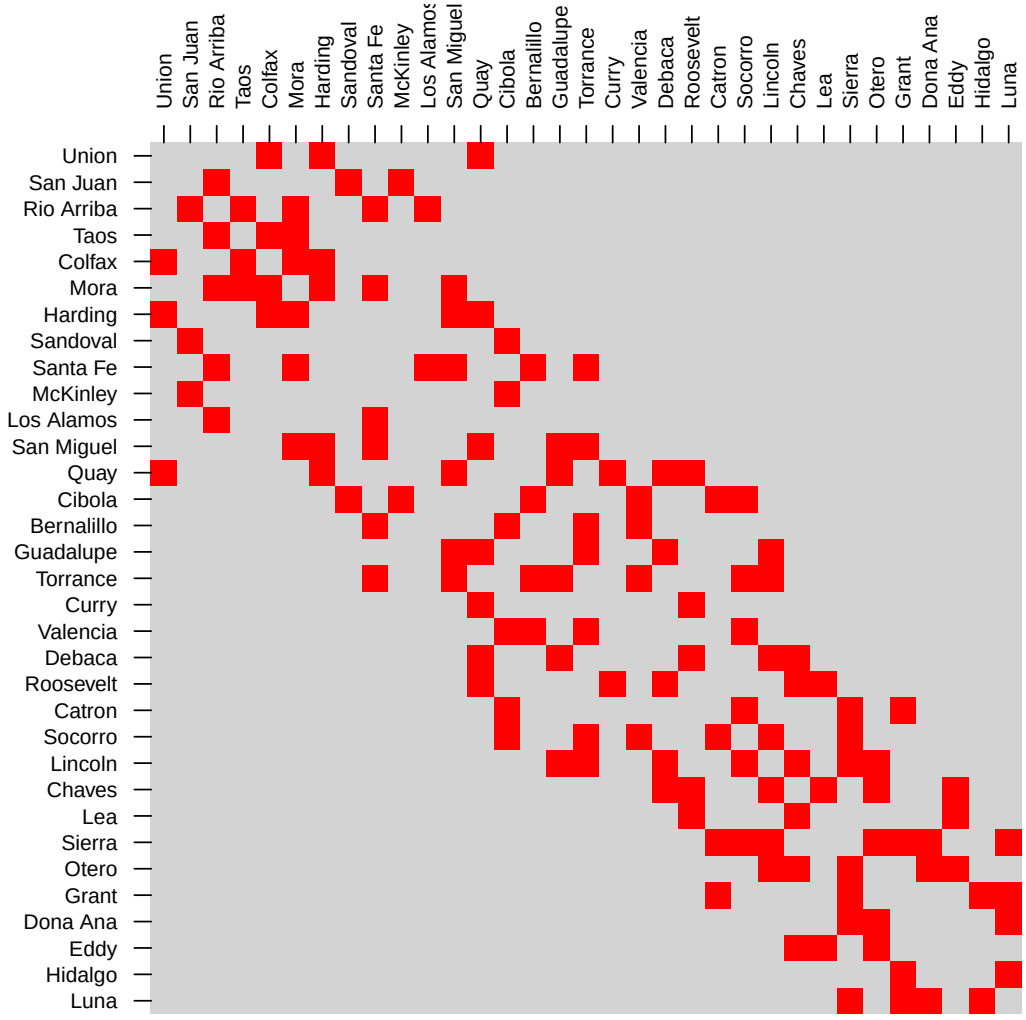

Figure 7: Using `st_touches` instead of `st_intersects`

Logical

- Question: How can we calculate `airports` count per county in `nm`, using `st_intersects`?

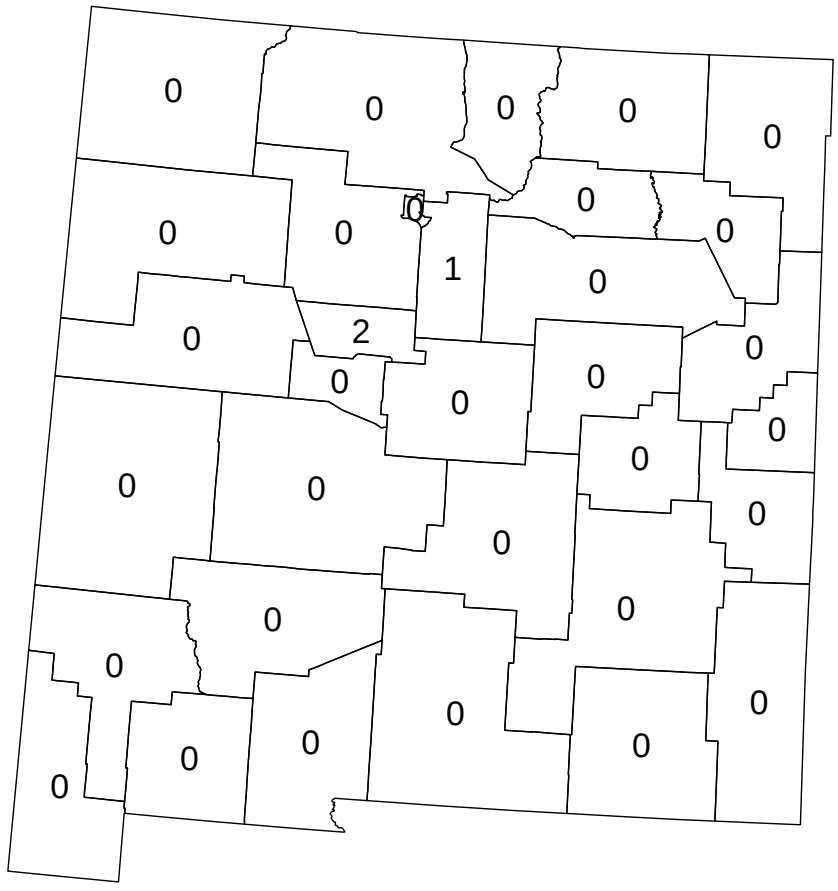

Figure 8: Airport count per county in New Mexico

Spatial

- `sf` provides common **geometry-generating** functions applicable to **individual** geometries, such as -
  - `st_centroid`
  - `st_buffer`
  - `st_sample`
  - `st_convex_hull`
  - `st_voronoi`

Spatial

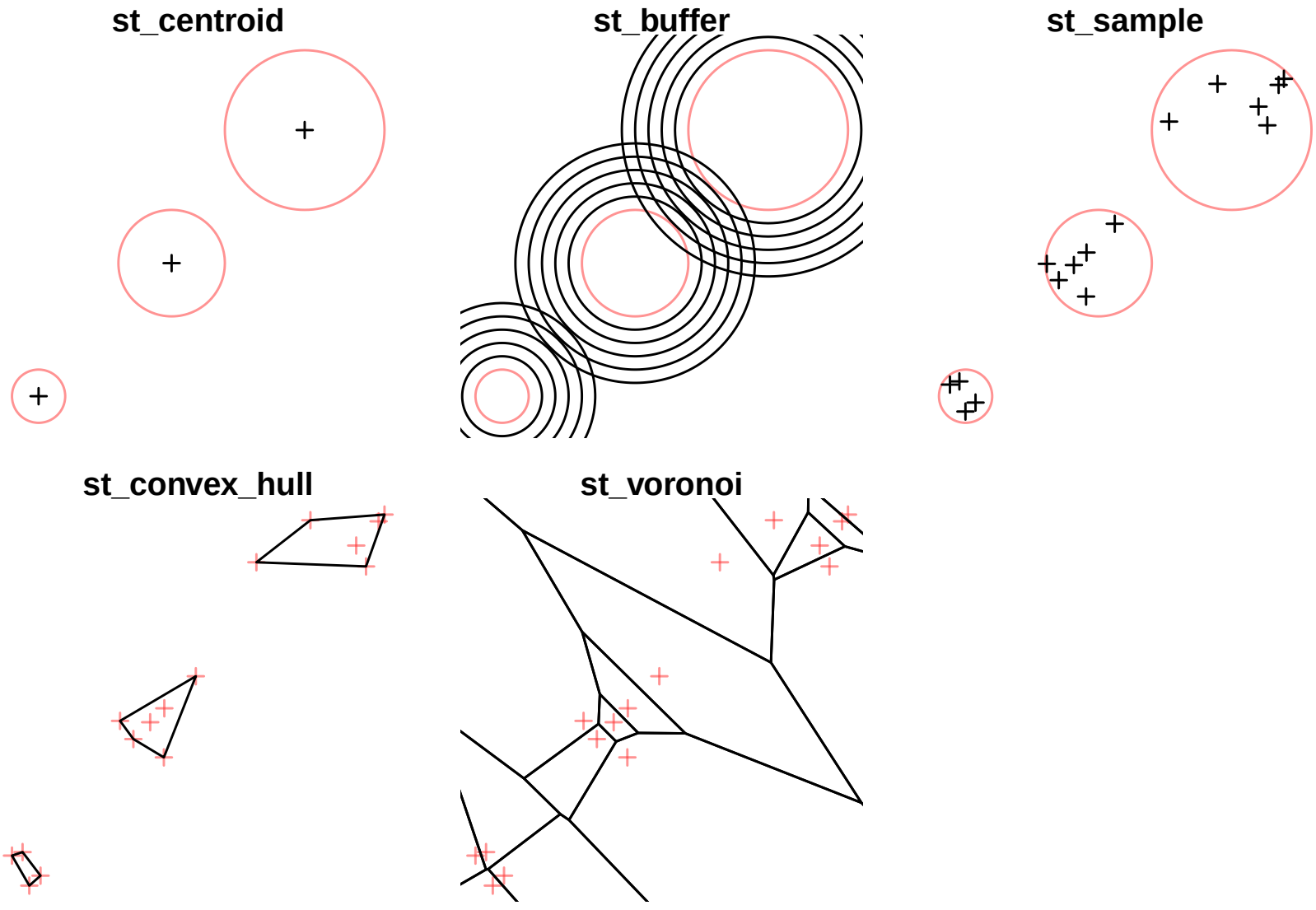

Figure 9: Geometry-generating operations on individual layers

Spatial

- For example, the following expression uses `st_centroid` to create a layer of **state centroids** -

```
ctr = st_centroid(nm)
```

- Plot -

```
plot(
  st_geometry(nm),
  border = "grey"
)
plot(
  st_geometry(ctr),
  col = "red", pch = 3,
  add = TRUE
)
```

Spatial

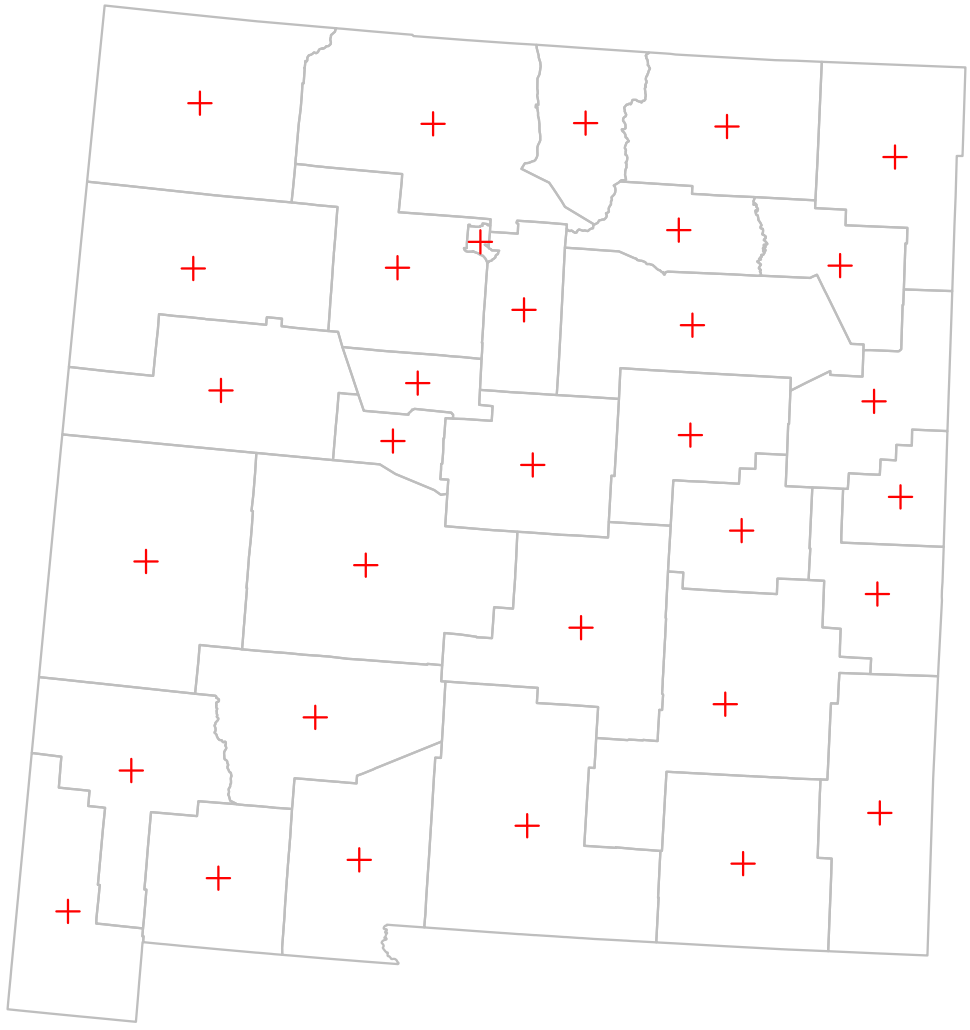

Figure 10: Centroids of New Mexico counties

Spatial

- What is the distance between the **centroids** of California and New Jersey?

```
ca = county[county$NAME_1 == "California", ]
nj = county[county$NAME_1 == "New Jersey", ]

ca_ctr = st_centroid(st_union(ca))
nj_ctr = st_centroid(st_union(nj))

d = st_distance(ca_ctr, nj_ctr)

set_units(d, "km")
## Units: km
##          [,1]
## [1,] 3846.64
```

- Note: we are using `st_union` to combine the counties into a single geometry and to "dissolve" the borders

Spatial

- Plot -

```
plot(
  st_geometry(county),
  border = "grey"
)
plot(
  c(ca_ctr, nj_ctr),
  col = "red", pch = 16, cex = 2,
  add = TRUE
)
```

Spatial

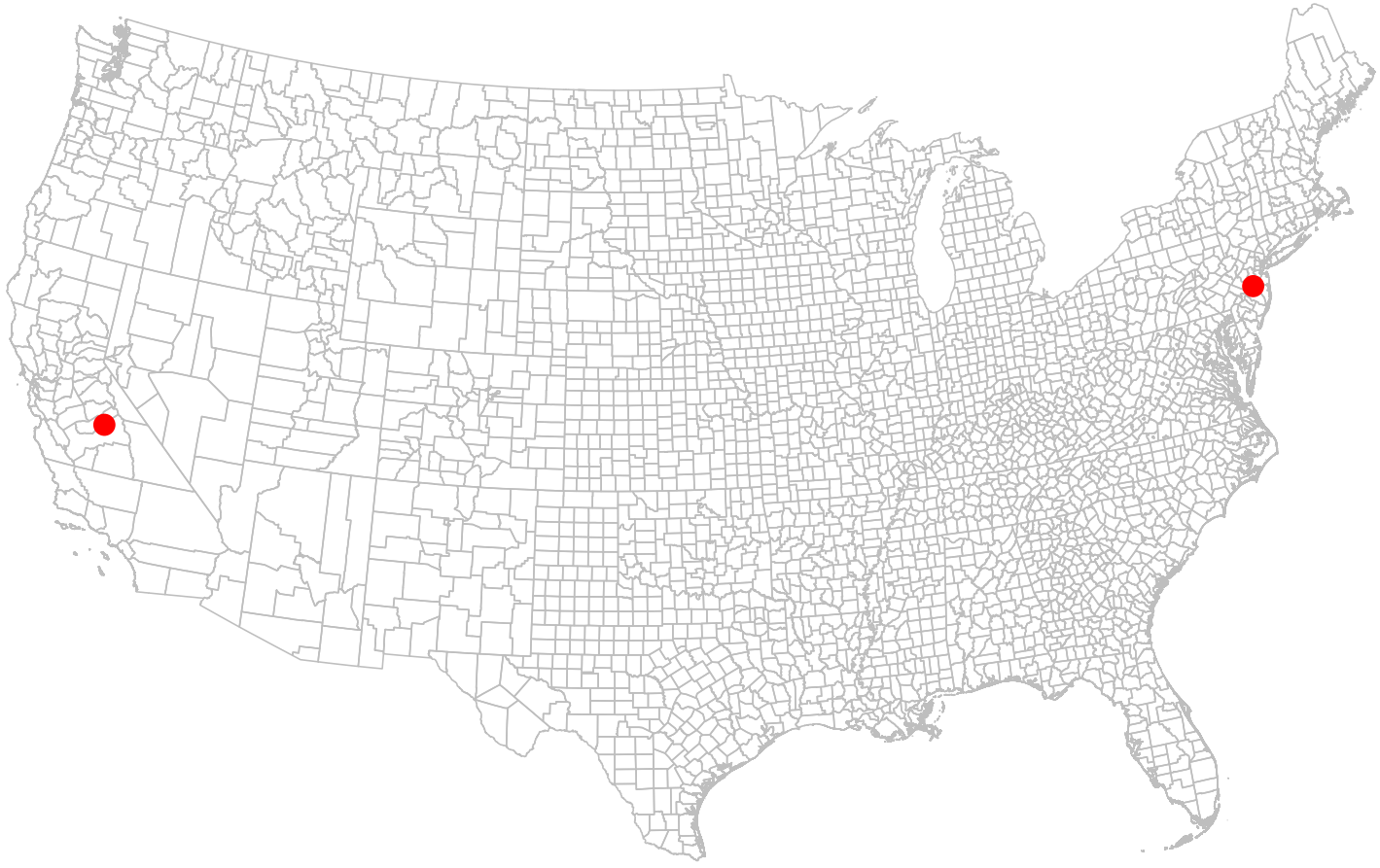

Figure 11: California and New Jersey centroids

Spatial

- How can we draw the corresponding **line**?
- First, we can **combine** the points into a single `sfc` object -

```
p = c(ca_ctr, nj_ctr)
p
```

```
## Geometry set for 2 features
## geometry type:  POINT
## dimension:      XY
## bbox:           xmin: -1707694 ymin: -667480.7 xmax: 2111204 ymax: -206332.8
## epsg (SRID):    2163
## proj4string:    +proj=laea +lat_0=45 +lon_0=-100 +x_0=0 +y_0=0 +a=6370997 +b
## POINT (-1707694 -667480.7)
## POINT (2111204 -206332.8)
```

Spatial

- Second, we can use `st_combine` to **transform** the points into a single `MULTIPOINT` geoemtry
- `st_combine` is similar to `st_union`, but only combines and **does not dissolve**
- Question: compare the results of `st_union(nm)` and `st_combine(nm)`

```
p = st_combine(p)
p
```

```
## Geometry set for 1 feature
## geometry type:  MULTIPOINT
## dimension:      XY
## bbox:           xmin: -1707694 ymin: -667480.7 xmax: 2111204 ymax: -206332.8
## epsg (SRID):    2163
## proj4string:    +proj=laea +lat_0=45 +lon_0=-100 +x_0=0 +y_0=0 +a=6370997 +b
## MULTIPOINT (-1707694 -667480.7, 2111204 -206332.8)
```

Spatial

- The `st_cast` function can be used to convert between different **geometry types**
- The `st_cast` function accepts -
  - The **input layer**
  - The destination **geometry type**
- Finally, we can use `st_cast` to **convert** a `MULTIPOINT` geometry to a `LINESTRING` geometry

```
l = st_cast(p, "LINESTRING")
```

- Note: Meaningless `st_cast` operations will fail: for example, try `st_cast(l, "POLYGON")`

Spatial

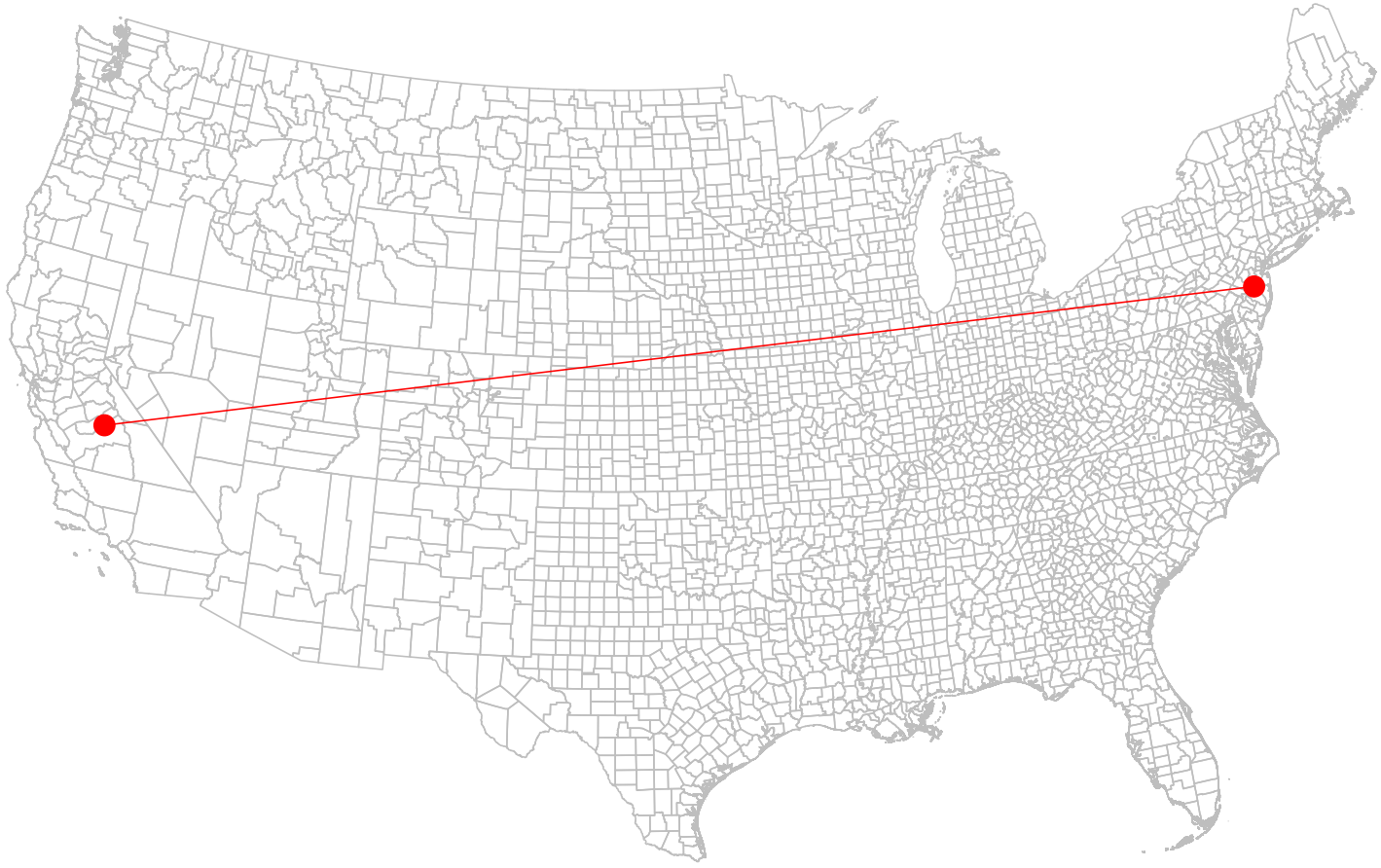

Figure 12: California and New Jersey centroids, with a line

Spatial

- Another example is the **buffer** function `st_buffer`
- For example, here is how we can calculate 100 km buffers around the `airports` -

```
# Method 1 - 'dist' is 'numeric'
airports_100 = st_buffer(
  airports,
  dist = 100 * 10^3
)
# Method 2 - 'dist' is 'units'
airports_100 = st_buffer(
  airports,
  dist = set_units(100, "km")
)
```

```
plot(st_geometry(nm))
plot(st_geometry(airports_100), add = TRUE)
```

Spatial

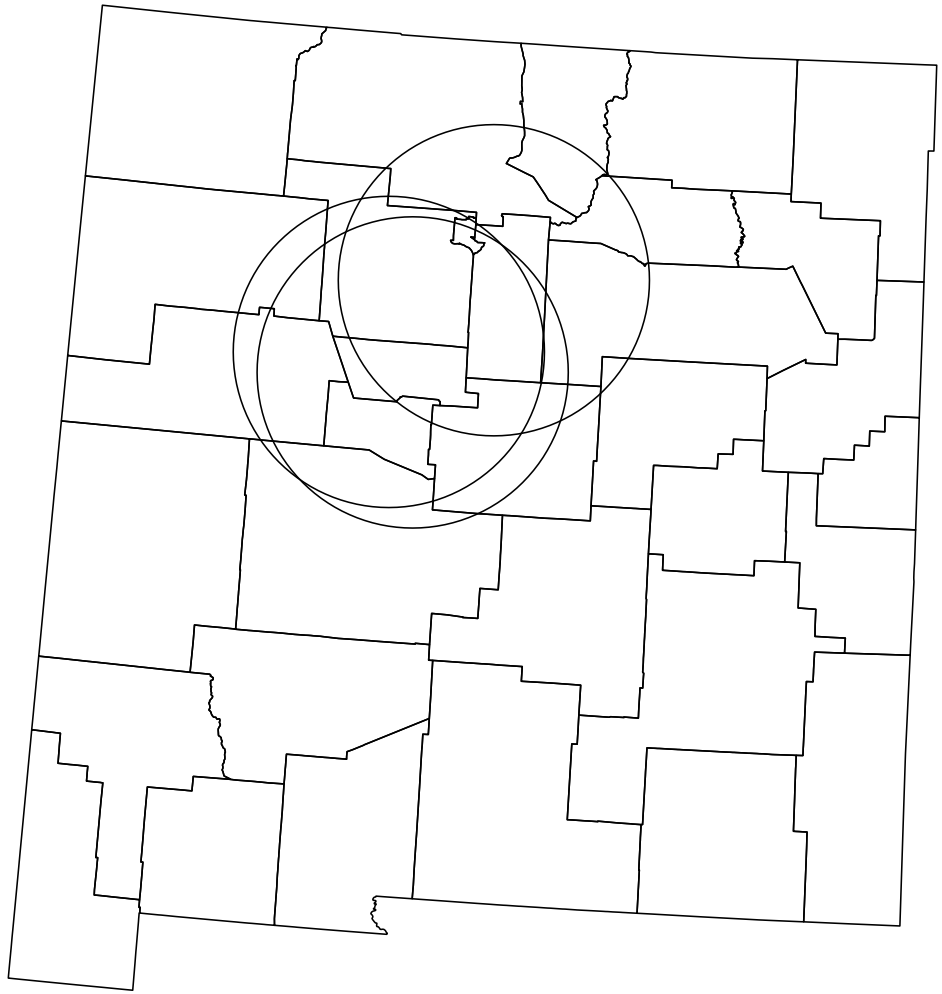

Figure 13: `airports` buffered by 100 km

Spatial

- Other **geometry-generating** functions work on **pairs** of input geometries -
  - `st_intersection`
  - `st_difference`
  - `st_sym_difference`
  - `st_union`

Spatial

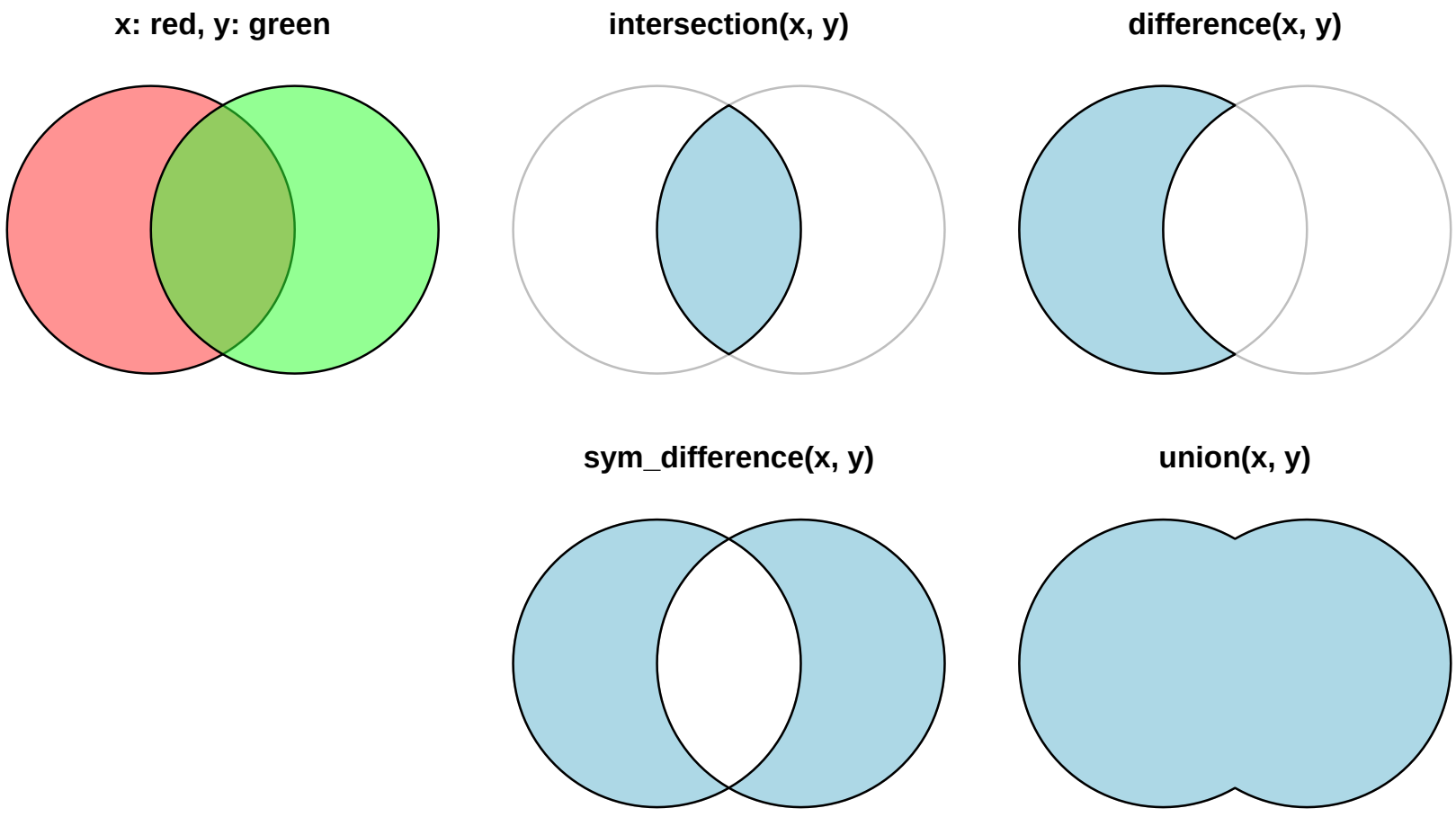

Figure 14: Geometry-generating operations on pairs of layers

Spatial

- How can we calculate the area that is within 100 km range of **all** three airports at the same time?
- We can find the area of **intersection** of the three airports, using `st_intersection` -

```
inter1 = st_intersection(
  airports_100[1, ], airports_100[2, ]
)
inter2 = st_intersection(
  inter1, airports_100[3, ]
)
```

- Plot -

```
plot(st_geometry(nm))
plot(st_geometry(airports_100), add = TRUE)
plot(inter2, col = "lightblue", add = TRUE)
```

Spatial

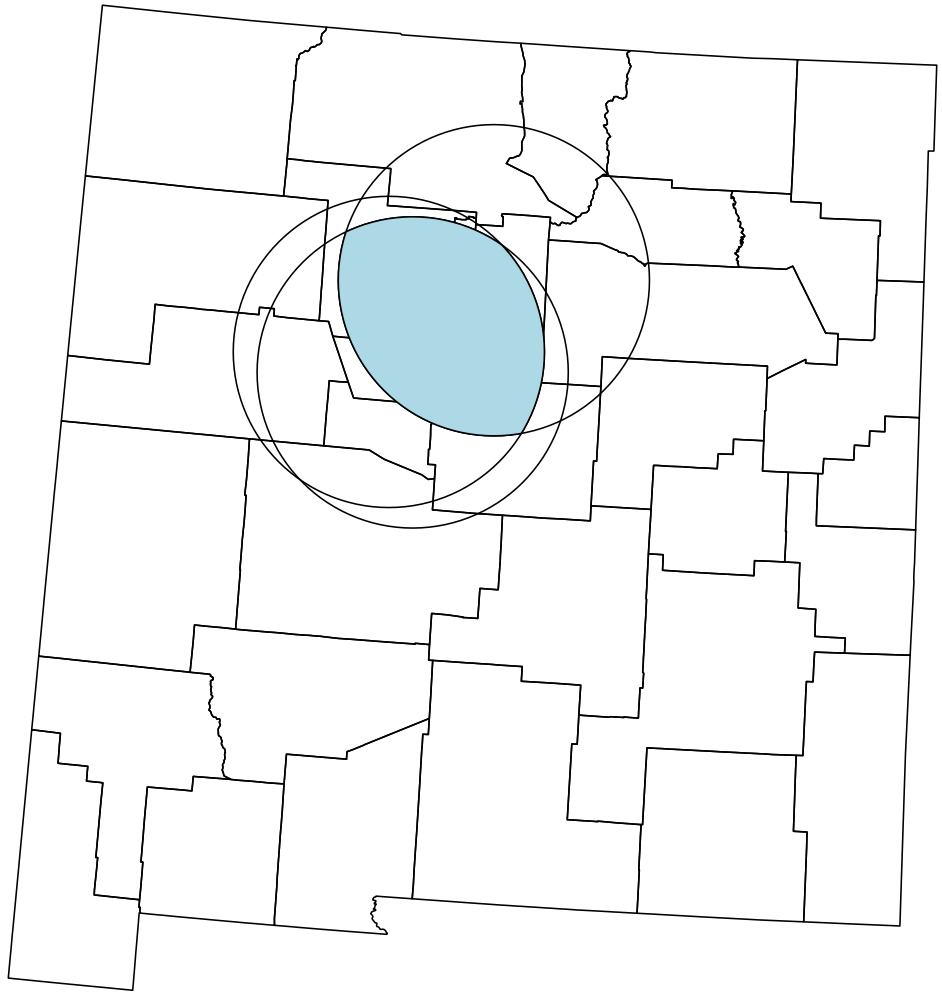

Figure 15: Intersection of three `airports` buffers

Spatial

- How can we calculate the area that is at within 100 km range of **at least one** of the three airports?
- We can **"dissolve"** the buffers, using `st_union` -

```
airports_100u = st_union(airports_100)
```

- Plot -

```
plot(st_geometry(nm))
plot(st_geometry(airports_100u), add = TRUE)
```

Spatial

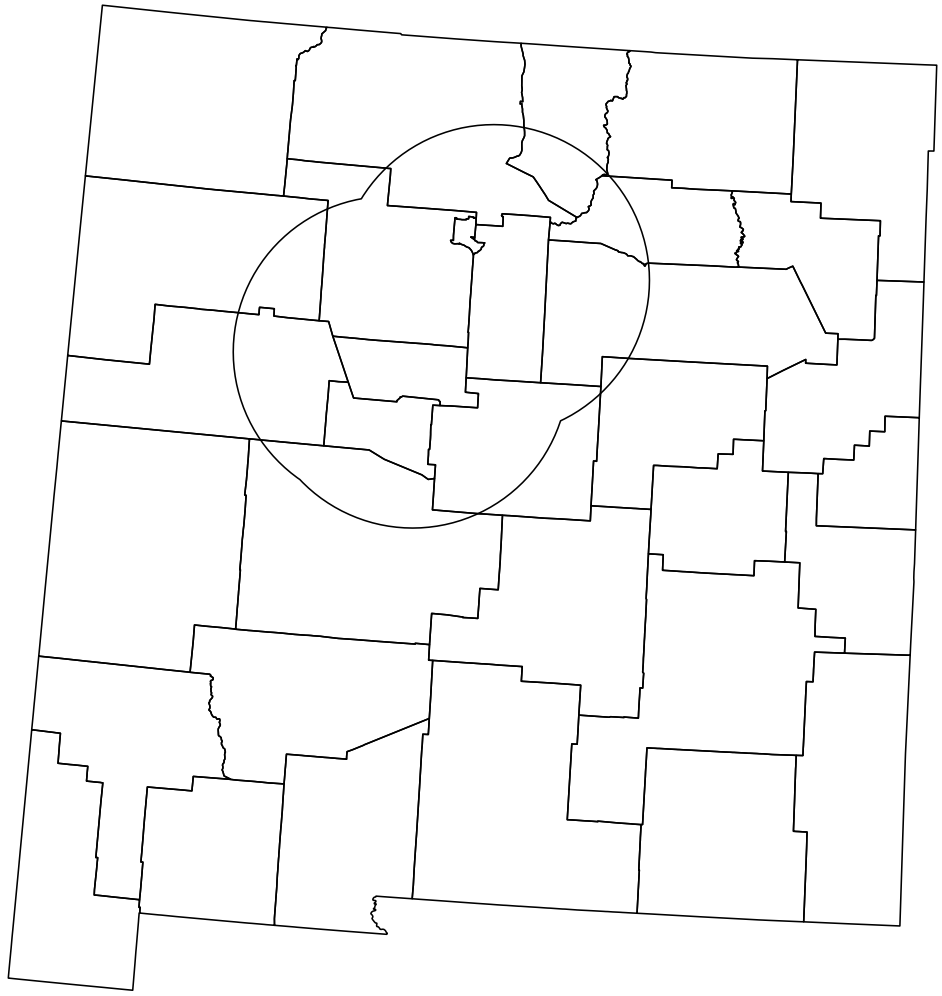

Figure 16: `airports` buffered by 100 km, after `st_union`

Spatial

- We don't have to dissolve **all** features - we can also dissolve by **attribute** or by **location**
- To demonstrate aggregation/dissolving **by attribute**, let's take a subset with all counties of Arizona and Utah -

```
s = county[county$NAME_1 %in% c("Arizona", "Utah"), ]
```

- Plot -

```
plot(s[, "NAME_1"])
```

Spatial

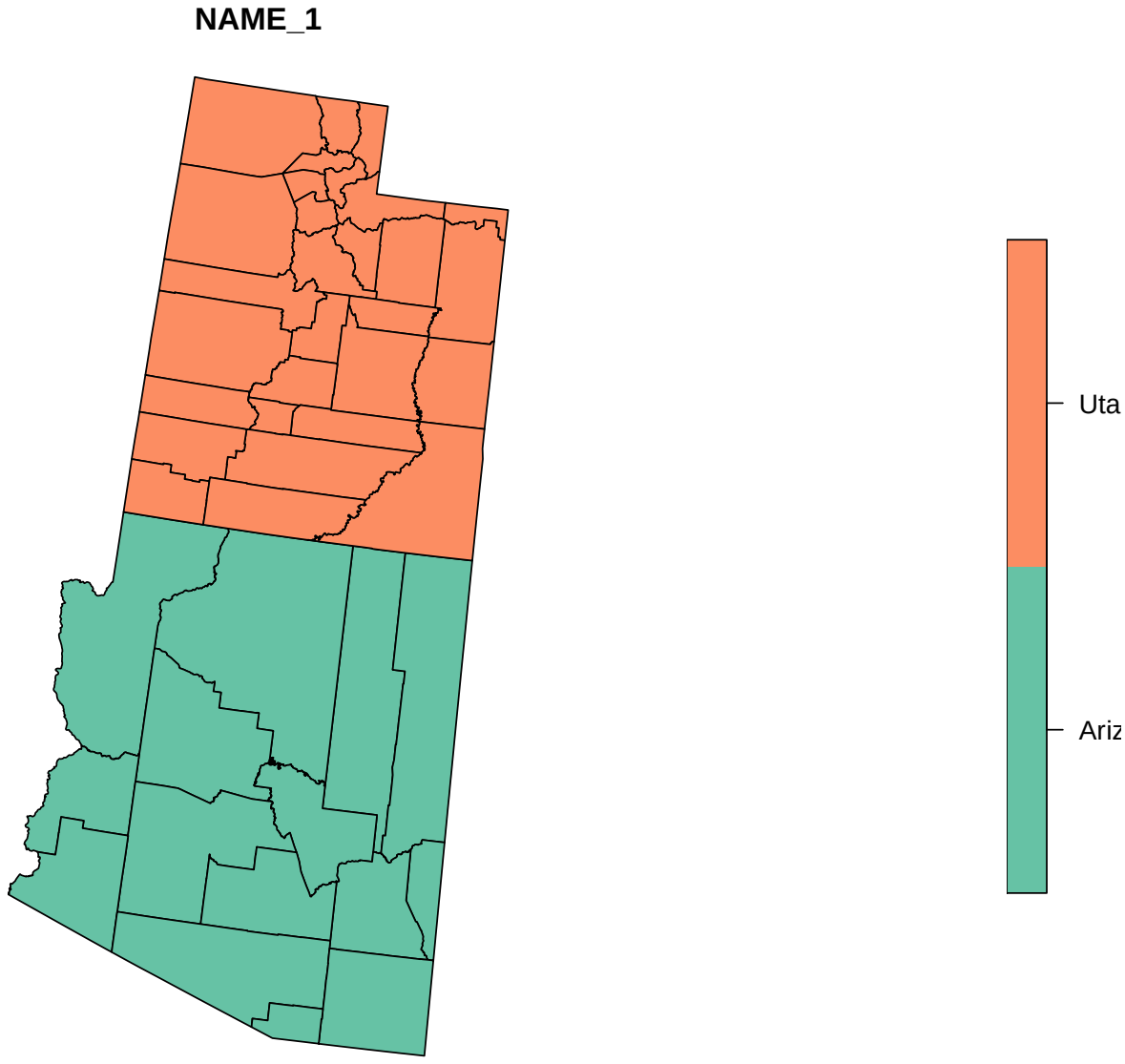

Figure 17: Subset of two states from `county`

Spatial

- As shown before, dissolving **all** features into a single feature is done with `st_union` -

```
s1 = st_union(s)
```

- Plot -

```
plot(s1)
```

Spatial

Figure 18: Union of all counties in `s`

Spatial

- Aggregating/dissolving **by attributes** can be done with aggregate (or using dplyr functions) -

```
# Method 1
s2 = aggregate(
  x = s[, "area"],
  by = st_set_geometry(s[, "NAME_1"], NULL),
  FUN = sum
)
# (Method 2 - 'dplyr')
s2 = s %>%
  group_by(NAME_1) %>%
  summarize(area = sum(area))
```

- Using **Method 1** the unit of measurement is lost, so we need to **reconstruct** it -

```
s2$area = set_units(as.numeric(s2$area), "km^2")
```

Spatial

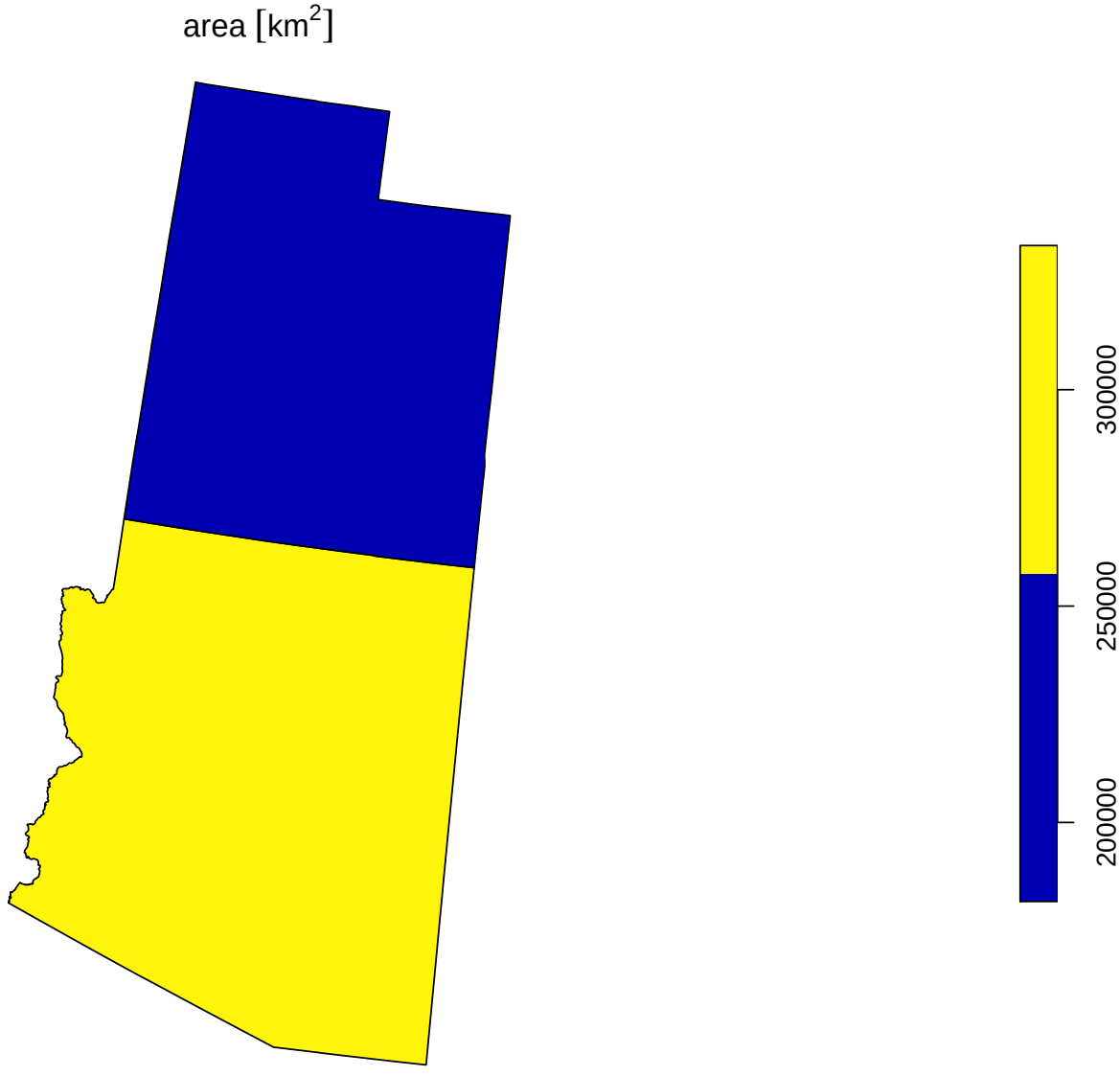

Figure 19: Union by state name of s

Spatial

- The **Convex Hull** of a set X of points is the smallest convex set that contains X

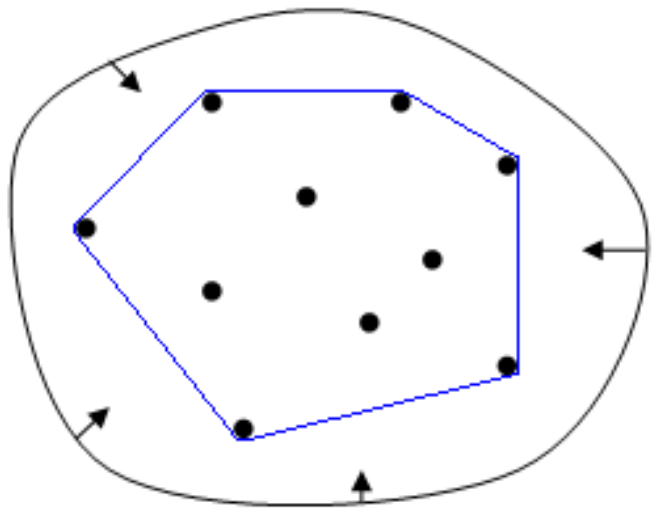

Figure 20: Convex Hull: elastic-band analogy[1]

[1] https://en.wikipedia.org/wiki/Convex_hull

Spatial

- For example -

```
h = st_convex_hull(nm1)
```

- Plot -

```
plot(st_geometry(nm1))
plot(st_geometry(h), add = TRUE, border = "red")
```

Spatial

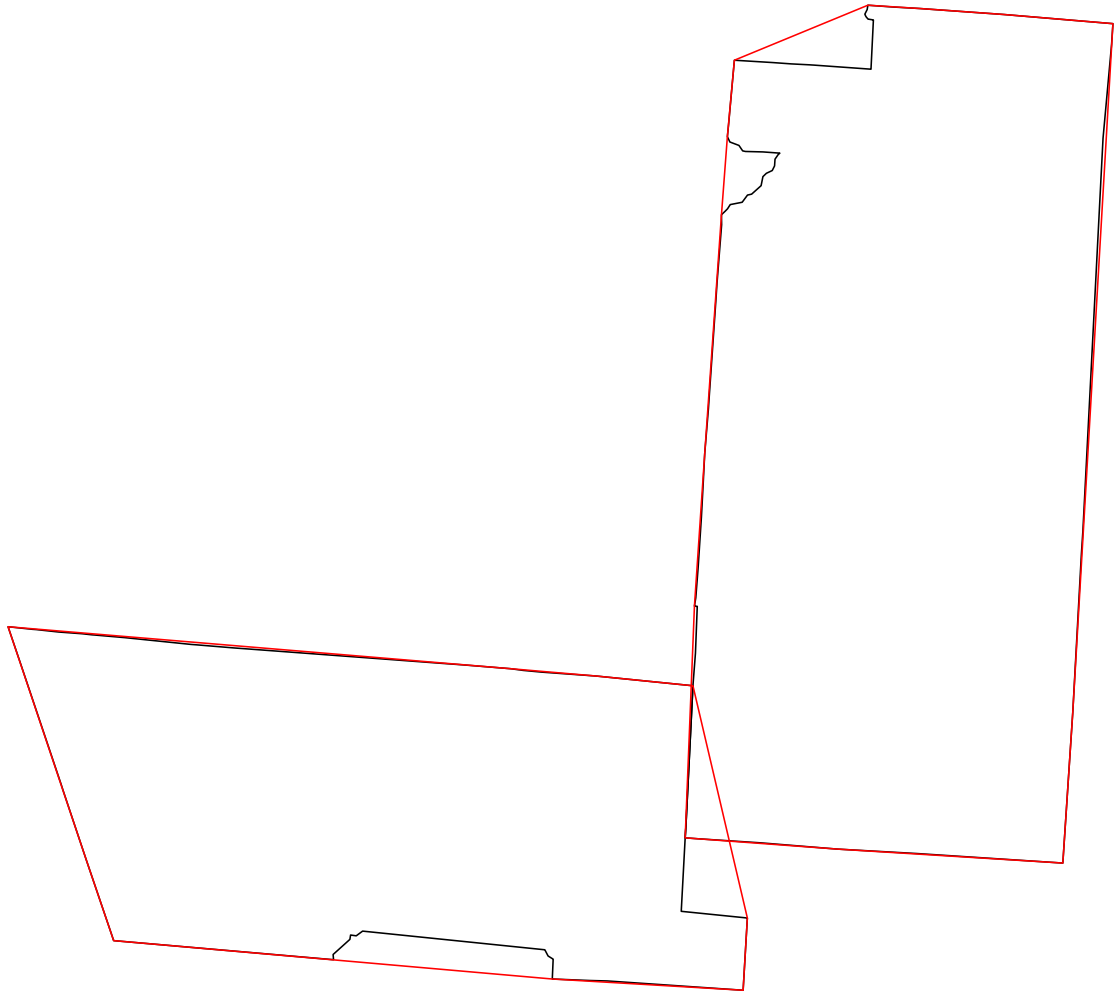

Figure 21: Convex hull polygons for two counties in New Mexico

Spatial

- Question: How can we calculate the convex hull of all polygons in `nm1`?

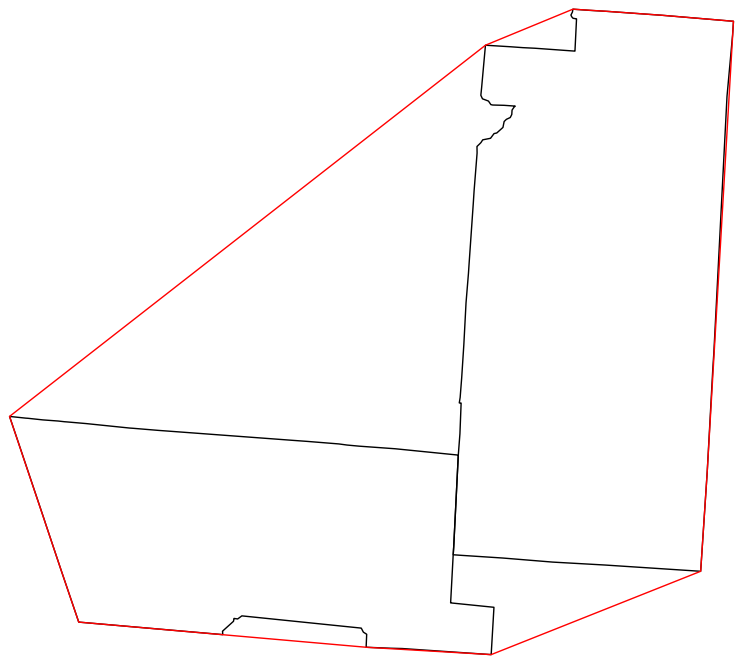

Figure 22: Convex Hull of multiple polygons

Spatial

- Suppose we build a **tunnel** 10 km wide between the centroids of "Harding" and "Sierra" counties in New Mexico
- **Which** counties does the tunnel go through?

```
w = nm[nm$NAME_2 %in% c("Harding", "Sierra"), ]
w_ctr = st_centroid(w)
w_ctr_buf = st_buffer(w_ctr, dist = 5000)
w_ctr_buf_u = st_union(w_ctr_buf)
w_ctr_buf_u_ch = st_convex_hull(w_ctr_buf_u)
nm_w = nm[w_ctr_buf_u_ch, ]
nm_w$NAME_2
## [1] "Harding"    "San Miguel" "Guadalupe"
## [4] "Torrance"   "Socorro"    "Lincoln"
## [7] "Sierra"
```

- Note: we can use `text` with `st_coordinates` to add labels (see next slide)

Spatial

```
plot(
  st_geometry(nm_w),
  border = NA,
  col = "lightblue"
)
plot(
  st_geometry(nm),
  add = TRUE
)
plot(
  st_geometry(w_ctr_buf_u_ch),
  add = TRUE
)
text(
  st_coordinates(st_centroid(nm_w)),
  nm_w$NAME_2
)
```

Spatial

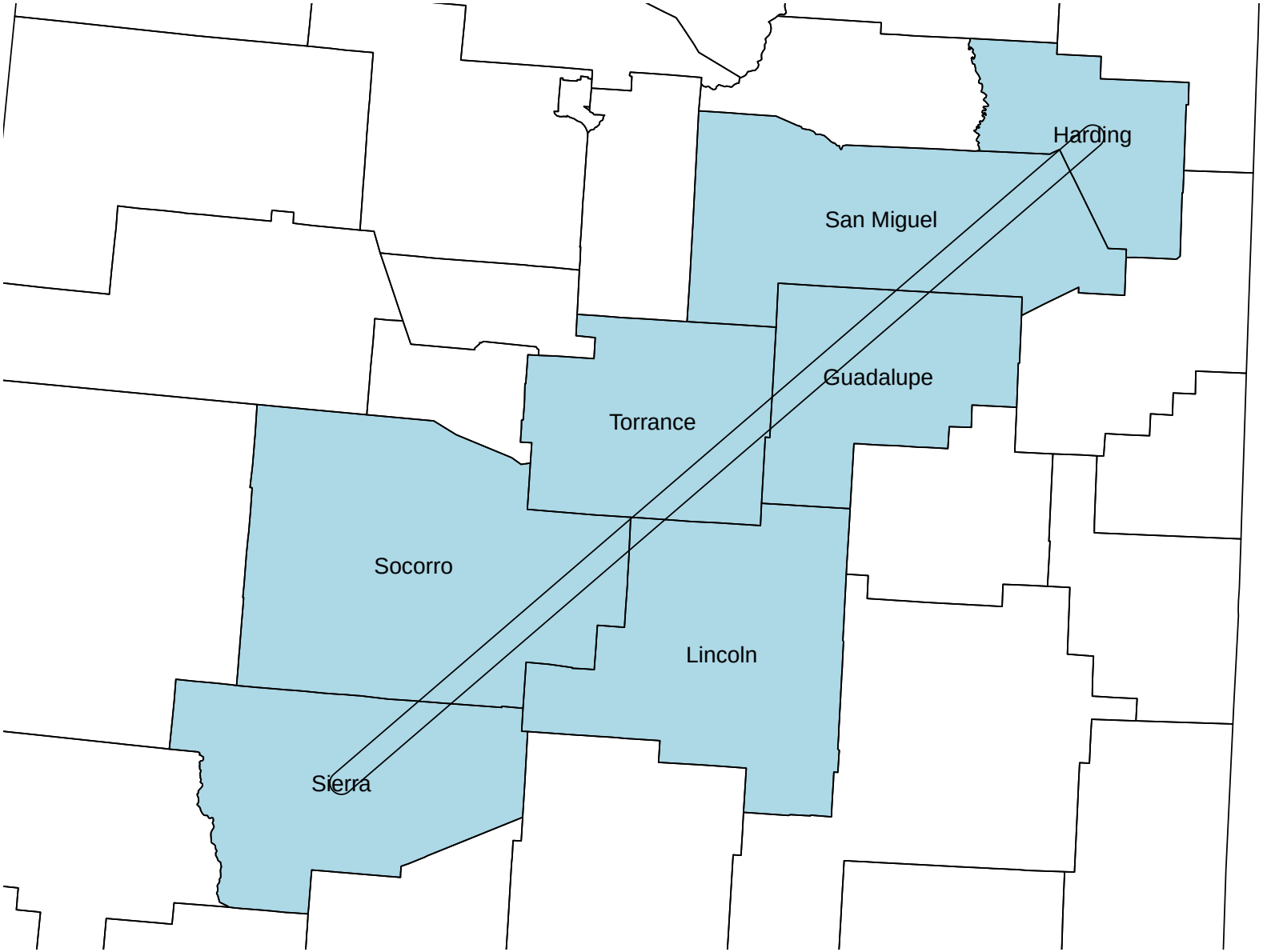

Figure 23: Tunnel between "Sierra" and "Harding" county centroids

Spatial

- How is the area of the "tunnel" **divided** between the various counties that it crosses?
- We can use the fact that `st_intersection(x, y)` returns all (non-empty) geometries resulting from applying the operation to all geometry pairs in `x` and `y` -

```
int = st_intersection(w_ctr_buf_u_ch, nm_w)
area = st_area(int)
area
## Units: m^2
## [1]  199064581  872145769  813577715  702562404
## [5] 1119738156  108077571  575538789
prop = area / sum(area)
prop
## Units: 1
## [1] 0.04533773 0.19863456 0.18529546 0.16001130
## [5] 0.25502469 0.02461508 0.13108118
```

Spatial

- Plot -

```
plot(
  st_geometry(nm_w),
  border = "darkgrey"
)
plot(
  int,
  col = rainbow(7),
  border = "grey",
  add = TRUE
)
text(
  st_coordinates(st_centroid(int)),
  paste0(round(prop, 2)*100, "%"),
  cex = 2
)
```

Spatial

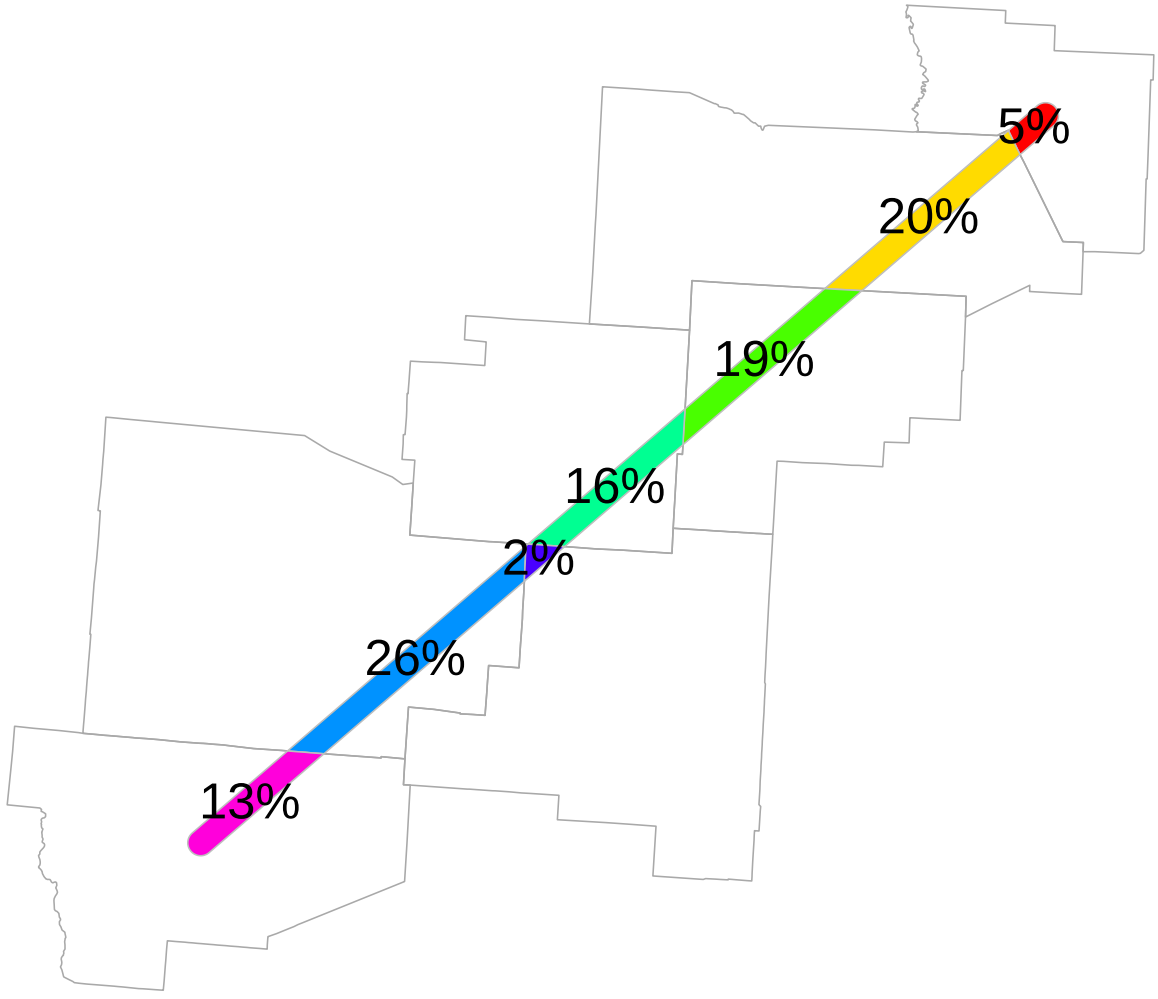

Figure 24: Proportion of tunnel area within each county

Join by attributes

- We can join a **vector layer** and a **table** exactly the same way as **two tables**, e.g. using the `left_join` function from `dplyr`
- This is analogous to **"Join attributes from a table"** in ArcGIS
- In the next example we will join county-level **demographic data** (from `CO-EST2012-Alldata.csv`) with the `county` layer
- The join will be based on the common **Federal Information Processing Standards** (FIPS) code of each county

Join by attributes

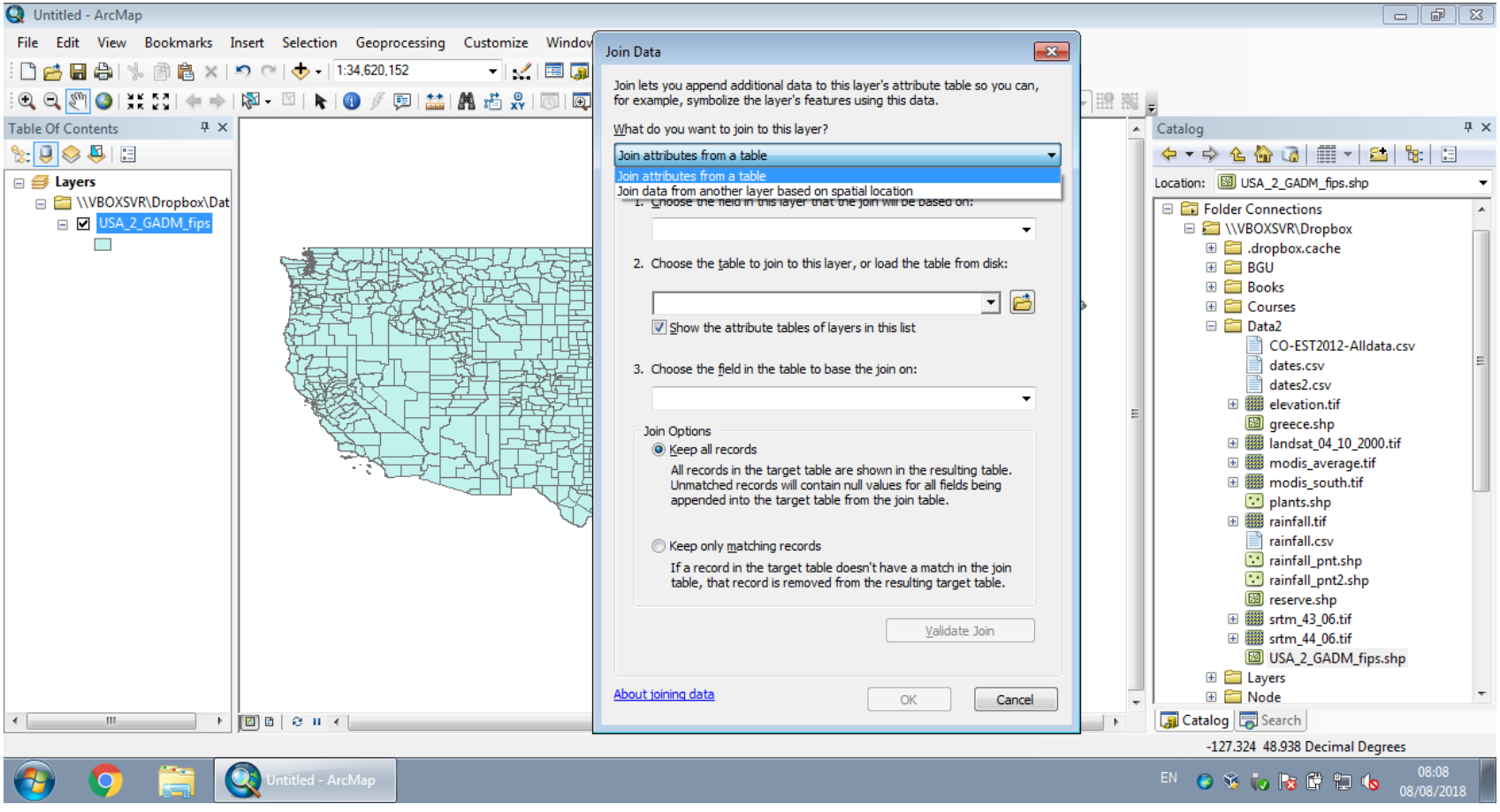

Figure 25: Join by attributes in ArcGIS

Join by attributes

- Let's **read** the `CO-EST2012-Alldata.csv` file -

```
dat = read.csv(
  "CO-EST2012-Alldata.csv",
  stringsAsFactors = FALSE
)
```

Join by attributes

- And **subset** the columns we are interested in -

```
dat = dat[, c("STATE", "COUNTY", "CENSUS2010POP")]
```

```
head(dat)
##   STATE COUNTY CENSUS2010POP
## 1     1      0       4779736
## 2     1      1         54571
## 3     1      3        182265
## 4     1      5         27457
## 5     1      7         22915
## 6     1      9         57322
```

Join by attributes

- Records where `COUNTY` code is 0 are states **sums**, which we will remove -

```
dat = dat[dat$COUNTY != 0, ]
```

```
head(dat)
##   STATE COUNTY CENSUS2010POP
## 2     1      1          54571
## 3     1      3         182265
## 4     1      5          27457
## 5     1      7          22915
## 6     1      9          57322
## 7     1     11          10914
```

Join by attributes

- To get the county FIPS code we need to **standardize** -
  - The state code to a **two-digit** code
  - the county code to **three-digit** code
- The `formatC` function can be used to change between various numeric **formats**, using different "scenarios"
- The **"add leading zeros"** scenario is specified using `width=n`, where `n` is the required number of digits, and `flag="0"` -

```
dat$STATE = formatC(dat$STATE, width = 2, flag = "0")
dat$COUNTY = formatC(dat$COUNTY, width = 3, flag = "0")
dat$FIPS = paste0(dat$STATE, dat$COUNTY)
```

Join by attributes

- Now we have a column named `FIPS` with exactly the **same format** as in the `county` layer -

```
head(dat)
##   STATE COUNTY CENSUS2010POP  FIPS
## 2    01    001          54571 01001
## 3    01    003         182265 01003
## 4    01    005          27457 01005
## 5    01    007          22915 01007
## 6    01    009          57322 01009
## 7    01    011          10914 01011
```

Join by attributes

- Now we can **join** the `county` layer with the `dat` table, based on the common column named `FIPS` -

```
county = left_join(
  county,
  dat[, c("FIPS", "CENSUS2010POP")],
  by = "FIPS"
)
```

- Note: we are using a `left_join`, therefore the `county` layer remains as is; features that do not have a matching `FIPS` value in `dat` will have `NA` in the new `CENSUS2010POP` column

Join by attributes

- Plot with **linear** breaks -

```
plot(county[, "CENSUS2010POP"])
```

- Plot with **quantile** breaks -

```
plot(county[, "CENSUS2010POP"], breaks = "quantile")
```

- Note: the number of color categories can be controlled with `nbreaks`

Join by attributes

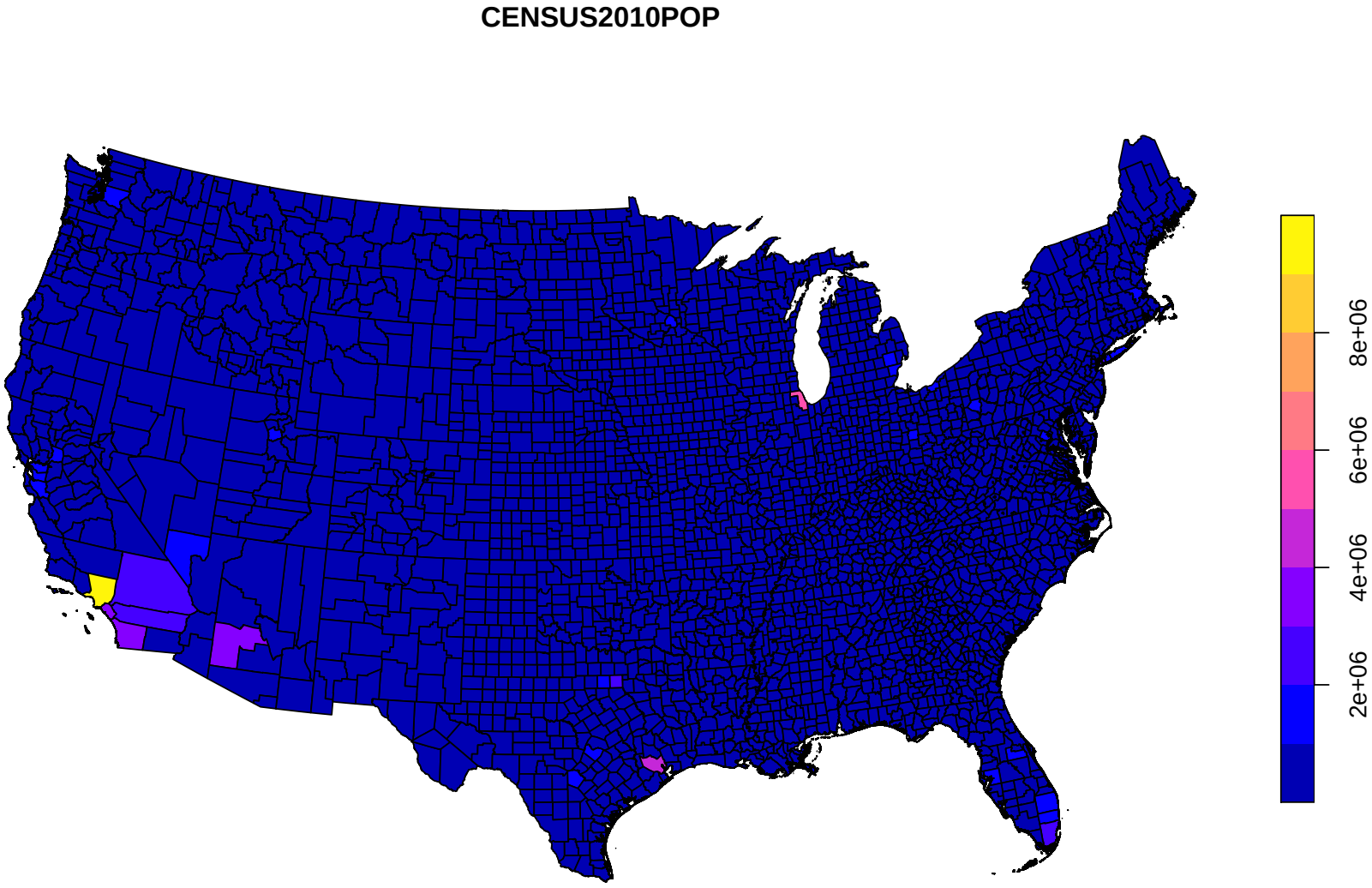

Figure 26: Population size per county in the US, linear breaks

Join by attributes

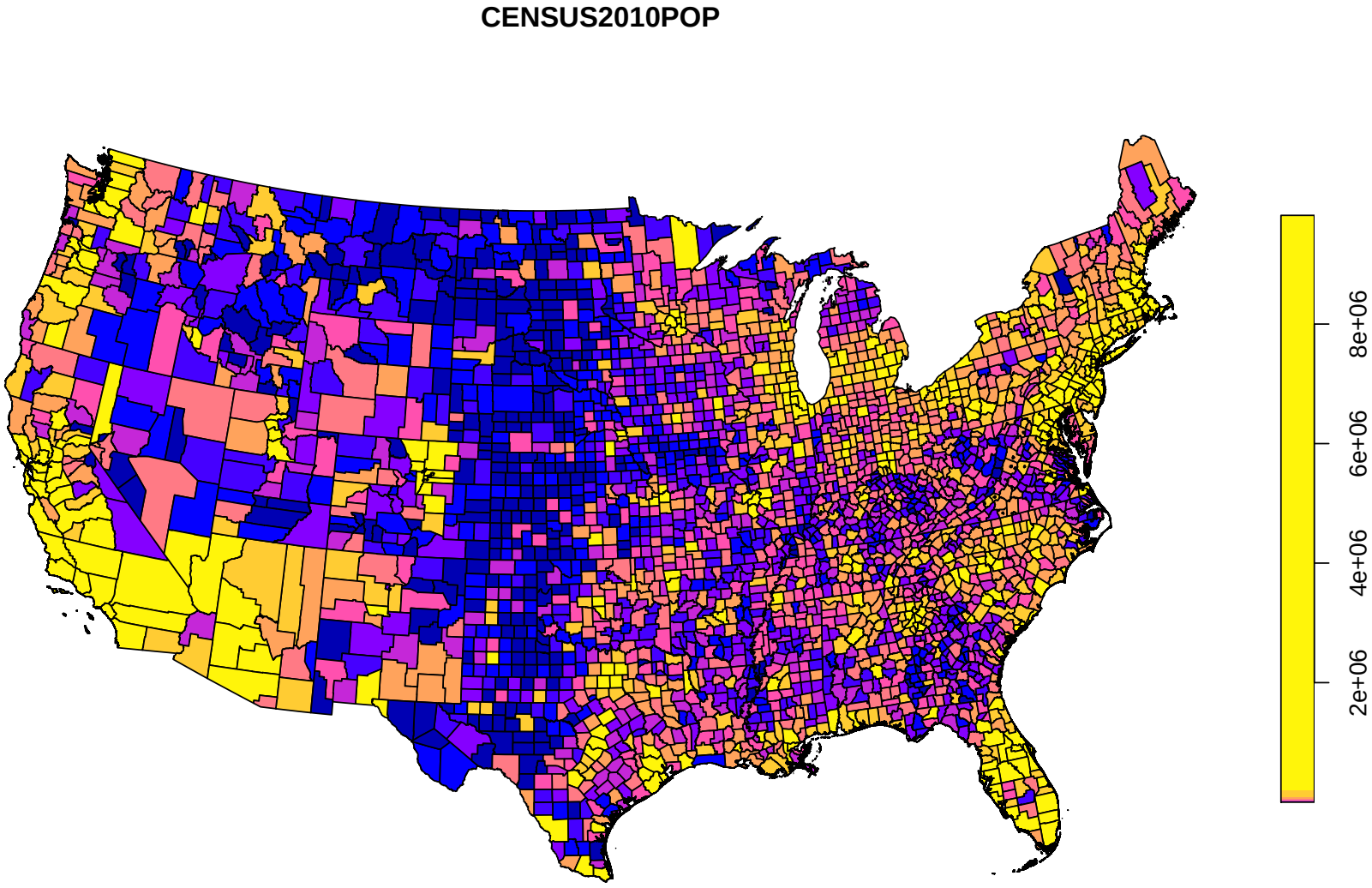

Figure 27: Population size per county in the US, quantile breaks

Join by attributes

- Now that we know the **amount** and **area size** we can calculate population **density** per county -

```
county$density = county$CENSUS2010POP / county$area
```

- Note: the measurement units for the new column are **automatically** determined based on the inputs
- Plot -

```
plot(county[, "density"], breaks = "quantile")
```

Join by attributes

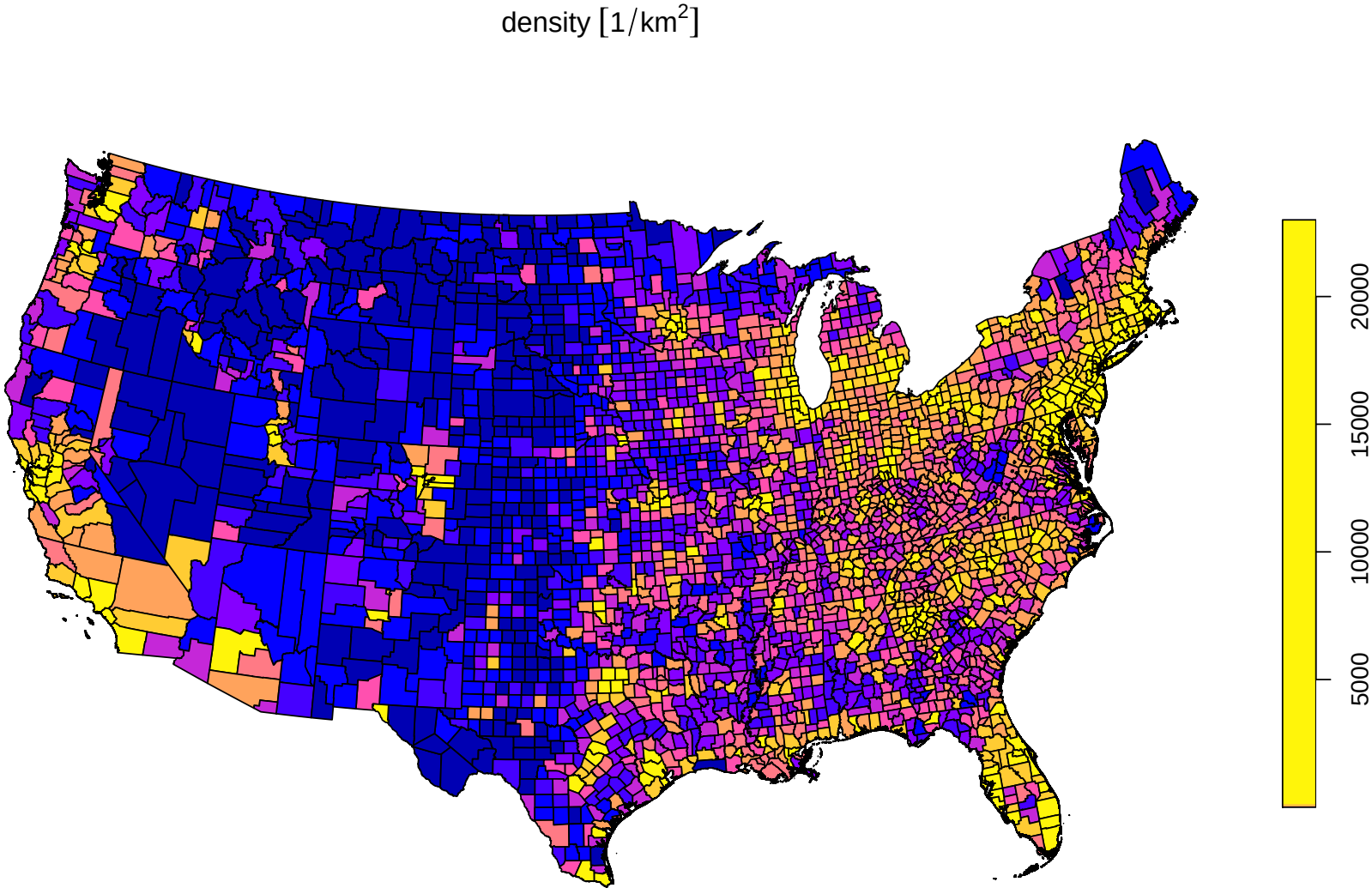

Figure 28: Population density in the US, quantile breaks

Join by attributes

- Question 1: how many features in `county` did not have a matching FIPS code in `dat`? What are their names?
- Question 2: how many features in `dat` do not have a matching FIPS code in `county`?

Join by attributes

- We can calculate the **average population density** in the US, by dividing the total population by the total area -

```
sum(county$CENSUS2010POP, na.rm = TRUE) /
  sum(county$area)
## 39.2295 1/km^2
```

- According to **Wikipedia** the correct value is **40.015**
- Note: simply averaging the `density` column gives an overestimation, because all counties get equal weight while in reality the smaller counties are more dense -

```
mean(county$density, na.rm = TRUE)
## 93.87977 1/km^2
```

Main data structures in the course

Table 1: Main data structures in the course

Category	Class	Lesson
Vector	`numeric`, `character`, `logical`	02
Date	`Date`	03
Table	`data.frame`	04
Matrix	`matrix`	05
Array	`array`	05
Raster	`RasterLayer`, `RasterStack`, `RasterBrick`	05
Vector layer	`sfg`, `sfc`, `sf`	07
Units	`units`	07
List	`list`	10

ggplot2

```
library(ggplot2)
library(raster)

state = getData("GADM", country = "USA", level = 1)
state = st_as_sf(state)
state = state[
  !(state$NAME_1 %in% c("Alaska", "Hawaii")),
]

county$density = as.numeric(county$density)
```

ggplot2

```
ggplot() +
  geom_sf(
    data = county,
    aes(fill = density),
    colour = NA
  ) +
  geom_sf(
    data = state,
    colour = "white", size = 0.25, fill = NA
  ) +
  scale_fill_distiller(
    name = expression(paste("Density (", km^-2, ")")),
    palette = "Spectral",
    trans = "log10",
    labels = as.character, breaks = 10^(-1:5)
  ) +
  theme_bw() +
  theme(
    axis.text.y = element_text(angle = 90, hjust = 0.5),
    panel.border = element_blank()
  )
```

ggplot2

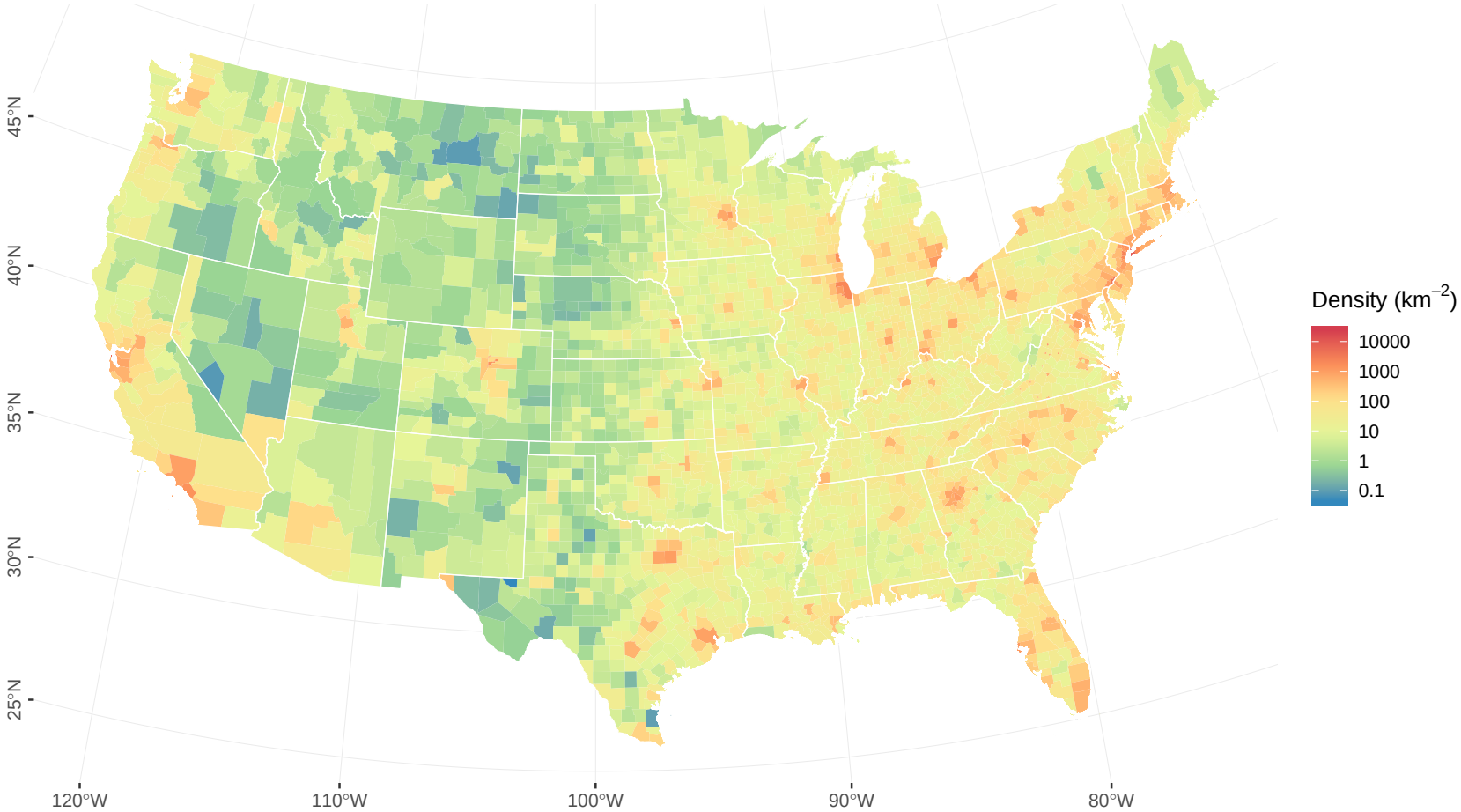

Figure 29: Population density in the US, using ggplot2

Exercise 4 - Submission date **2019-01-13**

Introduction to Spatial Data Programming

Execrise 4

Vector layers & Geometric operations with vector layers

Last updated: 2018-12-20 17:14:04

Question 1

- **Read** the Shapefile of US counties named `USA_2_GADM_fips`
- **Reproject** the layer to the US National Atlas projection (like we did in **Lesson 7**)
- **Choose** a county whose name (`NAME_2` attribute) starts with the same letter as your first name
- Note: there may be more than one county with the same `NAME_2` value in different states (`NAME_1`), in which case you need to choose one state
- **Plot** the US counties (in grey), with the county you chose highlighted (in red), and all bordering counties of the county you chose (in green)
- Note: use a 1 m buffer around the county you chose and subset all counties which intersect with the "buffered" county you chose