PS10 Spatial Data

PS 10 — Gerrymandering

STAT 133

Practice related to this lecture is found in the worksheet associated with Project: Gerrymandering.

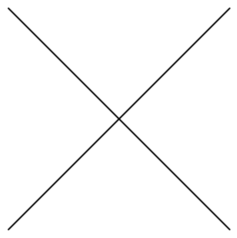
Spatial II

The following exercises serve as an introduction to the functionality of the **sf** package. Anytime you use a new function, use `?` to learn how it works. Some of these exercises ask you to plot the result — plots are provided to check your answer.

1. Following the approach used in lecture, use `st_linestring()` to create two separate line segments called:
   - `l1`: stretches from (0, 0) to (1, 1)
   - `l2`: stretches from (0, 1) to (1, 0)

     View them using base R `plot()`. > Each object will require a separate call to `plot()`.
     > You can add `add = TRUE` to `plot()` so that the contents of the second get added to the first.
     > Note that these plot additions don't carry over from code cell to code cell.

```r
l1 <- st_linestring(matrix(c(0,0,1,1), ncol = 2, byrow = TRUE))
l2 <- st_linestring(matrix(c(0,1,1,0), ncol = 2, byrow= TRUE))
#plot(c(l1,l2))
plot(l1)
plot(l2, add = TRUE)
```



2. Form a spatial data frame (an object with classes `sf` and `data.frame`) called **my_lines** with two columns:

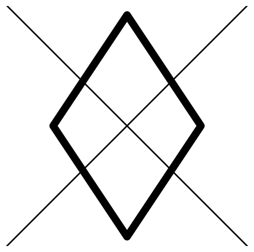
- `id`: an attribute with values 1 and 2.
- `geom`: a simple feature (geometry) column containing the two lines with the coordinate reference system `OGC:CRS84`.

```r
id <- c(1,2)
geom <- st_sfc(list(l1,l2), crs = 'OGC:CRS84')
my_lines <- st_sf(id= id, geom=geom)
my_lines
```

```
Simple feature collection with 2 features and 1 field
Geometry type: LINESTRING
Dimension:     XY
Bounding box:  xmin: 0 ymin: 0 xmax: 1 ymax: 1
Geodetic CRS:  WGS 84 (CRS84)
  id                  geom
1  1 LINESTRING (0 0, 1 1)
2  2 LINESTRING (0 1, 1 0)
```

3. Create a second spatial data frame with a diamond-shaped polygon.
   The diamond should have corners at (.5, .8), (.3, .5), (.5, .2), and (.7, .5) and use the same CRS.
   It should have an attribute called `price` with the value **100**.
   Call your spatial data frame **my_diamond** and plot it atop your lines.

```r
diamond <- matrix(c(.5,.8, .3,.5,.5,.2,.7,.5, .5,.8),ncol = 2, byrow = TRUE)
diamond <- st_polygon(list(diamond))
diamond_sfc <- st_sfc(diamond, crs = 'OGC:CRS84')
price <- 100
my_diamond <- st_sf(price = price, geom = diamond_sfc)
plot(st_geometry(my_diamond), lwd = 5)
plot(st_geometry(my_lines), add = TRUE)
```

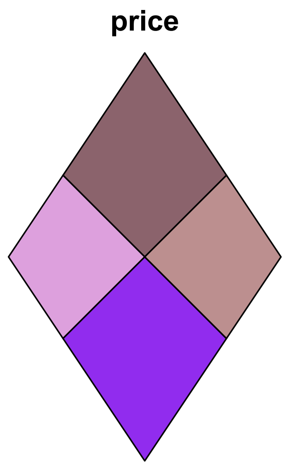


4. Calculate the **dimension**, **length**, and **area** of `my_lines()`. Dimension: 1 1 Length: 157249.6 157249.6 Area: 0 0
5. Calculate the **dimension**, **length**, and **area** of `my_diamond()`. Dimension: 2 Length: 0 Area: 1483670463
6. Use the `st_split()` function inside the **lwgeom** package to split the diamond into four diamonds using your lines.
   Name the resulting object **split_diamond**.

You can add colors to `plot()` using the `col` argument and call `colors()` at the console to see the color names available in base R.
Example: `col = c("skyblue", "tomato")` but you're welcome to pick your own colors.

After inspecting the result, run:

```
split_diamond <- st_split(my_diamond, my_lines)
split_diamond <- split_diamond |>
   st_collection_extract("POLYGON")
plot(split_diamond, col = c("pink4", "plum", "purple2", "rosybrown"))
```



Plot the resulting `sf` collection of 4 features, with each of the split diamonds filled with a color.

7. Add two attribute columns to `split_diamonds`:

- 1. `yield`: takes values of 10, 20, 30, and 40 for the top, right, bottom, and left split diamonds, respectively.
  (You can imagine this is the yield of four fields of corn.)
- 2. `split_price`: the original price of the full diamond scaled by the relative area of each of the four split diamonds.
  Round the split price to the nearest cent.

Be sure to save the new columns back into `split_diamonds`.

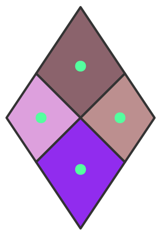
```
# SKIPPED
#st_split(my_diamond, st_union(my_lines)) |>

#split_diamonds <- st_sf(yield = c(10,20,30,40), split_price = price *
split_diamond)
```

8. Compute the **centroids** of the four split diamonds, call them `my_cents`, and add them to your plot. Set their `col` and `pch` as you like.

```
plot(
  st_geometry(split_diamond),
  col = c("pink4", "plum", "purple2", "rosybrown"),
  border = "gray20", lwd = 1.5,
  reset = FALSE
)
my_cents <- points(st_geometry(st_centroid(split_diamond)), pch = 16, col =
c("seagreen1","seagreen1","seagreen1","seagreen1"))
```

```
Warning: st_centroid assumes attributes are constant over geometries
```



---

9. Using what you know about data frame subsetting, remove the **bottom-most centroid**. Then take the **union** of the three remaining points, form a **convex hull** from them, and save the polygon to `my_tri_up`.
   > The functions for these set operations in `sf` are exactly as you'd expect:
   > the name of the operation prepended by `st_`.
   > Note also that the union operation will drop the `sf` class — you'll need to run your object through `st_sf()` to get it back. Plot `my_tri_up` on top of your split diamond.

10. Repeat the previous exercise, but form an `sf` object called `my_tri_down` that has the **top-most centroid** removed.

11. Under the definition of **"intersects,"** determine which of your split diamonds intersect with `my_tri_up`.
    Check by doing a **spatial join**.

12. Calculate the **area** of `my_tri_up` two ways:

13. By directly measuring the area of the triangle.

14. By interpolating from the `area` attribute of the split diamonds that it intersects with.

Do they agree? - SKIPPED —

13. Calculate the **yield** of both `my_tri_up` and `my_tri_down` by interpolating from the yield of the split diamonds that they intersect with.
    Why are they different from one another?

When interpolating, you'll need to decide whether your variable is **spatially extensive**.
An attribute that is spatially extensive has a magnitude proportional to the size of the feature

it’s measured on;
as the area of the feature increases, the magnitude of the attribute increases proportionally.