# Chapter 3 Spatial Interactions of Vector Data: Subsetting and Joining

**Introduction**

## 3.1 Topological relations

Before we learn spatial subsetting and joining, we first look at topological relations. Topological relations refer to the way multiple spatial objects are spatially related to one another. You can identify various types of spatial relations using the sf package. Our main focus is on the intersections of spatial objects, which can be found using st\_intersects().[63](#fn63) We also briefly cover st\_is\_within\_distance()[64](#fn64).

We first create sf objects we are going to use for illustrations.

**POINTS**

#--- create points ---#  
point\_1 <- st\_point(c(2, 2))  
point\_2 <- st\_point(c(1, 1))  
point\_3 <- st\_point(c(1, 3))  
  
#--- combine the points to make a single sf of points ---#  
(  
points <- list(point\_1, point\_2, point\_3) %>%   
 st\_sfc() %>%   
 st\_as\_sf() %>%   
 mutate(name = c("point 1", "point 2", "point 3"))  
)

Simple feature collection with 3 features and 1 field  
geometry type: POINT  
dimension: XY  
bbox: xmin: 1 ymin: 1 xmax: 2 ymax: 3  
CRS: NA  
 x name  
1 POINT (2 2) point 1  
2 POINT (1 1) point 2  
3 POINT (1 3) point 3

**LINES**

#--- create points ---#  
line\_1 <- st\_linestring(rbind(c(0, 0), c(2.5, 0.5)))  
line\_2 <- st\_linestring(rbind(c(1.5, 0.5), c(2.5, 2)))  
  
#--- combine the points to make a single sf of points ---#  
(  
lines <- list(line\_1, line\_2) %>%   
 st\_sfc() %>%   
 st\_as\_sf() %>%   
 mutate(name = c("line 1", "line 2"))  
)

Simple feature collection with 2 features and 1 field  
geometry type: LINESTRING  
dimension: XY  
bbox: xmin: 0 ymin: 0 xmax: 2.5 ymax: 2  
CRS: NA  
 x name  
1 LINESTRING (0 0, 2.5 0.5) line 1  
2 LINESTRING (1.5 0.5, 2.5 2) line 2

**POLYGONS**

#--- create polygons ---#  
polygon\_1 <- st\_polygon(list(  
 rbind(c(0, 0), c(2, 0), c(2, 2), c(0, 2), c(0, 0))   
))  
  
polygon\_2 <- st\_polygon(list(  
 rbind(c(0.5, 1.5), c(0.5, 3.5), c(2.5, 3.5), c(2.5, 1.5), c(0.5, 1.5))   
))  
  
polygon\_3 <- st\_polygon(list(  
 rbind(c(0.5, 2.5), c(0.5, 3.2), c(2.3, 3.2), c(2, 2), c(0.5, 2.5))   
))  
  
#--- combine the polygons to make an sf of polygons ---#  
(  
polygons <- list(polygon\_1, polygon\_2, polygon\_3) %>%   
 st\_sfc() %>%   
 st\_as\_sf() %>%   
 mutate(name = c("polygon 1", "polygon 2", "polygon 3"))  
)

Simple feature collection with 3 features and 1 field  
geometry type: POLYGON  
dimension: XY  
bbox: xmin: 0 ymin: 0 xmax: 2.5 ymax: 3.5  
CRS: NA  
 x name  
1 POLYGON ((0 0, 2 0, 2 2, 0 ... polygon 1  
2 POLYGON ((0.5 1.5, 0.5 3.5,... polygon 2  
3 POLYGON ((0.5 2.5, 0.5 3.2,... polygon 3

Figure [3.1](spatial-interactions-of-vector-data-subsetting-and-joining.html#fig:plot-point-polygons) shows how they look:

ggplot() +  
 geom\_sf(data = polygons, aes(fill = name), alpha = 0.3) +  
 scale\_fill\_discrete(name = "Polygons") +  
 geom\_sf(data = lines, aes(color = name)) +  
 scale\_color\_discrete(name = "Lines") +   
 geom\_sf(data = points, aes(shape = name), size = 3) +  
 scale\_shape\_discrete(name = "Points")

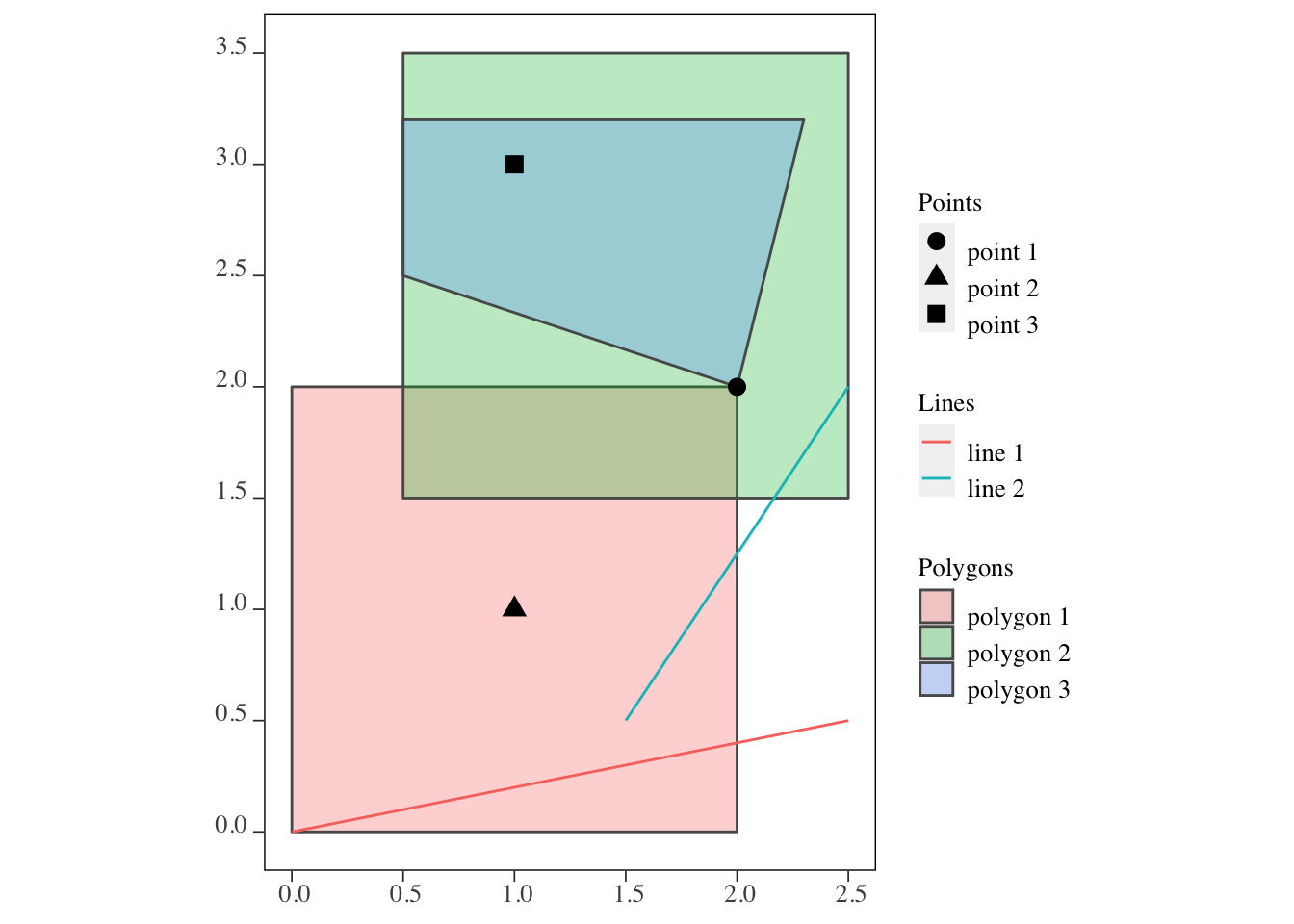


Figure 3.1: Visualization of the points, lines, and polygons

### 3.1.1 st\_intersects()

This function identifies which sfg object in an sf (or sfc) intersects with sfg object(s) in another sf. For example, you can use the function to identify which well is located within which county. st\_intersects() is the most commonly used topological relations. You may not find yourself using st\_intersects(), but it is important to understand what it does as it is the default topological relation used when performing spatial subsetting and joining, which we will cover later.

**points and polygons**

st\_intersects(points, polygons)

Sparse geometry binary predicate list of length 3, where the predicate was `intersects'  
 1: 1, 2, 3  
 2: 1  
 3: 2, 3

As you can see, the output is a list of which polygon(s) each of the points intersect with. 1, 2, and 3 for the first row means that 1st (polygon 1), 2nd (polygon 2), and 3rd (polygon 3) objects of the polygons intersect with the first point (point 1) of the points object. The fact that point 1 is considered to be intersecting with polygon 2 means that the area inside the border is considered a part of the polygon (of course).

If you would like the results of st\_intersects() in a matrix form with boolean values filling the matrix, you can add sparse = FALSE option.

st\_intersects(points, polygons, sparse = FALSE)

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

**lines and polygons**

st\_intersects(lines, polygons)

Sparse geometry binary predicate list of length 2, where the predicate was `intersects'  
 1: 1  
 2: 1, 2

The output is a list of which polygon(s) each of the lines intersect with.

**polygons and polygons**

For polygons vs polygons interaction, st\_intersects() identifies any polygons that either touches (even at a point like polygons 1 and 3) or share some area.

st\_intersects(polygons, polygons)

Sparse geometry binary predicate list of length 3, where the predicate was `intersects'  
 1: 1, 2, 3  
 2: 1, 2, 3  
 3: 1, 2, 3

### 3.1.2 st\_intersection()

Instead of getting just indices of intersecting objects, **st\_intersection()** returns intersecting spatial objects. Another important feature of the function is that non-intersecting parts of the sf objects will be cut out and do not remain in the resulting object. This feature can be very useful. See below for the details.

**lines and polygons**

The following code gets the intersection of line 2 and the polygons.

intersections <- st\_intersection(lines[2, ], polygons) %>%   
 mutate(int\_name = paste0(name, "-", name.1))  
  
#--- take a look ---#  
intersections

Simple feature collection with 2 features and 3 fields  
geometry type: LINESTRING  
dimension: XY  
bbox: xmin: 1.5 ymin: 0.5 xmax: 2.5 ymax: 2  
CRS: NA  
 name name.1 x int\_name  
1 line 2 polygon 1 LINESTRING (1.5 0.5, 2 1.25) line 2-polygon 1  
2 line 2 polygon 2 LINESTRING (2.166667 1.5, 2... line 2-polygon 2

As you can see in Figure [3.2](spatial-interactions-of-vector-data-subsetting-and-joining.html#fig:lines-polygons-int) below, each instance of the intersections of the line and polygons become an observation (line 2-polygon 1 and line 2-polygon 2). Note also that the part of the line that did not intersect is cut out and does not remain in the returned sf.[65](#fn65) This feature can be useful as you can see in Demonstration 4 (Chapter @ref(fig: ))

ggplot() +  
 #--- here are all the original polygons ---#  
 geom\_sf(data = polygons, aes(fill = name), alpha = 0.1) +  
 #--- here is what is returned after st\_intersection ---#  
 geom\_sf(data = intersections, aes(color = int\_name), size = 1.5)

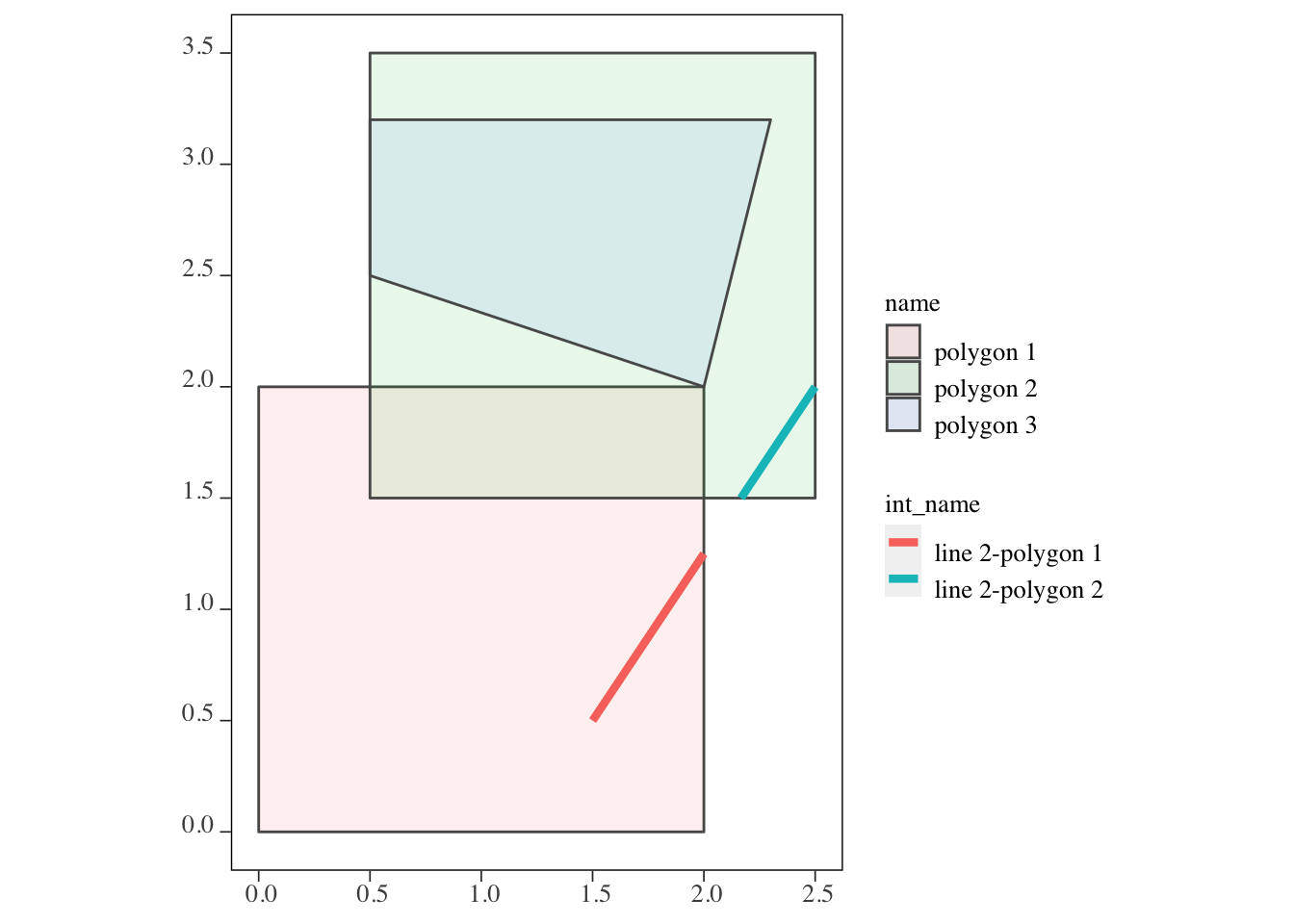


Figure 3.2: The outcome of the intersections of the lines and polygons

**polygons and polygons**

The following code gets the intersection of polygon 1 and polygon 3 with polygon 2.

intersections <- st\_intersection(polygons[c(1,3), ], polygons[2, ]) %>%   
 mutate(int\_name = paste0(name, "-", name.1))  
  
#--- take a look ---#  
intersections

Simple feature collection with 2 features and 3 fields  
geometry type: POLYGON  
dimension: XY  
bbox: xmin: 0.5 ymin: 1.5 xmax: 2.3 ymax: 3.2  
CRS: NA  
 name name.1 x int\_name  
1 polygon 1 polygon 2 POLYGON ((0.5 2, 2 2, 2 1.5... polygon 1-polygon 2  
2 polygon 3 polygon 2 POLYGON ((0.5 2.5, 0.5 3.2,... polygon 3-polygon 2

As you can see in Figure [3.3](spatial-interactions-of-vector-data-subsetting-and-joining.html#fig:polygons-polygons-int), each instance of the intersections of polygons 1 and 3 against polygon 2 becomes an observation (polygon 1-polygon 2 and polygon 3-polygon 2). Just like the lines-polygons case, the non-intersecting part of polygons 1 and 3 are cut out and do not remain in the returned sf. We will see later that st\_intersection() can be used to find area-weighted values from the intersecting polygons with a help from st\_area().

ggplot() +  
 #--- here are all the original polygons ---#  
 geom\_sf(data = polygons, aes(fill = name), alpha = 0.1) +  
 #--- here is what is returned after st\_intersection ---#  
 geom\_sf(data = intersections, aes(fill = int\_name))

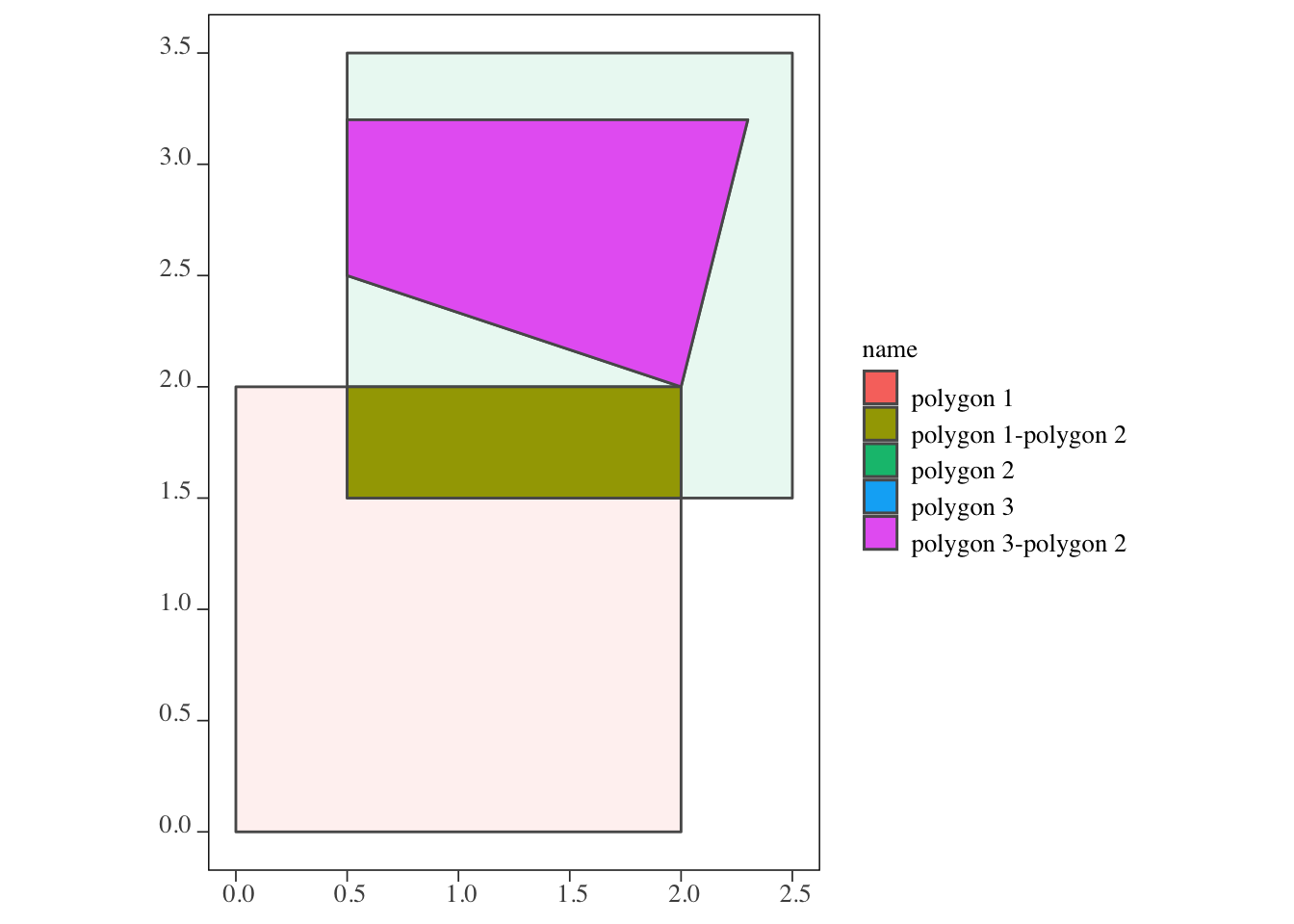


Figure 3.3: The outcome of the intersections of polygon 2 and polygons 1 and 3

### 3.1.3 st\_is\_within\_distance()

This function identifies whether two spatial objects are within the distance you specify as the name suggests[66](#fn66).

Let’s first create two sets of points.

set.seed(38424738)  
  
points\_set\_1 <- lapply(1:5, function(x) st\_point(runif(2))) %>%   
 st\_sfc() %>% st\_as\_sf() %>%   
 mutate(id = 1:nrow(.))  
  
points\_set\_2 <- lapply(1:5, function(x) st\_point(runif(2))) %>%   
 st\_sfc() %>% st\_as\_sf() %>%   
 mutate(id = 1:nrow(.))

Here is how they are spatially distributed (Figure [3.4](spatial-interactions-of-vector-data-subsetting-and-joining.html#fig:map-points-points-points)). Instead of circles of points, their corresponding id (or equivalently row number here) values are displayed.

ggplot() +  
 geom\_sf\_text(data = points\_set\_1, aes(label = id), color = "red") +  
 geom\_sf\_text(data = points\_set\_2, aes(label = id), color = "blue")

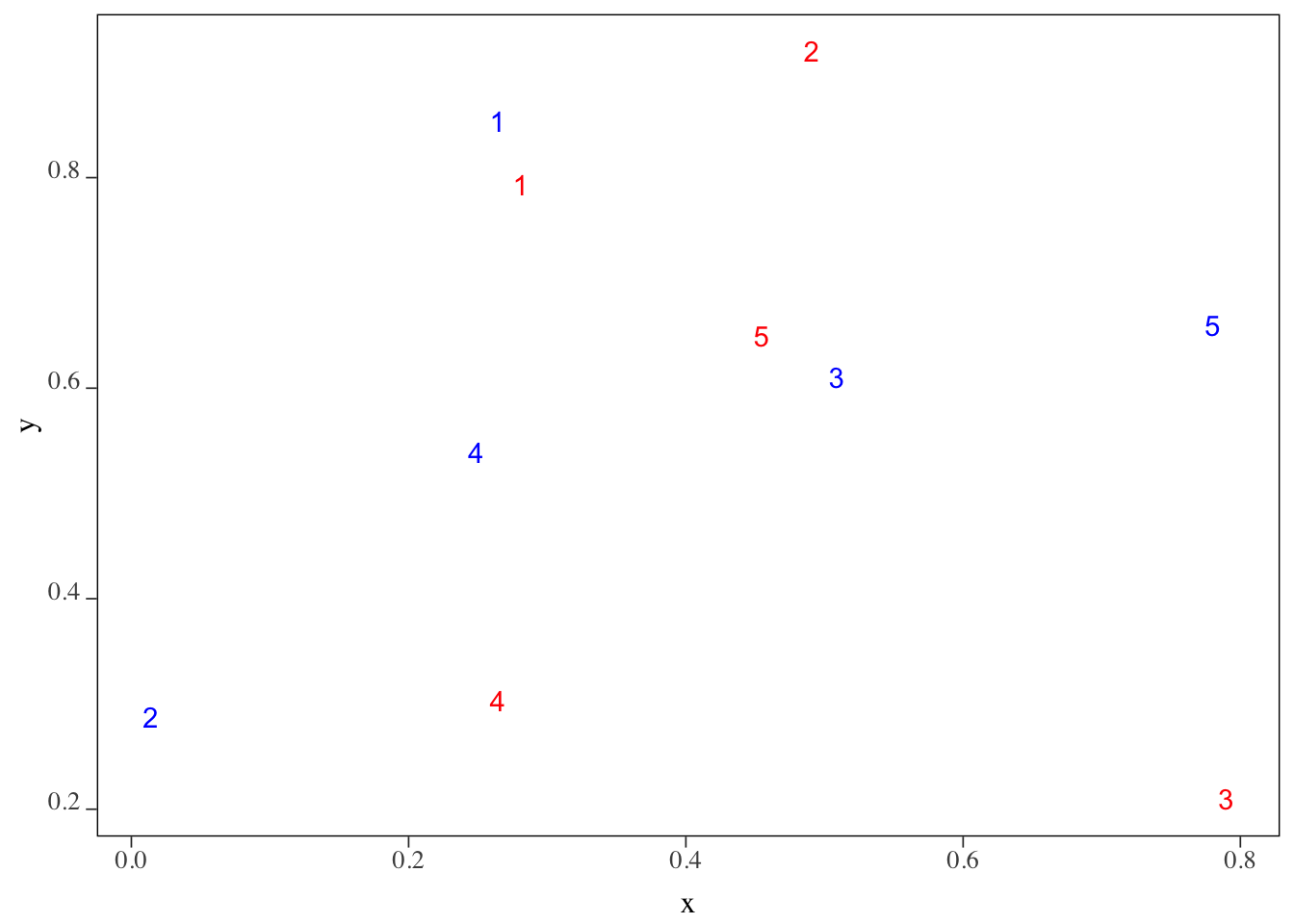


Figure 3.4: The locations of the set of points

We want to know which of the blue points (points\_set\_2) are located within 0.2 from each of the red points (points\_set\_1). The following figure (Figure [3.5](spatial-interactions-of-vector-data-subsetting-and-joining.html#fig:points-points-within)) gives us the answer visually.

#--- create 0.2 buffers around points in points\_set\_1 ---#  
buffer\_1 <- st\_buffer(points\_set\_1, dist = 0.2)  
  
ggplot() +  
 geom\_sf(data = buffer\_1, color = "red", fill = NA) +  
 geom\_sf\_text(data = points\_set\_1, aes(label = id), color = "red") +  
 geom\_sf\_text(data = points\_set\_2, aes(label = id), color = "blue")

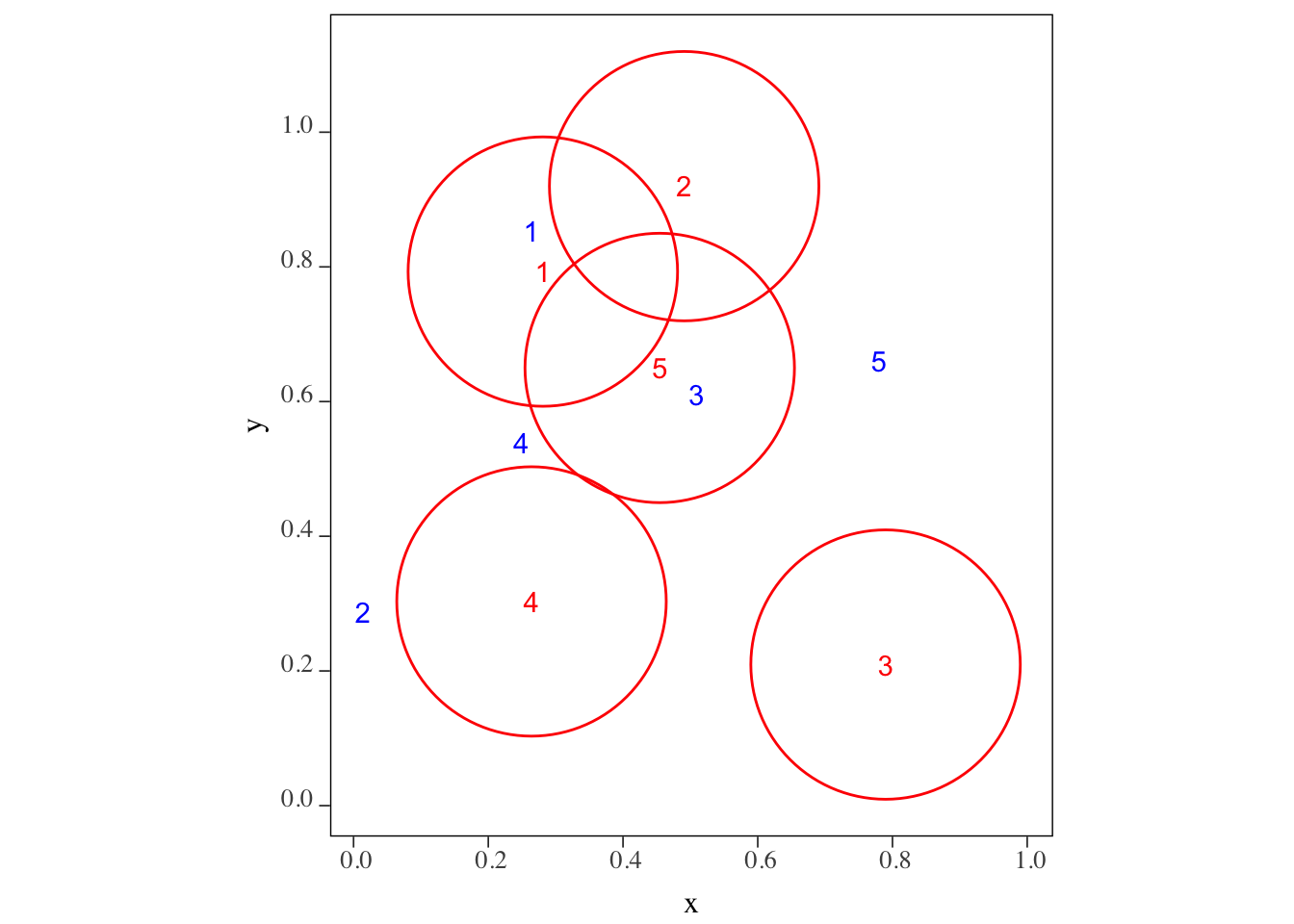


Figure 3.5: The blue points within 0.2 radius of the red points

Confirm your visual inspection results with the outcome of the following code using st\_is\_within\_distance() function.

st\_is\_within\_distance(points\_set\_1, points\_set\_2, dist = 0.2)

Sparse geometry binary predicate list of length 5, where the predicate was `is\_within\_distance'  
 1: 1  
 2: (empty)  
 3: (empty)  
 4: (empty)  
 5: 3

## 3.2 Spatial Subsetting (or Flagging)

Spatial subsetting refers to operations that narrow down the geographic scope of a spatial object based on another spatial object. We illustrate spatial subsetting using Kansas county borders, the boundary of the High-Plains Aquifer (HPA), and agricultural irrigation wells in Kansas.

First, let’s import all the files we will use in this section.

#--- Kansas county borders ---#  
KS\_counties <- readRDS("./Data/KS\_county\_borders.rds")  
  
#--- HPA boundary ---#  
hpa <- st\_read(dsn = "./Data", layer = "hp\_bound2010") %>%   
 .[1, ] %>%   
 st\_transform(st\_crs(KS\_counties))   
  
#--- all the irrigation wells in KS ---#  
KS\_wells <- readRDS("./Data/Kansas\_wells.rds") %>%   
 st\_transform(st\_crs(KS\_counties))  
  
#--- US railroad ---#  
rail\_roads <- st\_read(dsn = "./Data/", layer = "tl\_2015\_us\_rails") %>%   
 st\_transform(st\_crs(KS\_counties))

### 3.2.1 polygons vs polygons

The following map (Figure [3.6](spatial-interactions-of-vector-data-subsetting-and-joining.html#fig:overlap-KS-county-HPA)) shows the Kansas portion of the HPA and KS counties.

#--- add US counties layer ---#  
tm\_shape(KS\_counties) +  
 tm\_polygons() +  
#--- add High-Plains Aquifer layer ---#  
tm\_shape(hpa) +  
 tm\_fill(col = "blue", alpha = 0.3)

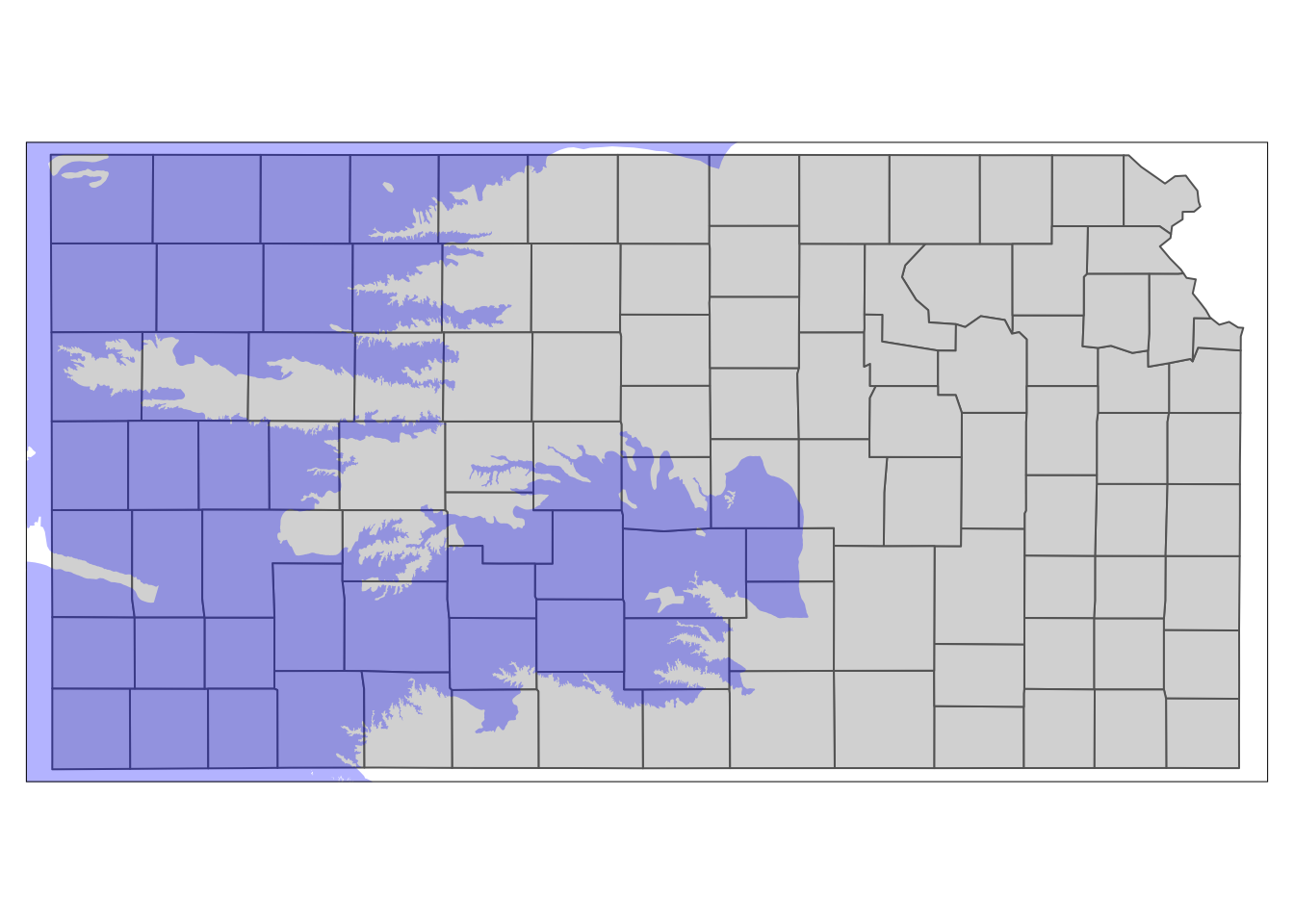


Figure 3.6: Kansas portion of High-Plains Aquifer and Kansas counties

The goal here is to select only the counties that intersects with the HPA boundary. When subsetting a data.frame by specifying the row numbers you would like to select, you can do

#--- NOT RUN ---#  
data.frame[vector of row numbers, ]

Spatial subsetting of sf objects works in a similar syntax:

#--- NOT RUN ---#  
sf\_1[sf\_2, ]

where you are subsetting sf\_1 based on sf\_2. Instead of row numbers, you provide another sf object in place. The following code spatially subsets KS counties based on the HPA boundary.

counties\_in\_hpa <- KS\_counties[hpa, ]

See the results below in Figure [3.7](spatial-interactions-of-vector-data-subsetting-and-joining.html#fig:default-subset).

#--- add US counties layer ---#  
tm\_shape(counties\_in\_hpa) +  
 tm\_polygons() +  
#--- add High-Plains Aquifer layer ---#  
tm\_shape(hpa) +  
 tm\_fill(col = "blue", alpha = 0.3)

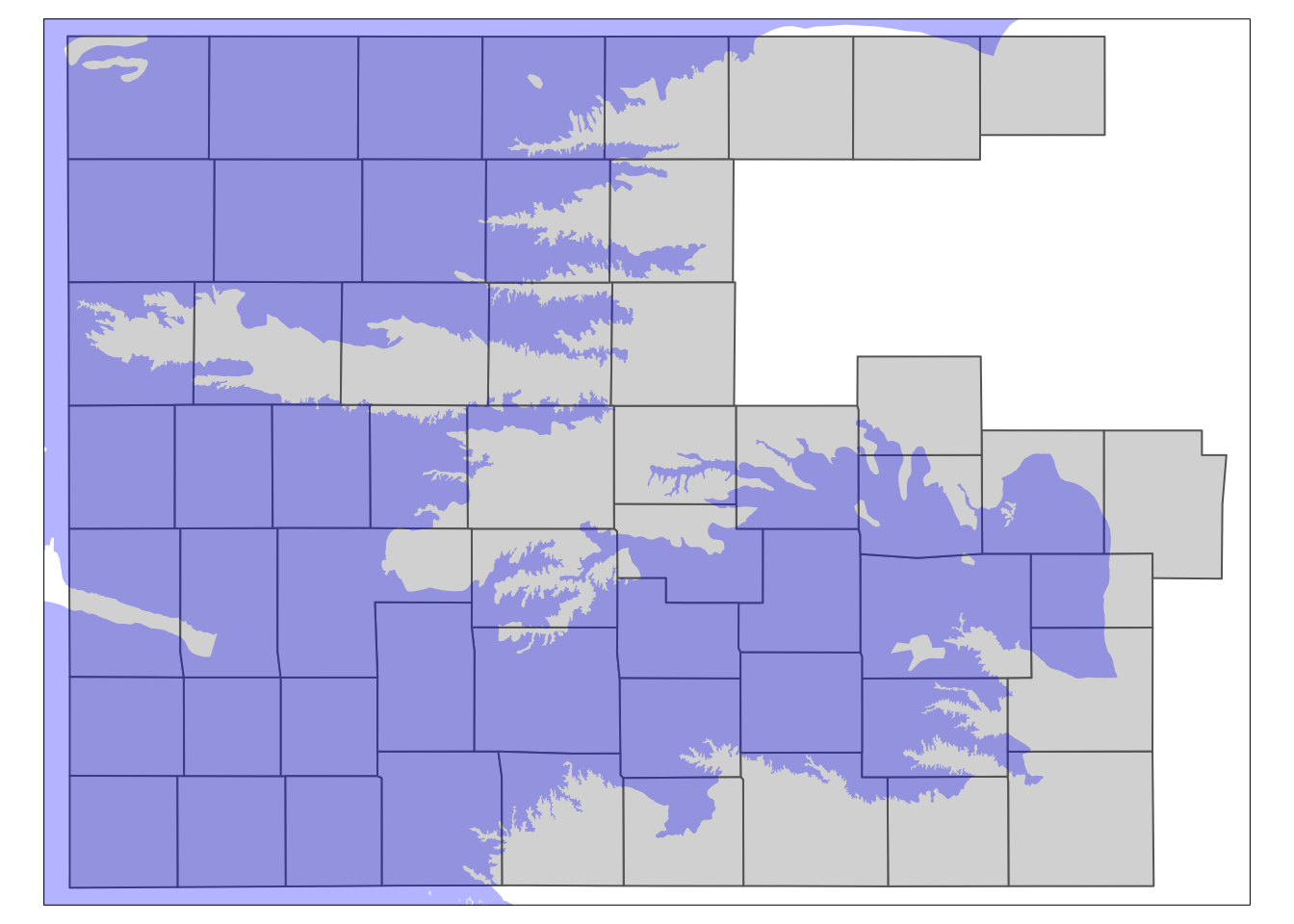


Figure 3.7: The results of spatially subsetting KS counties based on HPA boundary

You can see that only the counties that intersect with the HPA boundary remained. This is because when you use the above syntax of sf\_1[sf\_2, ], the default underlying topological relations is st\_intersects(). So, if an object in sf\_1 intersects with any of the objects in sf\_2 even slightly, then it will remain after subsetting.

You can specify the spatial operation to be used as an option as in

#--- NOT RUN ---#  
sf\_1[sf\_2, op = topological\_relation\_type]

For example, if you only want counties that are completely within the HPA boundary, you can do the following (the map of the results in Figure [3.8](spatial-interactions-of-vector-data-subsetting-and-joining.html#fig:within-subset)):

counties\_within\_hpa <- KS\_counties[hpa, , op = st\_within]

#--- add US counties layer ---#  
tm\_shape(counties\_within\_hpa) +  
 tm\_polygons() +  
#--- add High-Plains Aquifer layer ---#  
tm\_shape(hpa) +  
 tm\_fill(col = "blue", alpha = 0.3)

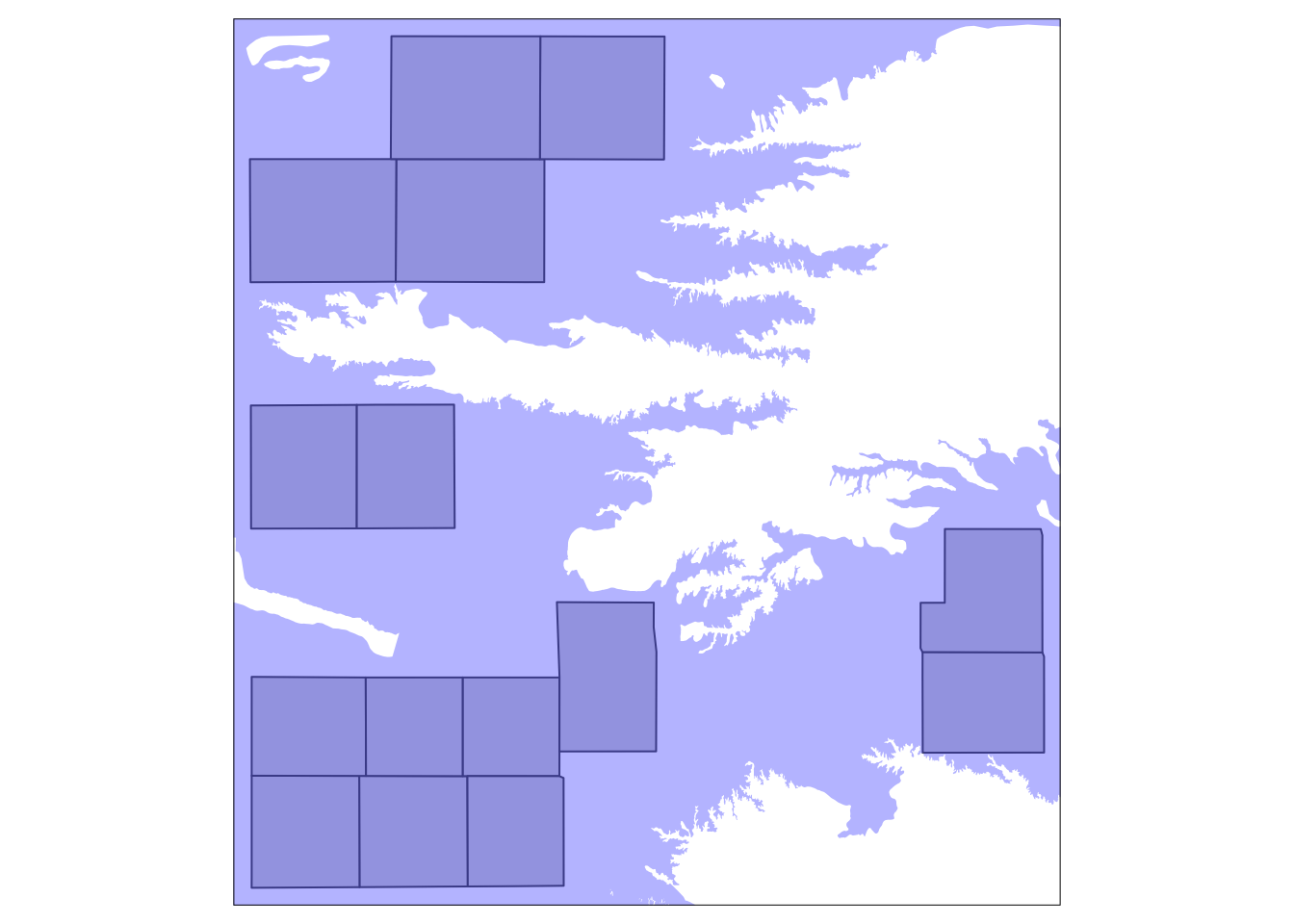


Figure 3.8: Kansas counties that are completely within HPA boundary

### 3.2.2 points vs polygons

The following map (Figure [3.9](spatial-interactions-of-vector-data-subsetting-and-joining.html#fig:map-wells-county)) shows the Kansas portion of the HPA and all the irrigation wells in KS.

tm\_shape(KS\_wells) +  
 tm\_symbols(size = 0.1) +  
tm\_shape(hpa) +  
 tm\_polygons(col = "blue", alpha = 0.1)

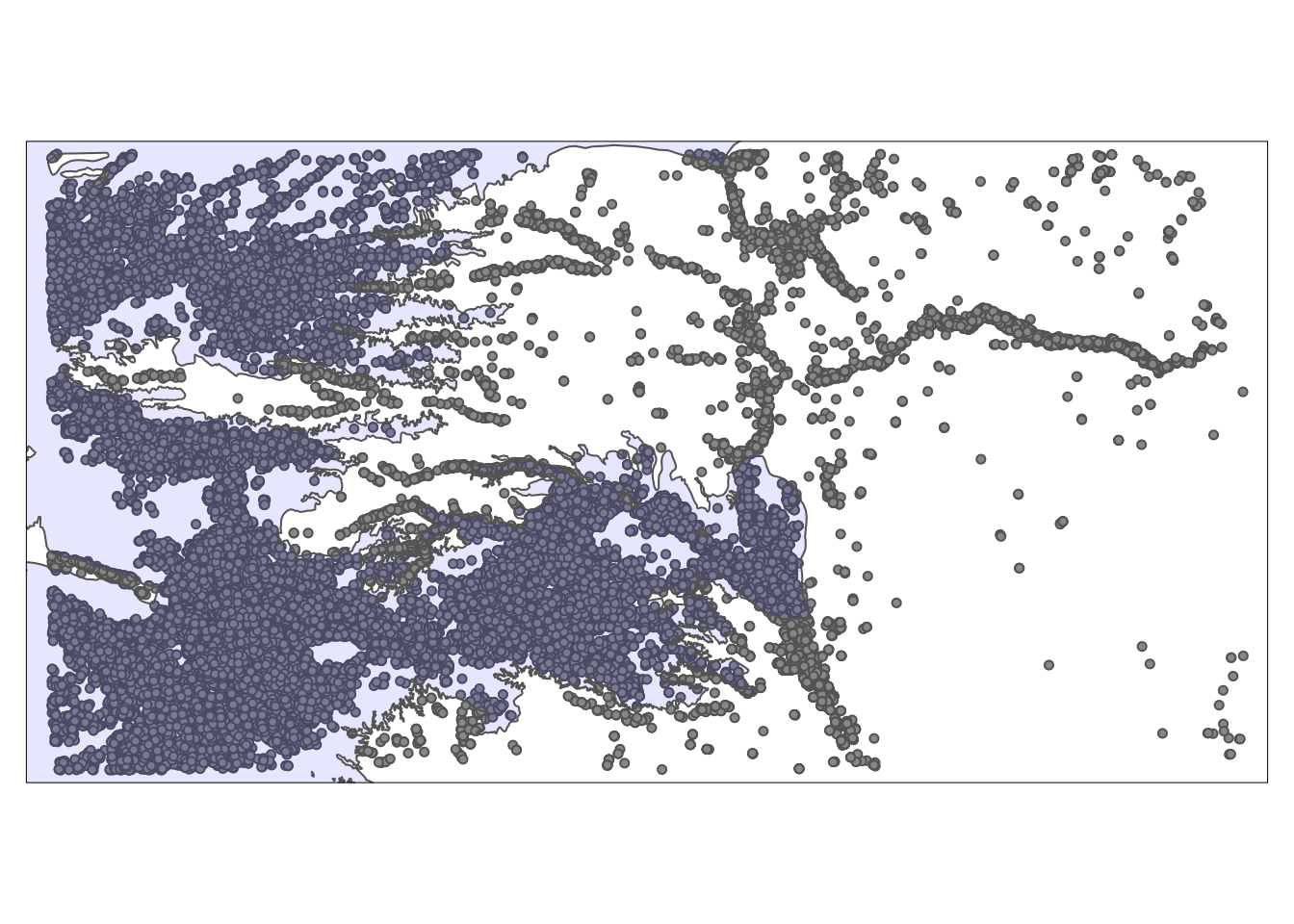


Figure 3.9: A map of Kansas irrigation wells and HPA

We can select only wells that reside within the HPA boundary using the same syntax as the above example.

KS\_wells\_in\_hpa <- KS\_wells[hpa, ]

As you can see in Figure [3.10](spatial-interactions-of-vector-data-subsetting-and-joining.html#fig:map-wells-in-hpa) below, only the wells that are inside (or intersects with) the HPA remained as the default topological relation is st\_intersects().

tm\_shape(KS\_wells\_in\_hpa) +  
 tm\_symbols(size = 0.1) +  
tm\_shape(hpa) +  
 tm\_polygons(col = "blue", alpha = 0.1)

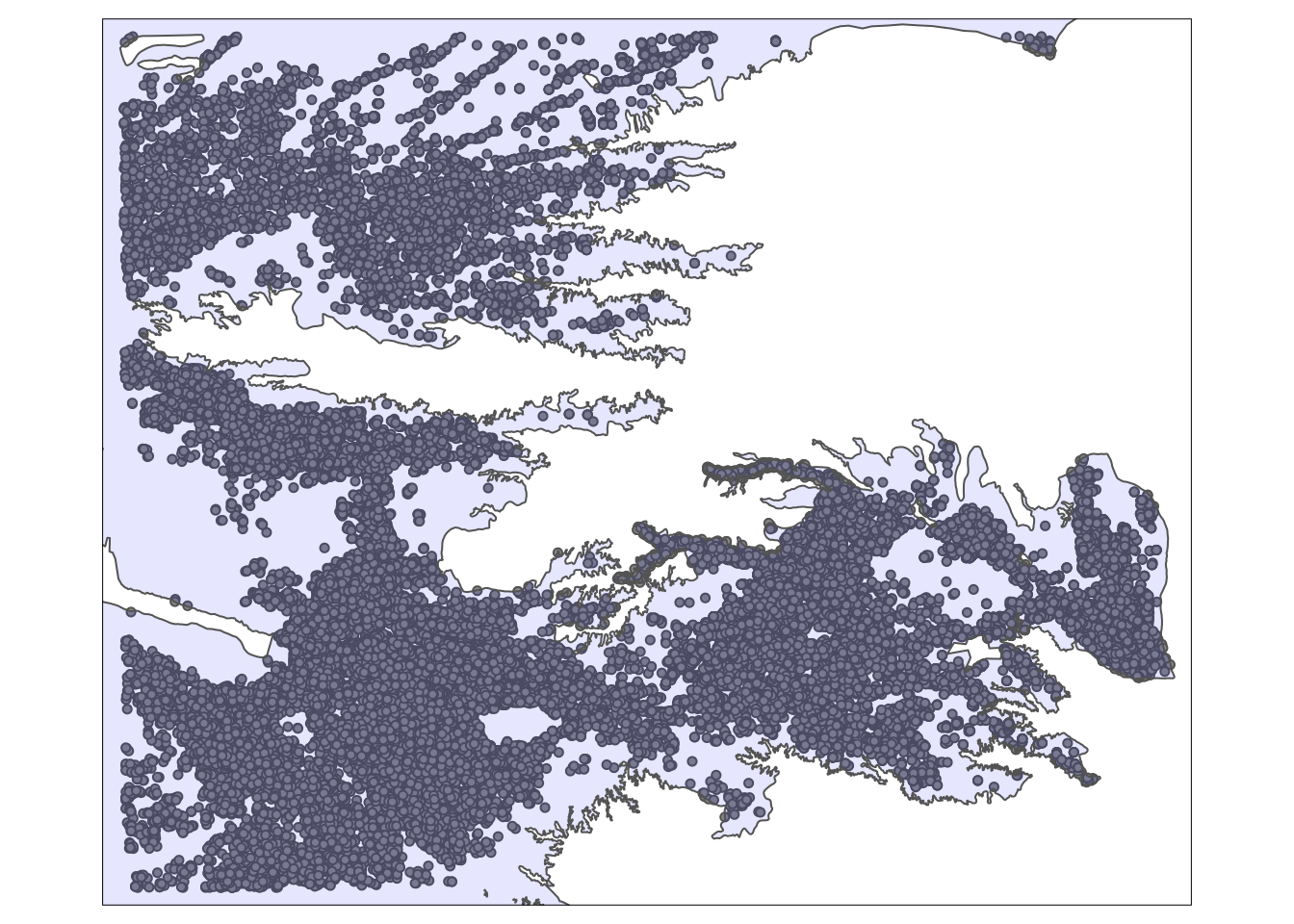


Figure 3.10: A map of Kansas irrigation wells and HPA

### 3.2.3 lines vs polygons

The following map (Figure [3.11](spatial-interactions-of-vector-data-subsetting-and-joining.html#fig:mapl-lines-county)) shows the Kansas counties and U.S. railroads.

ggplot() +  
 geom\_sf(data = rail\_roads, col = "blue") +  
 geom\_sf(data = KS\_counties, fill = NA)

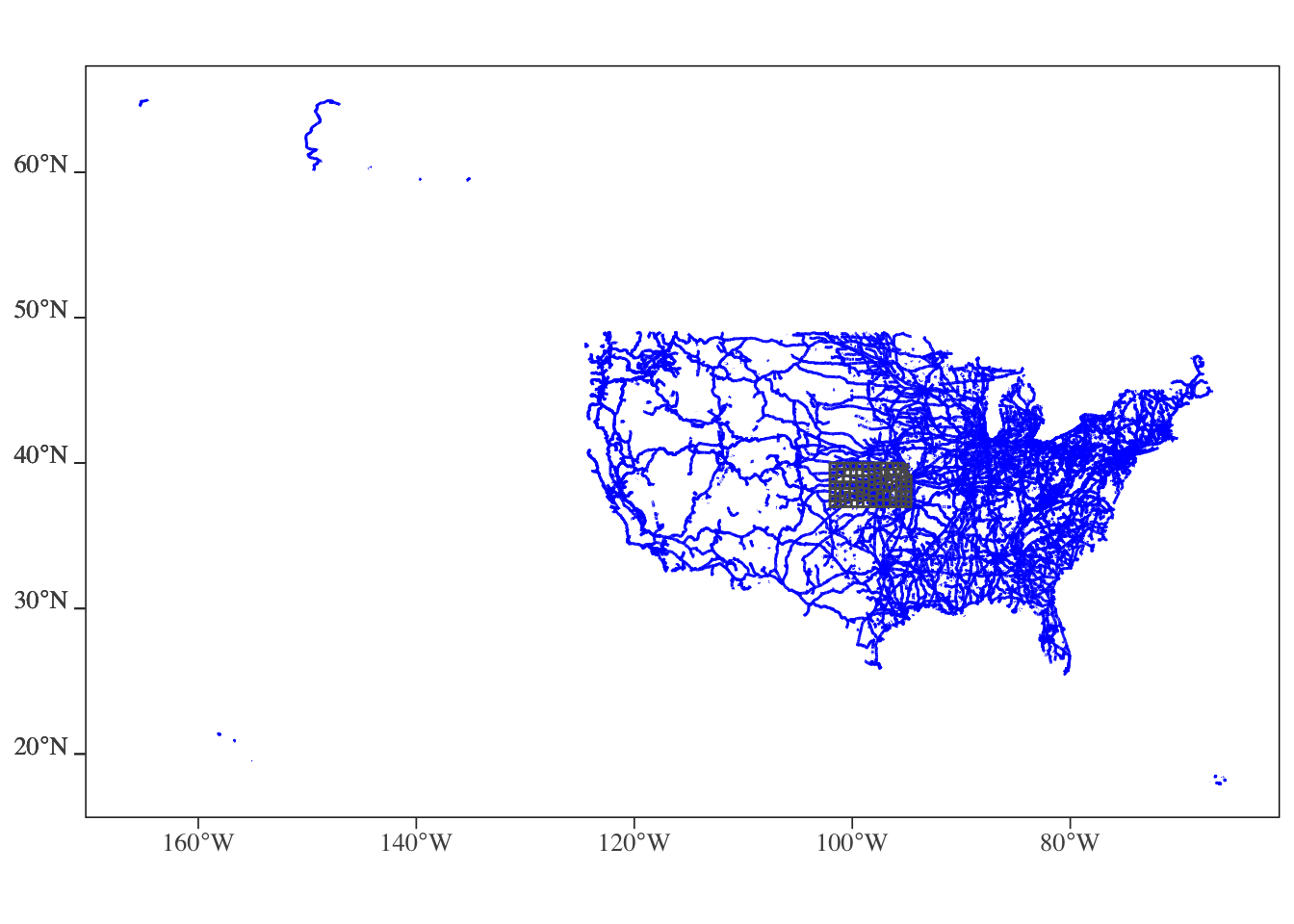


Figure 3.11: U.S. railroads and Kansas county boundary

We can select only railroads that intersects with Kansas.

railroads\_KS <- rail\_roads[KS\_counties, ]

As you can see in Figure [3.12](spatial-interactions-of-vector-data-subsetting-and-joining.html#fig:map-rail-ks) below, only the railroads that intersect with Kansas were selected. Note the the lines that go beyond the Kansas boundary are also selected. Remember, the default is st\_intersect(). If you would like the lines beyond the state boundary to be cut out, but the intersecting parts of those lines to remain, use st\_intersection().

tm\_shape(railroads\_KS) +  
 tm\_lines(col = "blue") +  
tm\_shape(KS\_counties) +  
 tm\_polygons(alpha = 0) +  
 tm\_layout(frame = FALSE)

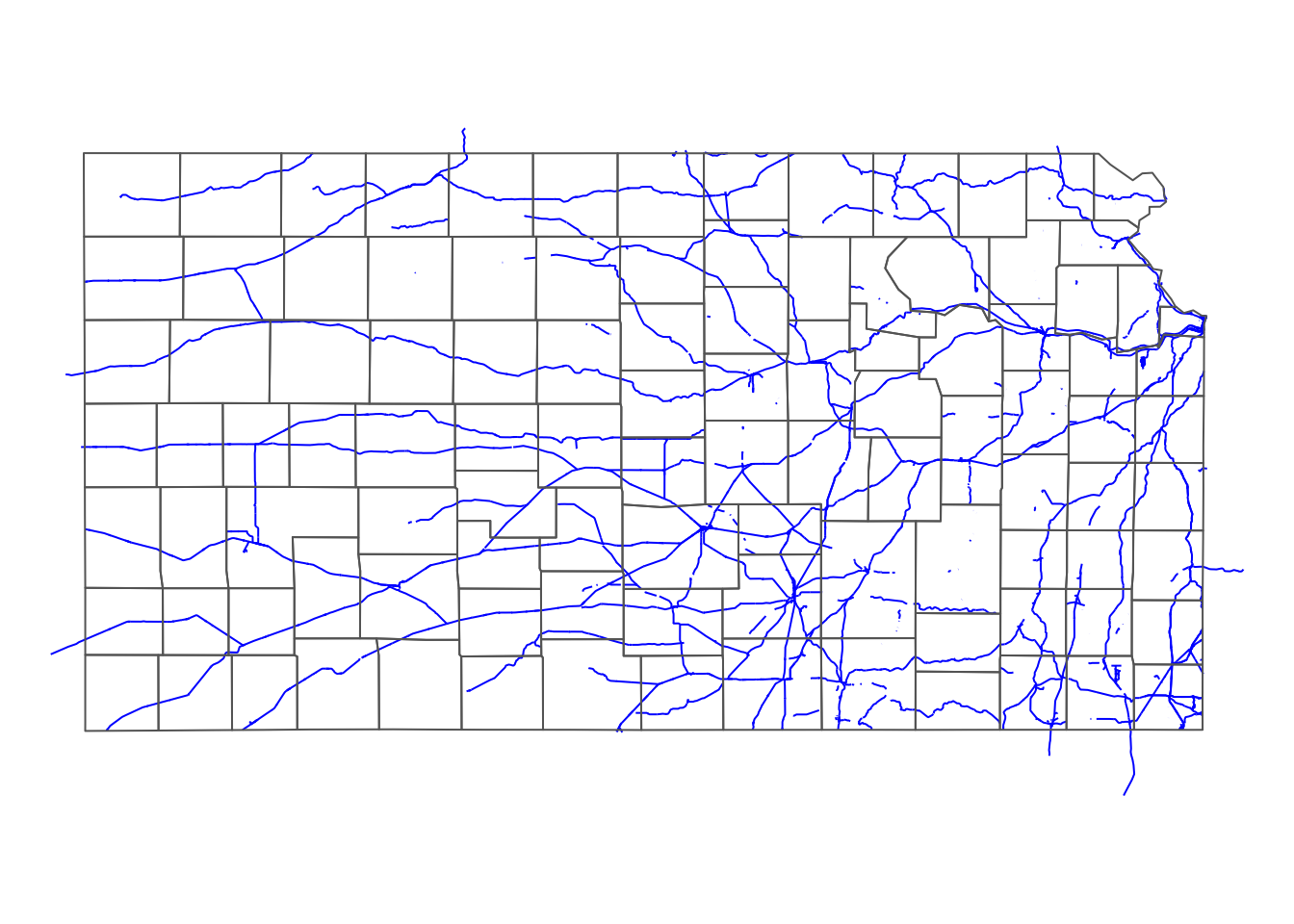


Figure 3.12: Railroads that intersects Kansas county boundary

### 3.2.4 Flagging instead of subsetting

Sometimes, you just want to flag whether two spatial objects intersect or not, instead of dropping non-overlapping observations. In that case, you can use st\_intersects().

**Counties (polygons) against HPA boundary (polygons)**

#--- county ---#  
KS\_counties <- mutate(KS\_counties, intersects\_hpa = st\_intersects(KS\_counties, hpa, sparse = FALSE))  
  
#--- take a look ---#  
dplyr::select(KS\_counties, COUNTYFP, intersects\_hpa)

Simple feature collection with 105 features and 2 fields  
geometry type: MULTIPOLYGON  
dimension: XY  
bbox: xmin: -102.0517 ymin: 36.99308 xmax: -94.59193 ymax: 40.00308  
CRS: EPSG:4269  
First 10 features:  
 COUNTYFP intersects\_hpa geometry  
1 133 FALSE MULTIPOLYGON (((-95.5255 37...  
2 075 TRUE MULTIPOLYGON (((-102.0446 3...  
3 123 FALSE MULTIPOLYGON (((-98.48738 3...  
4 189 TRUE MULTIPOLYGON (((-101.5566 3...  
5 155 TRUE MULTIPOLYGON (((-98.47279 3...  
6 129 TRUE MULTIPOLYGON (((-102.0419 3...  
7 073 FALSE MULTIPOLYGON (((-96.52278 3...  
8 023 TRUE MULTIPOLYGON (((-102.0517 4...  
9 089 TRUE MULTIPOLYGON (((-98.50445 4...  
10 059 FALSE MULTIPOLYGON (((-95.50827 3...

**Wells (points) against HPA boundary (polygons)**

#--- wells ---#  
KS\_wells <- mutate(KS\_wells, in\_hpa = st\_intersects(KS\_wells, hpa, sparse = FALSE))  
  
#--- take a look ---#  
dplyr::select(KS\_wells, site, in\_hpa)

Simple feature collection with 37647 features and 2 fields  
geometry type: POINT  
dimension: XY  
bbox: xmin: -102.0495 ymin: 36.99552 xmax: -94.62089 ymax: 40.00199  
CRS: EPSG:4269  
First 10 features:  
 site in\_hpa geometry  
1 1 TRUE POINT (-100.4423 37.52046)  
2 3 TRUE POINT (-100.7118 39.91526)  
3 5 TRUE POINT (-99.15168 38.48849)  
4 7 TRUE POINT (-101.8995 38.78077)  
5 8 TRUE POINT (-100.7122 38.0731)  
6 9 FALSE POINT (-97.70265 39.04055)  
7 11 TRUE POINT (-101.7114 39.55035)  
8 12 FALSE POINT (-95.97031 39.16121)  
9 15 TRUE POINT (-98.30759 38.26787)  
10 17 TRUE POINT (-100.2785 37.71539)

**U.S. railroads (lines) against Kansas county (polygons)**

Unlike the previous two cases, multiple objects (lines) are checked against multiple objects (polygons) for intersection[67](#fn67). Therefore, we cannot use the strategy we took above of returning a vector of true or false using sparse = TRUE option. Here, we need to count the number of intersecting counties and then assign TRUE if the number is greater than 0.

#--- check the number of intersecting KS counties ---#  
int\_mat <- st\_intersects(rail\_roads, KS\_counties) %>%   
 lapply(length) %>%   
 unlist()   
  
#--- railroads ---#  
rail\_roads <- mutate(rail\_roads, intersect\_ks = int\_mat > 0)  
  
#--- take a look ---#  
dplyr::select(rail\_roads, LINEARID, intersect\_ks)

Simple feature collection with 180958 features and 2 fields  
geometry type: MULTILINESTRING  
dimension: XY  
bbox: xmin: -165.4011 ymin: 17.95174 xmax: -65.74931 ymax: 65.00006  
CRS: 4269  
First 10 features:  
 LINEARID intersect\_ks geometry  
1 11020239500 FALSE MULTILINESTRING ((-79.47058...  
2 11020239501 FALSE MULTILINESTRING ((-79.46687...  
3 11020239502 FALSE MULTILINESTRING ((-79.66819...  
4 11020239503 FALSE MULTILINESTRING ((-79.46687...  
5 11020239504 FALSE MULTILINESTRING ((-79.74031...  
6 11020239575 FALSE MULTILINESTRING ((-79.43695...  
7 11020239576 FALSE MULTILINESTRING ((-79.47852...  
8 11020239577 FALSE MULTILINESTRING ((-79.43695...  
9 11020239589 FALSE MULTILINESTRING ((-79.38736...  
10 11020239591 FALSE MULTILINESTRING ((-79.53848...

## 3.3 Spatial Join

By spatial join, we mean spatial operations that involve all of the followings:

* overlay one spatial layer (target layer) onto another spatial layer (source layer)
* for each of the observation in the target layer
  + identify which objects in the source layer it geographically intersects (or being close) with
  + extract values associated with the intersecting objects in the source layer (and summarize if necessary),
  + assign the extracted value to the object in the target layer

For economists, this is probably the most common motivation of using GIS software, with the ultimate goal being including the spatially joined variables as covariates in regression analysis.

We can classify spatial join into four categories by the type of the underlying spatial objects:

* vector-vector: vector data (target) against vector data (source)
* vector-raster: vector data (target) against raster data (source)
* raster-vector: raster data (target) against vector data (source)
* raster-raster: raster data (target) against raster data (source)

Among the four, our focus here is the first case. The second case will be discussed in Chapter 5. We will not cover the third and fourth cases in this course. This is because it is almost always the case that our target data is a vector data (e.g., city or farm fields as points, political boundaries as polygons, etc).

Category 1 can be further broken down into different sub categories depending on the type of spatial objects (point, line, and polygon). Here, we will ignore any spatial joins that involve lines. This is because objects represented by lines are rarely observations units in econometric analysis nor the source data that we will extract values from.[68](#fn68) So, here is the list of the types of spatial joins we will learn.

1. points (target) against polygons (source)
2. polygons (target) against points (source)
3. polygons (target) against polygons (source)

### 3.3.1 Case 1: points (target) vs polygons (source)

Case 1, for each of the observations (points) in the target data, finds which polygon in the source file it intersects, and then assign the value associated with the polygon to the point[69](#fn69). In order to achieve this, we can use the st\_join() function, whose syntax is as follows:

#--- NOT RUN ---#  
st\_join(target\_sf, source\_sf)

Similar to spatial subsetting, the default topological relation is st\_intersects()[70](#fn70).

We use the KS irrigation wells data (points) and KS county boundary data (polygons) for a demonstration. Our goal is to assign the county-level corn price information from the KS county data to wells. First let me create and add a fake county-level corn price variable to the KS county data.

KS\_corn\_price <- KS\_counties %>%   
 mutate(  
 corn\_price = seq(3.2, 3.9, length = nrow(.))   
 ) %>%   
 dplyr::select(COUNTYFP, corn\_price)

Here is the map of KS county color-differentiated by fake corn price (Figure [3.13](spatial-interactions-of-vector-data-subsetting-and-joining.html#fig:map-corn-price)):

tm\_shape(KS\_corn\_price) +   
 tm\_polygons(col = "corn\_price") +  
 tm\_layout(frame = FALSE, legend.outside = TRUE)

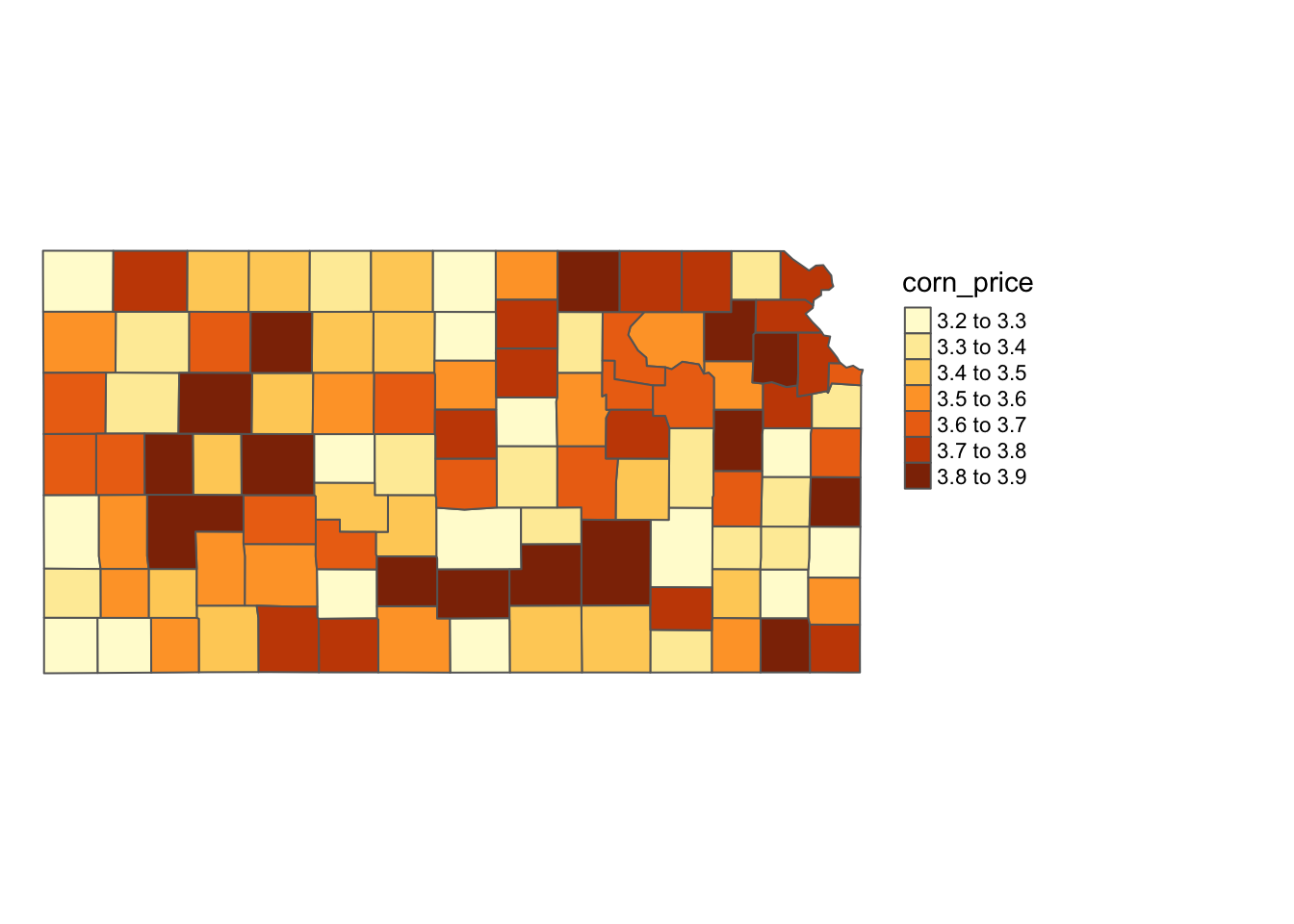


Figure 3.13: Map of county-level fake corn price

For this particular context, the following code will do the job:

#--- spatial join ---#  
(  
KS\_wells\_County <- st\_join(KS\_wells, KS\_corn\_price)  
)

Simple feature collection with 37647 features and 5 fields  
geometry type: POINT  
dimension: XY  
bbox: xmin: -102.0495 ymin: 36.99552 xmax: -94.62089 ymax: 40.00199  
CRS: EPSG:4269  
First 10 features:  
 site af\_used in\_hpa COUNTYFP corn\_price geometry  
1 1 232.099948 TRUE 069 3.556731 POINT (-100.4423 37.52046)  
2 3 13.183940 TRUE 039 3.449038 POINT (-100.7118 39.91526)  
3 5 99.187052 TRUE 165 3.287500 POINT (-99.15168 38.48849)  
4 7 0.000000 TRUE 199 3.644231 POINT (-101.8995 38.78077)  
5 8 145.520499 TRUE 055 3.832692 POINT (-100.7122 38.0731)  
6 9 3.614535 FALSE 143 3.799038 POINT (-97.70265 39.04055)  
7 11 188.423543 TRUE 181 3.590385 POINT (-101.7114 39.55035)  
8 12 77.335960 FALSE 177 3.550000 POINT (-95.97031 39.16121)  
9 15 0.000000 TRUE 159 3.610577 POINT (-98.30759 38.26787)  
10 17 167.819034 TRUE 069 3.556731 POINT (-100.2785 37.71539)

You can see from Figure [3.14](spatial-interactions-of-vector-data-subsetting-and-joining.html#fig:map-corn-wells) below that all the wells inside the same county has the same corn price value.

tm\_shape(KS\_counties) +  
 tm\_polygons() +  
tm\_shape(KS\_wells\_County) +  
 tm\_symbols(col = "corn\_price", size = 0.1) +  
 tm\_layout(frame = FALSE, legend.outside = TRUE)

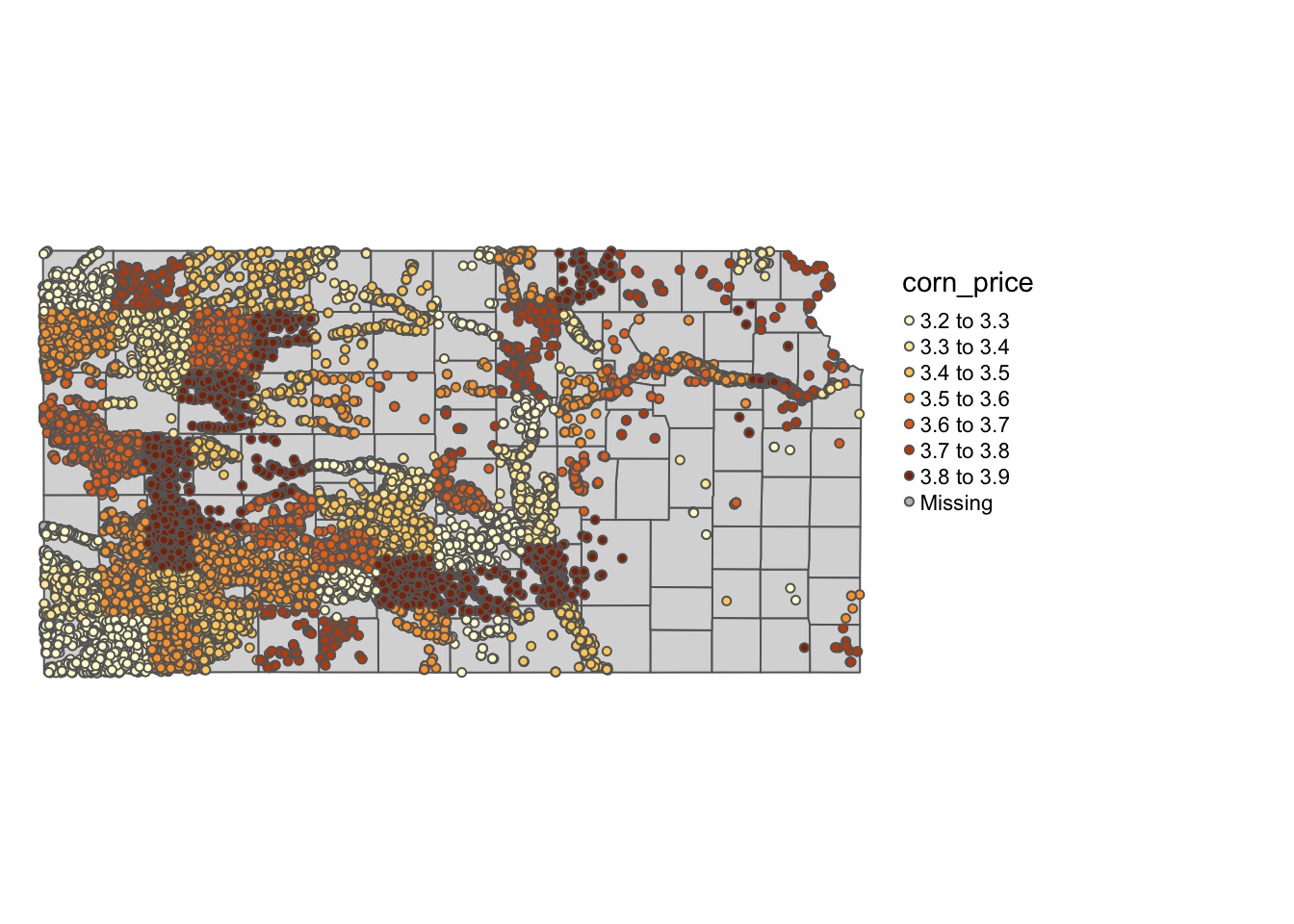


Figure 3.14: Map of wells color-differentiated by corn price

### 3.3.2 Case 2: polygons (target) vs points (source)

Case 2, for each of the observations (polygons) in the target data, find which observations (points) in the source file it intersects, and then assign the values associated with the points to the polygon. We use the same function: st\_join()[71](#fn71).

Suppose you are now interested in county-level analysis and you would like to get county-level total groundwater pumping. The target file is KS\_counties, and the source file is KS\_wells.

#--- spatial join ---#  
KS\_County\_wells <- st\_join(KS\_counties, KS\_wells)  
  
#--- take a look ---#  
dplyr::select(KS\_County\_wells, COUNTYFP, site, af\_used)

Simple feature collection with 37652 features and 3 fields  
geometry type: MULTIPOLYGON  
dimension: XY  
bbox: xmin: -102.0517 ymin: 36.99308 xmax: -94.59193 ymax: 40.00308  
CRS: EPSG:4269  
First 10 features:  
 COUNTYFP site af\_used geometry  
1 133 53861 17.01790 MULTIPOLYGON (((-95.5255 37...  
1.1 133 70592 0.00000 MULTIPOLYGON (((-95.5255 37...  
2 075 328 394.04513 MULTIPOLYGON (((-102.0446 3...  
2.1 075 336 80.65036 MULTIPOLYGON (((-102.0446 3...  
2.2 075 436 568.25359 MULTIPOLYGON (((-102.0446 3...  
2.3 075 1007 215.80416 MULTIPOLYGON (((-102.0446 3...  
2.4 075 1170 0.00000 MULTIPOLYGON (((-102.0446 3...  
2.5 075 1192 77.39120 MULTIPOLYGON (((-102.0446 3...  
2.6 075 1249 0.00000 MULTIPOLYGON (((-102.0446 3...  
2.7 075 1300 320.22612 MULTIPOLYGON (((-102.0446 3...

As you can see, in the resulting dataset, all the unique polygon - point intersecting combinations comprise the observations. For each of the polygons, you will have as many observations as the number of wells that intersect with the polygon. Once you joined the two layers, you can find statistics by polygon (county here). Since we want groundwater extraction by county, the following does the job.

KS\_County\_wells %>%   
 group\_by(COUNTYFP) %>%   
 summarize(af\_used = sum(af\_used, na.rm = TRUE))

Simple feature collection with 105 features and 2 fields  
geometry type: MULTIPOLYGON  
dimension: XY  
bbox: xmin: -102.0517 ymin: 36.99308 xmax: -94.59193 ymax: 40.00308  
CRS: EPSG:4269  
# A tibble: 105 x 3  
 COUNTYFP af\_used geometry  
 <fct> <dbl> <MULTIPOLYGON [°]>  
 1 001 0 (((-95.51931 37.82026, -95.51897 38.03823, -95.07788 38.037…  
 2 003 0 (((-95.50833 38.39028, -95.06583 38.38994, -95.07788 38.037…  
 3 005 771. (((-95.56413 39.65287, -95.33974 39.65298, -95.11519 39.652…  
 4 007 4972. (((-99.0126 37.47042, -98.46466 37.47101, -98.46493 37.3841…  
 5 009 61083. (((-99.03297 38.69676, -98.48611 38.69688, -98.47991 38.681…  
 6 011 0 (((-95.08808 37.73248, -95.07969 37.8198, -95.07788 38.0377…  
 7 013 480. (((-95.78811 40.00047, -95.78457 40.00046, -95.3399 40.0000…  
 8 015 343. (((-97.15248 37.91273, -97.15291 38.0877, -96.84077 38.0856…  
 9 017 0 (((-96.83765 38.34864, -96.81951 38.52245, -96.35378 38.521…  
10 019 0 (((-96.52487 37.30273, -95.9644 37.29923, -95.96427 36.9992…  
# … with 95 more rows

Of course, it is just as easy to get other types of statistics by simply modifying the summarize() part.

However, this two-step process can be actually done in one step using aggregate(), in which you specify how you want to aggregate with the FUN option as follows:

#--- mean ---#  
aggregate(KS\_wells, KS\_counties, FUN = mean)

Simple feature collection with 105 features and 3 fields  
geometry type: MULTIPOLYGON  
dimension: XY  
bbox: xmin: -102.0517 ymin: 36.99308 xmax: -94.59193 ymax: 40.00308  
CRS: EPSG:4269  
First 10 features:  
 site af\_used in\_hpa geometry  
1 62226.50 8.508950 0.0000000 MULTIPOLYGON (((-95.5255 37...  
2 35184.64 176.390742 0.4481793 MULTIPOLYGON (((-102.0446 3...  
3 40086.82 35.465123 0.0000000 MULTIPOLYGON (((-98.48738 3...  
4 40179.41 285.672916 1.0000000 MULTIPOLYGON (((-101.5566 3...  
5 51249.39 46.048048 0.9743783 MULTIPOLYGON (((-98.47279 3...  
6 33033.13 202.612377 1.0000000 MULTIPOLYGON (((-102.0419 3...  
7 29840.40 0.000000 0.0000000 MULTIPOLYGON (((-96.52278 3...  
8 28235.82 94.585634 0.9736842 MULTIPOLYGON (((-102.0517 4...  
9 36180.06 44.033911 0.3000000 MULTIPOLYGON (((-98.50445 4...  
10 40016.00 1.142775 0.0000000 MULTIPOLYGON (((-95.50827 3...

#--- sum ---#  
aggregate(KS\_wells, KS\_counties, FUN = sum)

Simple feature collection with 105 features and 3 fields  
geometry type: MULTIPOLYGON  
dimension: XY  
bbox: xmin: -102.0517 ymin: 36.99308 xmax: -94.59193 ymax: 40.00308  
CRS: EPSG:4269  
First 10 features:  
 site af\_used in\_hpa geometry  
1 124453 1.701790e+01 0 MULTIPOLYGON (((-95.5255 37...  
2 12560917 6.297149e+04 160 MULTIPOLYGON (((-102.0446 3...  
3 1964254 1.737791e+03 0 MULTIPOLYGON (((-98.48738 3...  
4 42389277 3.013849e+05 1055 MULTIPOLYGON (((-101.5566 3...  
5 68007942 6.110576e+04 1293 MULTIPOLYGON (((-98.47279 3...  
6 15756801 9.664610e+04 477 MULTIPOLYGON (((-102.0419 3...  
7 149202 0.000000e+00 0 MULTIPOLYGON (((-96.52278 3...  
8 17167377 5.750807e+04 592 MULTIPOLYGON (((-102.0517 4...  
9 1809003 2.201696e+03 15 MULTIPOLYGON (((-98.50445 4...  
10 160064 4.571102e+00 0 MULTIPOLYGON (((-95.50827 3...

Notice that the mean() function was applied to all the columns in KS\_wells, including site id number. So, you might want to select variables you want to join before you apply the aggregate() function like this:

aggregate(dplyr::select(KS\_wells, af\_used), KS\_counties, FUN = mean)

Simple feature collection with 105 features and 1 field  
geometry type: MULTIPOLYGON  
dimension: XY  
bbox: xmin: -102.0517 ymin: 36.99308 xmax: -94.59193 ymax: 40.00308  
CRS: EPSG:4269  
First 10 features:  
 af\_used geometry  
1 8.508950 MULTIPOLYGON (((-95.5255 37...  
2 176.390742 MULTIPOLYGON (((-102.0446 3...  
3 35.465123 MULTIPOLYGON (((-98.48738 3...  
4 285.672916 MULTIPOLYGON (((-101.5566 3...  
5 46.048048 MULTIPOLYGON (((-98.47279 3...  
6 202.612377 MULTIPOLYGON (((-102.0419 3...  
7 0.000000 MULTIPOLYGON (((-96.52278 3...  
8 94.585634 MULTIPOLYGON (((-102.0517 4...  
9 44.033911 MULTIPOLYGON (((-98.50445 4...  
10 1.142775 MULTIPOLYGON (((-95.50827 3...

### 3.3.3 Case 3: polygons (target) vs polygons (source)

For this case, st\_join(target\_sf, source\_sf) will return all the unique intersecting polygon-polygon combinations with the information of the polygon from source\_sf attached.

We will use county-level corn acres in Iowa in 2018 from USDA NASS[72](#fn72) and Hydrologic Units[73](#fn73) Our objective here is to find corn acres by HUC units based on the county-level corn acres data[74](#fn74).

We first import the Iowa corn acre data:

#--- IA boundary ---#  
IA\_corn <- readRDS("./Data/IA\_corn.rds")  
  
#--- take a look ---#  
IA\_corn

Simple feature collection with 93 features and 3 fields  
geometry type: MULTIPOLYGON  
dimension: XY  
bbox: xmin: 203228.6 ymin: 4470941 xmax: 736832.9 ymax: 4822687  
CRS: EPSG:26915  
First 10 features:  
 county\_code year acres geometry  
1 083 2018 183500 MULTIPOLYGON (((458997 4711...  
2 141 2018 167000 MULTIPOLYGON (((267700.8 47...  
3 081 2018 184500 MULTIPOLYGON (((421231.2 47...  
4 019 2018 189500 MULTIPOLYGON (((575285.6 47...  
5 023 2018 165500 MULTIPOLYGON (((497947.5 47...  
6 195 2018 111500 MULTIPOLYGON (((459791.6 48...  
7 063 2018 110500 MULTIPOLYGON (((345214.3 48...  
8 027 2018 183000 MULTIPOLYGON (((327408.5 46...  
9 121 2018 70000 MULTIPOLYGON (((396378.1 45...  
10 077 2018 107000 MULTIPOLYGON (((355180.1 46...

Here is the map of IA county color-differentiated by corn acres (Figure [3.15](spatial-interactions-of-vector-data-subsetting-and-joining.html#fig:map-IA-corn)):

#--- here is the map ---#  
tm\_shape(IA\_corn) +  
 tm\_polygons(col = "acres") +  
 tm\_layout(frame = FALSE, legend.outside = TRUE)

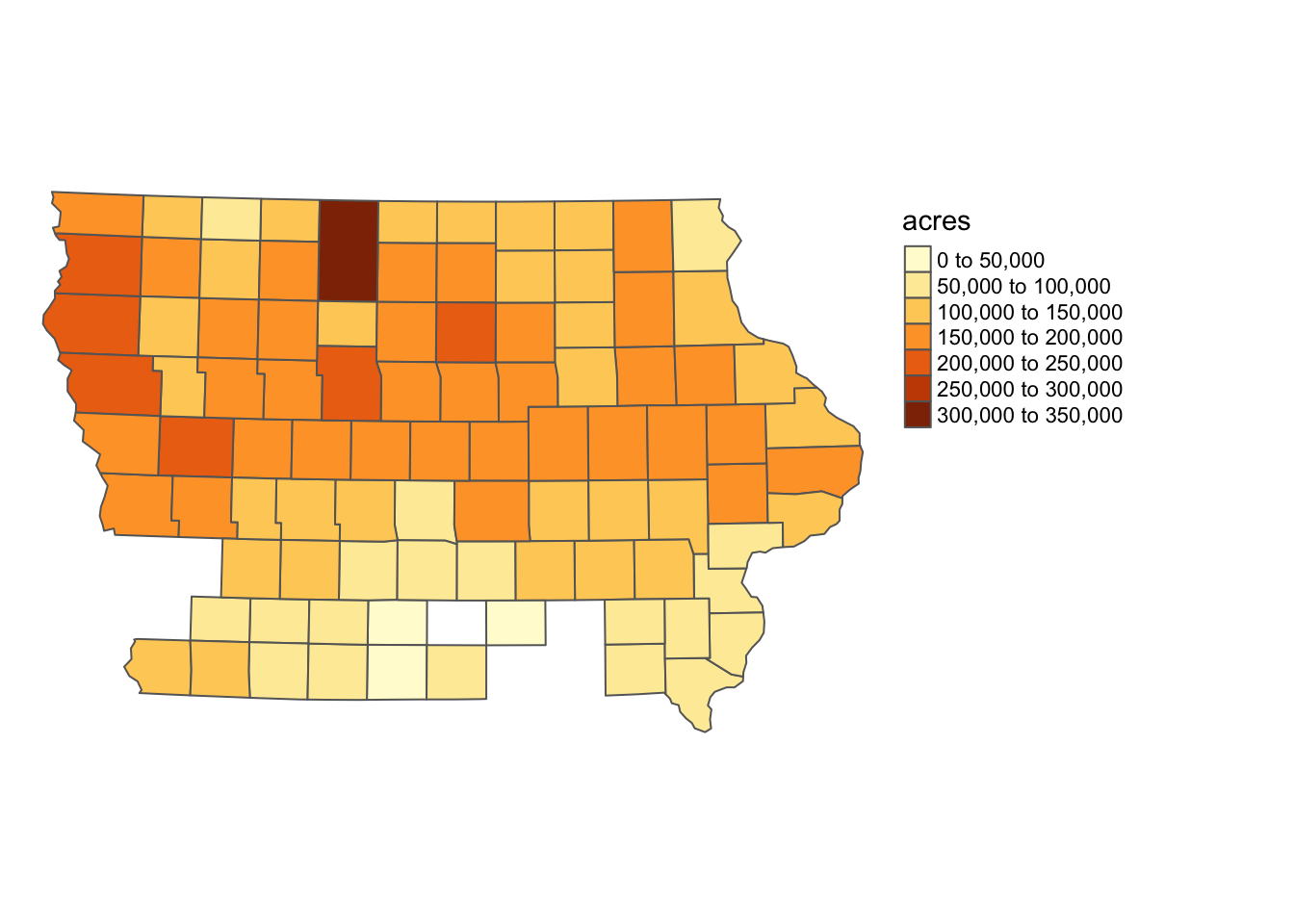


Figure 3.15: Map of Iowa counties color-differentiated by corn planted acreage

Now import the HUC units data:

#--- import HUC units ---#  
HUC\_IA <- st\_read(dsn = "./Data/huc250k\_shp", layer = "huc250k") %>%   
 dplyr::select(HUC\_CODE) %>%   
 #--- reproject to the CRS of IA ---#  
 st\_transform(st\_crs(IA\_corn)) %>%   
 #--- select HUC units that overlaps with IA ---#  
 .[IA\_corn, ]

Here is the map of HUC units (Figure [3.16](spatial-interactions-of-vector-data-subsetting-and-joining.html#fig:HUC-map)):

tm\_shape(HUC\_IA) +  
 tm\_polygons() +  
 tm\_layout(frame = FALSE, legend.outside = TRUE)



Figure 3.16: Map of HUC units that intersect with Iowa state boundary

IA county with HUC units superimposed on top (Figure [3.17](spatial-interactions-of-vector-data-subsetting-and-joining.html#fig:HUC-county-map)):

tm\_shape(IA\_corn) +  
 tm\_polygons(col = "acres") +  
tm\_shape(HUC\_IA) +  
 tm\_polygons(alpha = 0) +  
 tm\_layout(frame = FALSE, legend.outside = TRUE)

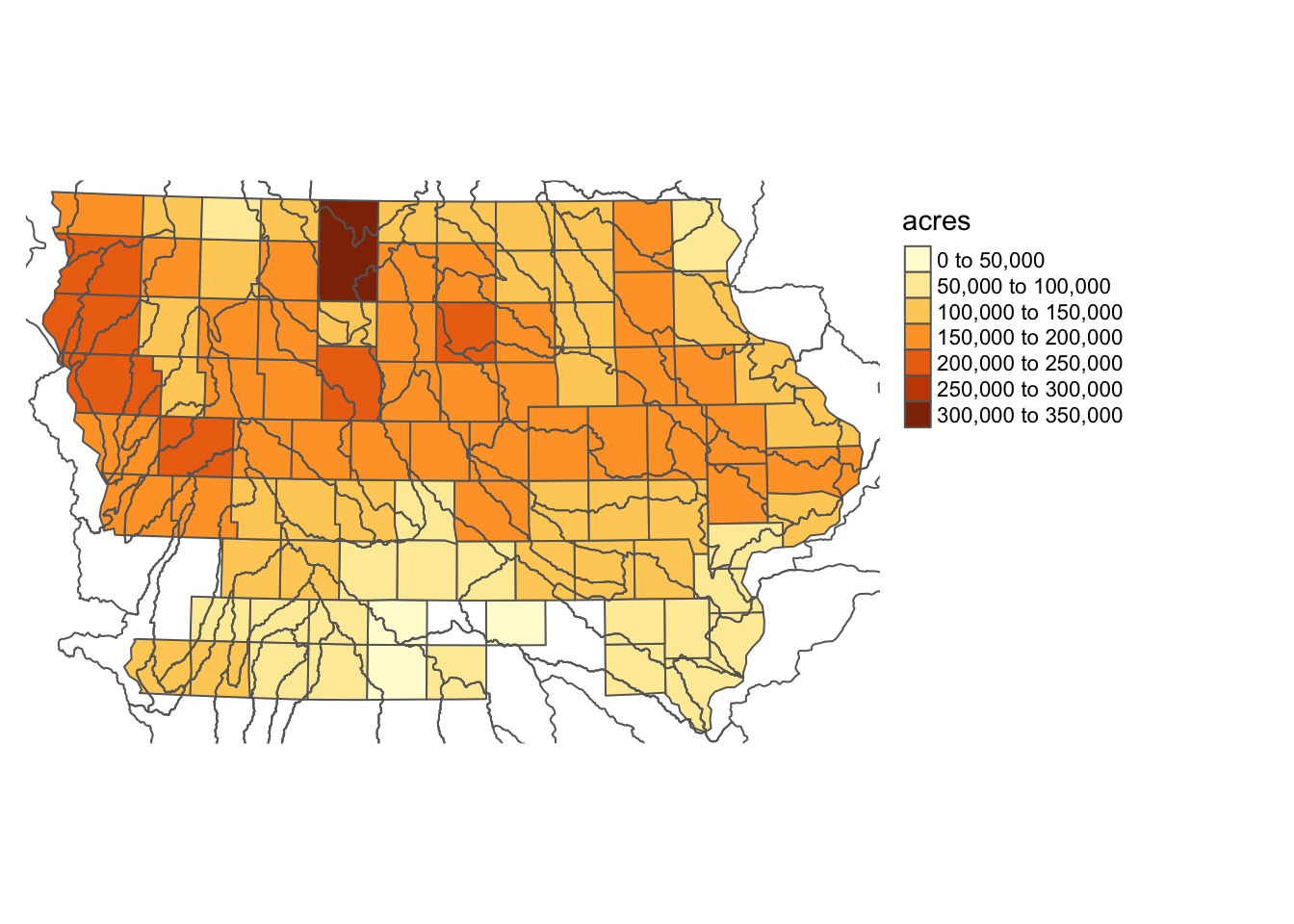


Figure 3.17: Map of HUC units superimposed on Iowas counties

Spatial joining will produce the following.

(  
HUC\_joined <- st\_join(HUC\_IA, IA\_corn)  
)

Simple feature collection with 349 features and 4 fields  
geometry type: POLYGON  
dimension: XY  
bbox: xmin: 154970 ymin: 4346324 xmax: 773307 ymax: 4907737  
CRS: EPSG:26915  
First 10 features:  
 HUC\_CODE county\_code year acres geometry  
608 10170203 149 2018 226500 POLYGON ((235577 4907515, 2...  
608.1 10170203 167 2018 249000 POLYGON ((235577 4907515, 2...  
608.2 10170203 193 2018 201000 POLYGON ((235577 4907515, 2...  
608.3 10170203 119 2018 184500 POLYGON ((235577 4907515, 2...  
621 07020009 063 2018 110500 POLYGON ((408600.2 4880800,...  
621.1 07020009 109 2018 304000 POLYGON ((408600.2 4880800,...  
621.2 07020009 189 2018 120000 POLYGON ((408600.2 4880800,...  
627 10170204 141 2018 167000 POLYGON ((248140.3 4891654,...  
627.1 10170204 143 2018 116000 POLYGON ((248140.3 4891654,...  
627.2 10170204 167 2018 249000 POLYGON ((248140.3 4891654,...

Each of the intersecting HUC-county combinations becomes an observation with its resulting geometry same as the geometry of the HUC unit. To see this, let’s take a look at one of the HUC units.

The HUC unit with HUC\_CODE ==10170203 intersects with four County.

#--- get the HUC unit with `HUC\_CODE ==10170203` ---#  
(  
temp\_HUC\_county <- filter(HUC\_joined, HUC\_CODE == 10170203)  
)

Simple feature collection with 4 features and 4 fields  
geometry type: POLYGON  
dimension: XY  
bbox: xmin: 154970 ymin: 4709628 xmax: 248140.3 ymax: 4907737  
CRS: EPSG:26915  
 HUC\_CODE county\_code year acres geometry  
1 10170203 149 2018 226500 POLYGON ((235577 4907515, 2...  
2 10170203 167 2018 249000 POLYGON ((235577 4907515, 2...  
3 10170203 193 2018 201000 POLYGON ((235577 4907515, 2...  
4 10170203 119 2018 184500 POLYGON ((235577 4907515, 2...

Figure [3.18](spatial-interactions-of-vector-data-subsetting-and-joining.html#fig:four-county-huc) shows the map of the four observations.

tm\_shape(temp\_HUC\_county) +  
 tm\_polygons() +  
 tm\_layout(frame = FALSE)



Figure 3.18: Map of the HUC unit

So, all of the four observations have the identical geometry, which is the geometry of the HUC unit, meaning that the st\_join() did not leave the information about the nature of the intersection of the HUC unit and the four county. Again, remember that the default option is st\_intersects(), which checks whether spatial objects intersect or not, nothing more. If you are just calculating the simple average of corn acres ignoring the degree of spatial overlaps, this is just fine. However, if you would like to calculate area-weighted average, you are not left with sufficient information.

To find an area-weighted average, we can use st\_intersection(). For each of the polygons in the target layer, this function, finds the intersecting polygons from the source data, and then divide the target polygon into parts based on the boundary of the intersecting polygons.

(  
HUC\_intersections <- st\_intersection(HUC\_IA, IA\_corn) %>%   
 mutate(huc\_county = paste0(HUC\_CODE, "-", county\_code))  
)

Simple feature collection with 349 features and 5 fields  
geometry type: GEOMETRY  
dimension: XY  
bbox: xmin: 203228.6 ymin: 4470941 xmax: 736832.9 ymax: 4822687  
CRS: EPSG:26915  
First 10 features:  
 HUC\_CODE county\_code year acres geometry huc\_county  
1 07080207 083 2018 183500 POLYGON ((482916.4 4711686,... 07080207-083  
2 07080205 083 2018 183500 POLYGON ((499779.4 4696836,... 07080205-083  
3 07080105 083 2018 183500 POLYGON ((461846.1 4683469,... 07080105-083  
4 10170204 141 2018 167000 POLYGON ((269432.3 4793329,... 10170204-141  
5 10230003 141 2018 167000 POLYGON ((271607.5 4754542,... 10230003-141  
6 10230002 141 2018 167000 POLYGON ((267630 4790936, 2... 10230002-141  
7 07100003 081 2018 184500 POLYGON ((436142.9 4789503,... 07100003-081  
8 07080203 081 2018 184500 MULTIPOLYGON (((459473.3 47... 07080203-081  
9 07080207 081 2018 184500 POLYGON ((429601.9 4779600,... 07080207-081  
10 07100005 081 2018 184500 POLYGON ((420999.1 4772191,... 07100005-081

The key difference from the st\_join() example is that each observation of the returned data is a unique HUC-county intersection. Figure [3.19](spatial-interactions-of-vector-data-subsetting-and-joining.html#fig:inter-ex) below is a map of all the intersections of the HUC unit with HUC\_CODE ==10170203 and the four intersecting county.

tm\_shape(filter(HUC\_intersections, HUC\_CODE == "10170203")) +   
 tm\_polygons(col = "huc\_county") +  
 tm\_layout(frame = FALSE)

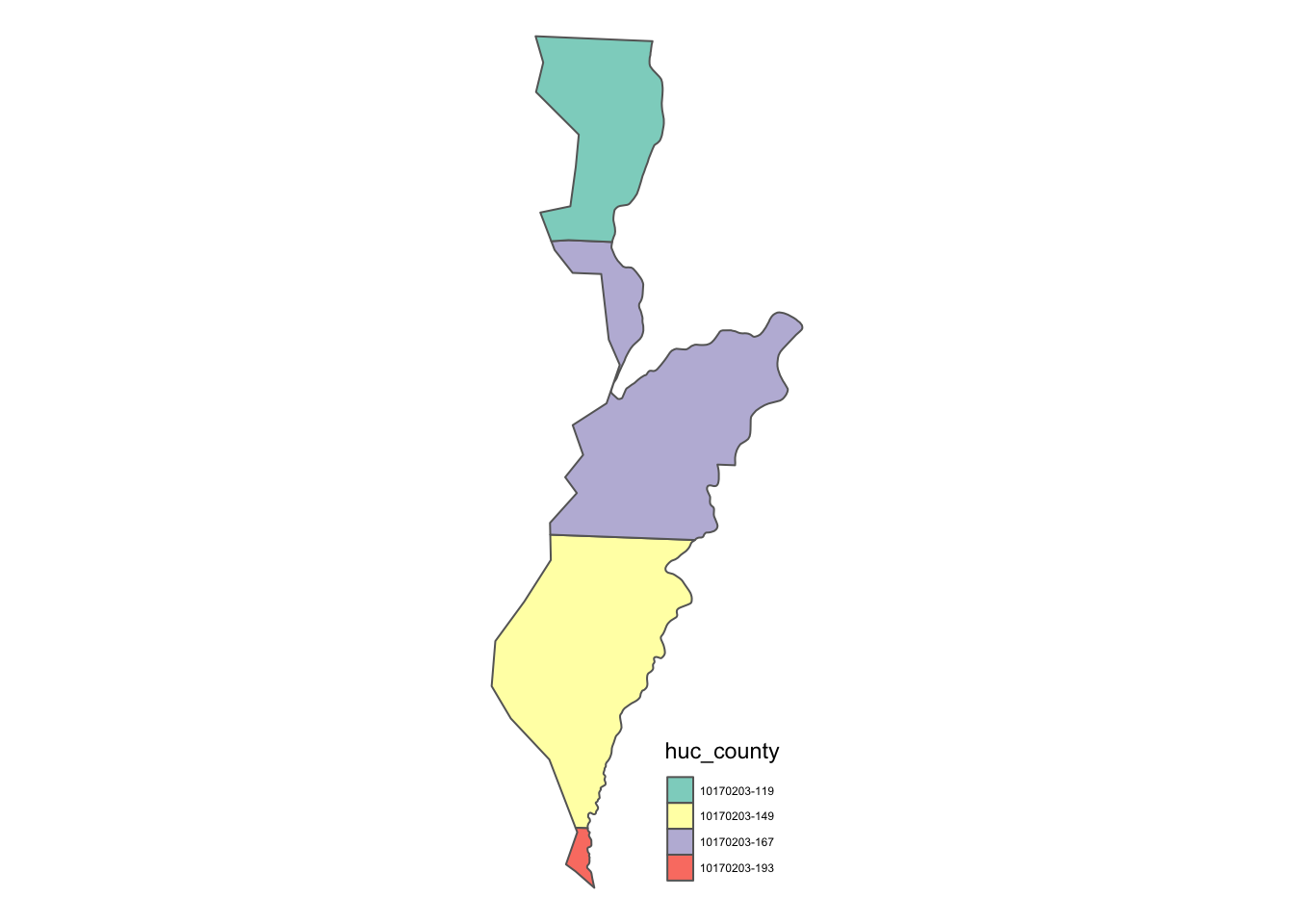


Figure 3.19: Intersections of a HUC unit and Iowa counties

Note also that the attributes of county data are joined as you can see acres in the output above. So, st\_intersection() is really a spatial kind of spatial join where the resulting observations are the intersections of the target and source sf objects.

In order to find the area-weighted average of corn acres, you can use st\_area() first to calculate the area of the intersections, and then find the area-weighted average as follows:

(  
HUC\_aw\_acres <- HUC\_intersections %>%   
 #--- get area ---#  
 mutate(area = as.numeric(st\_area(.))) %>%   
 #--- get area-weight by HUC unit ---#  
 group\_by(HUC\_CODE) %>%   
 mutate(weight = area / sum(area)) %>%   
 #--- calculate area-weighted corn acreage by HUC unit ---#  
 summarize(aw\_acres = sum(weight \* acres))  
)

Simple feature collection with 55 features and 2 fields  
geometry type: GEOMETRY  
dimension: XY  
bbox: xmin: 203228.6 ymin: 4470941 xmax: 736832.9 ymax: 4822687  
CRS: EPSG:26915  
# A tibble: 55 x 3  
 HUC\_CODE aw\_acres geometry  
 <chr> <dbl> <GEOMETRY [m]>  
 1 07020009 251140. POLYGON ((421317.4 4797758, 421179.2 4797632, 421079.3 479…  
 2 07040008 165000 POLYGON ((602943.6 4817205, 602935.1 4817167, 602875.1 481…  
 3 07060001 105224. MULTIPOLYGON (((631611.9 4817707, 631609.2 4817706, 631519…  
 4 07060002 140192. POLYGON ((593286.7 4817067, 593403.8 4817047, 593583.8 481…  
 5 07060003 149000 MULTIPOLYGON (((646504.9 4762382, 646518 4762383, 646554.8…  
 6 07060004 162121. POLYGON ((653200.4 4718423, 652967.7 4718504, 652457.7 471…  
 7 07060005 142428. POLYGON ((735347.8 4642385, 734779.1 4642296, 734459 46422…  
 8 07060006 159628. POLYGON ((692755.3 4694862, 692788.3 4694758, 692788.3 469…  
 9 07080101 115574. POLYGON ((667472 4558778, 667391.8 4558691, 667221.7 45585…  
10 07080102 160017. POLYGON ((635032.8 4675786, 635247.8 4675644, 635367.8 467…  
# … with 45 more rows

* I would say it is very rare that you use other topological relations like st\_within() or st\_touches().[↩](spatial-interactions-of-vector-data-subsetting-and-joining.html#fnref63)
* Run ?geos\_binary\_pred to see other topological relations you can find.[↩](spatial-interactions-of-vector-data-subsetting-and-joining.html#fnref64)
* See Chapter 1, Demonstration 3 for an example of lines-polygons intersection in an economic study.[↩](spatial-interactions-of-vector-data-subsetting-and-joining.html#fnref65)
* This function can be useful to identify neighbors. For example, you may want to find irrigation wells located around well \(i\) to label them as well \(i\)’s neighbor.[↩](spatial-interactions-of-vector-data-subsetting-and-joining.html#fnref66)
* Of course, this situation arises for a polygons-polygons case as well. The above polygons-polygons example was an exception because the hpa has only one polygon object.[↩](spatial-interactions-of-vector-data-subsetting-and-joining.html#fnref67)
* Note that we did not extract any attribute values of railroads in Chapter 1, Demonstration 4. We just calculated the travel length of the railroads, which does not fall under our definition of spatial join.[↩](spatial-interactions-of-vector-data-subsetting-and-joining.html#fnref68)
* You can see a practical example of this case in action in Demonstration 1 of Chapter X.[↩](spatial-interactions-of-vector-data-subsetting-and-joining.html#fnref69)
* While it is unlikely you face the need to change the topological relation, you could do so using the join option.[↩](spatial-interactions-of-vector-data-subsetting-and-joining.html#fnref70)
* You can see a practical example of this case in action in Demonstration 2 of Chapter X.[↩](spatial-interactions-of-vector-data-subsetting-and-joining.html#fnref71)
* see [here](link_here) for how to download Quick Stats data from within R.[↩](spatial-interactions-of-vector-data-subsetting-and-joining.html#fnref72)
* see [here](https://water.usgs.gov/GIS/huc.html) for explanation of what they are. You do not really need to know what HUC units are to understand what’s done in this section.[↩](spatial-interactions-of-vector-data-subsetting-and-joining.html#fnref73)
* Yes, there will be substantial measurement errors as the source polygons (corn acres by county) are large relative to the target polygons (HUC units). But, this serves as a good illustration of a polygon-polygon join.[↩](spatial-interactions-of-vector-data-subsetting-and-joining.html#fnref74)