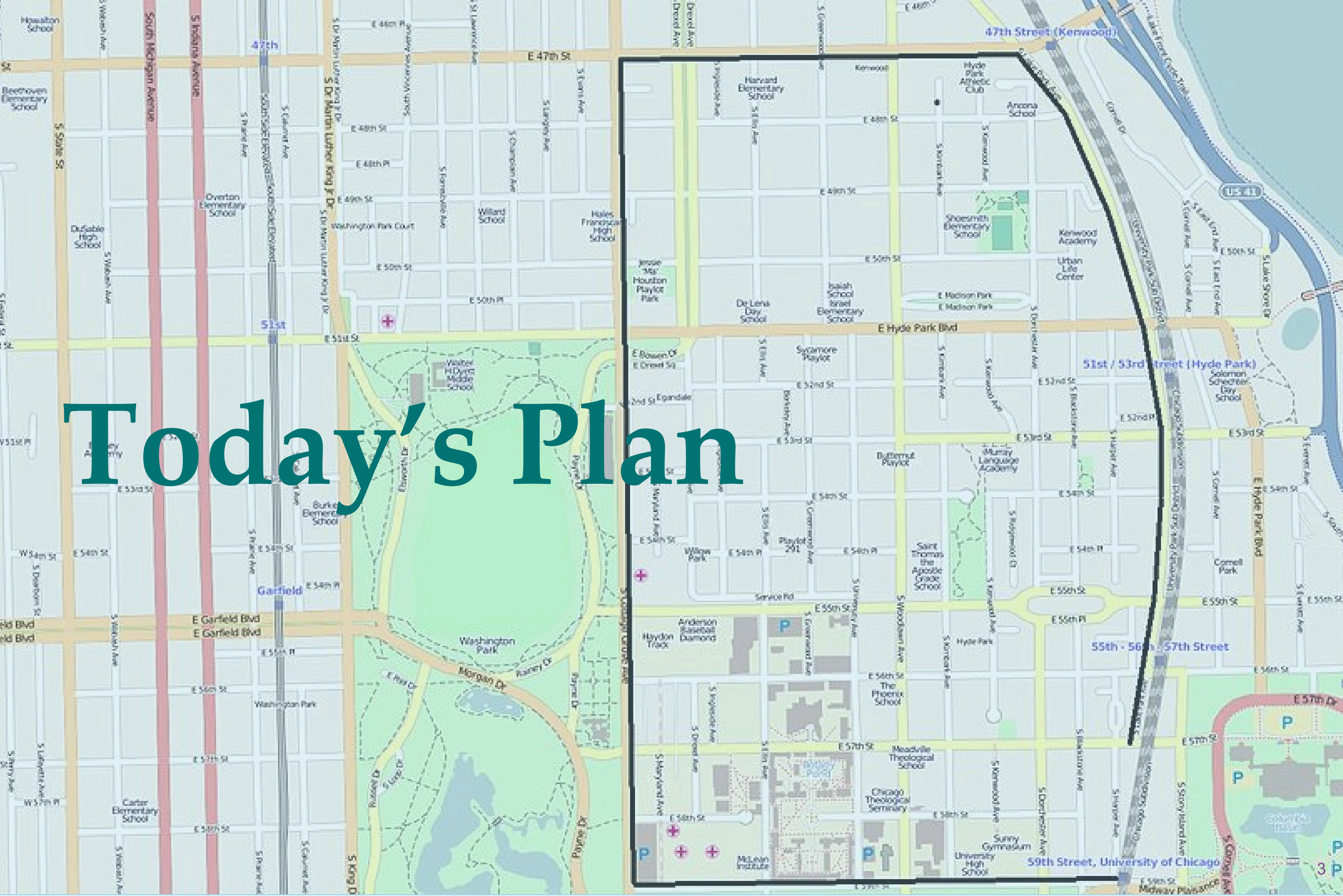
Areal Data: Vector Data

HES 505 Fall 2024: Session 7

Carolyn Koehn

Today’s Plan



E 47th St
47th
47th Street (Kenwood)
South Michigan Avenue
S Indiana Avenue
S Dr Martin Luther King Jr Dr
S State St
Howalton School
Beethoven Elementary School
Overton Elementary School
DuSable High School
Willard School
Hales Franciscan High School
Harvard Elementary School
Hyde Park Athletic Club
Ancona School
Shoesmith Elementary School
Kenwood Academy
Urban Life Center
Jessie "Ma" Houston Playlot Park
Dr Lena Day School
Isaiah School Israel Elementary School
E Hyde Park Blvd
E 51st St
51st
51st / 53rd Street (Hyde Park)
Solomon Schechter Day School
Walter H Dyett Middle School
Sycamore Playlot
E 53rd St
Butternut Playlot
Murray Language Academy
Burke Elementary School
Willow Park
Playlot 291
Saint Thomas the Apostle Grade School
Cornell Park
Garfield
E Garfield Blvd
E 55th St
Washington Park
Morgan Dr
Anderson Baseball Diamond
Haydon Track
55th - 56th - 57th Street
The Phoenix School
E 57th St
Meadville Theological School
Chicago Theological Seminary
Carter Elementary School
E 58th St
McLean Institute
Sunny Gymnasium
University High School
59th Street, University of Chicago
Midway Plaisance
S Cottage Grove Ave
S Woodlawn Ave
S Lake Shore Dr
E 56th St

Objectives

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

- Understand `predicates` and `measures` in the context of spatial operations in `sf`
- Define valid geometries and approaches for assessing geometries in `R`
- Use `st_*` and `sf_*` to evaluate attributes of geometries and calculate measurements

Understanding the language

Revisiting Simple Features

- The `sf` package relies on a simple feature data model to represent geometries
  - hierarchical
  - standardized methods
  - complementary binary and human-readable encoding

type	description
`POINT`	single point geometry
`MULTIPOINT`	set of points
`LINESTRING`	single linestring (two or more points connected by straight lines)
`MULTILINESTRING`	set of linestrings
`POLYGON`	exterior ring with zero or more inner rings, denoting holes
`MULTIPOLYGON`	set of polygons
`GEOMETRYCOLLECTION`	set of the geometries above

Revisiting Simple Features

- You already know how to access some elements of a simple feature
- `st_crs` - returns the coordinate reference system
- `st_bbox` - returns the bounding box for the simple feature

Standaridized Methods

We can categorize `sf` operations based on what they return and/or how many geometries they accept as input.

- *Output Categories*
  - **Predicates**: evaluate a logical statement asserting that a property is `TRUE`
  - **Measures**: return a numeric value with units based on the units of the CRS
  - **Transformations**: create new geometries based on input geometries.
- *Input Geometries*
  - **Unary**: operate on a single geometry at a time (meaning that if you have a `MULTI*` object the function works on each geometry individually)
  - **Binary**: operate on pairs of geometries
  - **n-ary**: operate on sets of geometries

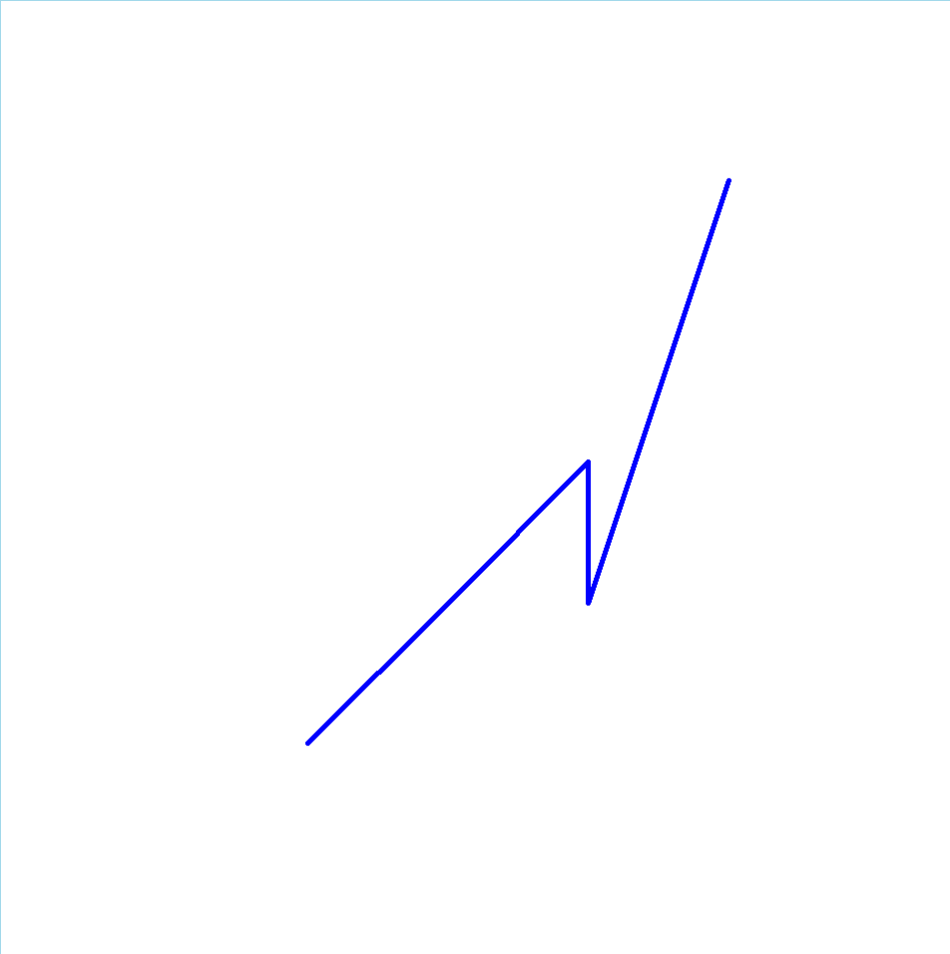
Valid Geometries

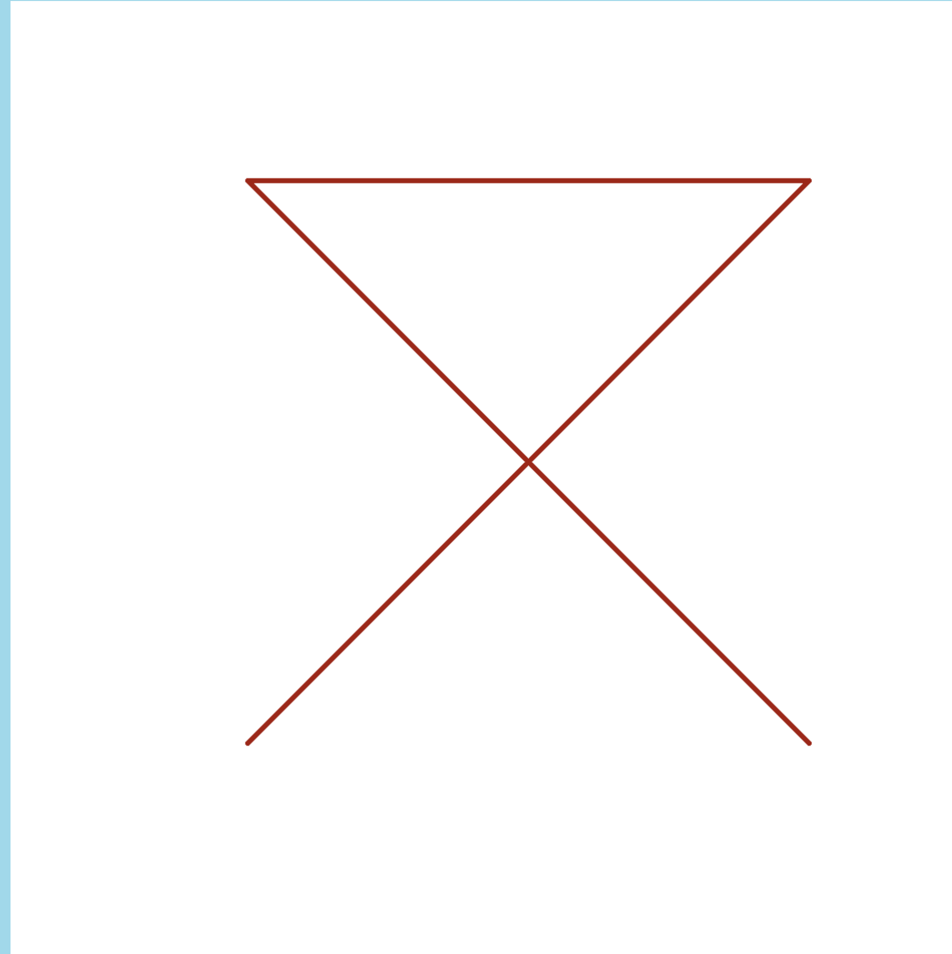
Remembering Valid Geometries

- A `linestring` is *simple* if it does not intersect

```
library(sf)
library(tidyverse)
ls = st_linestring(rbind(c(0,0), c(1,1),  c(2,2), c(2,1), c(3,4)))

ls2 = st_linestring(rbind(c(0,0), c(1,1),  c(2,2), c(0,2), c(1,1),
```



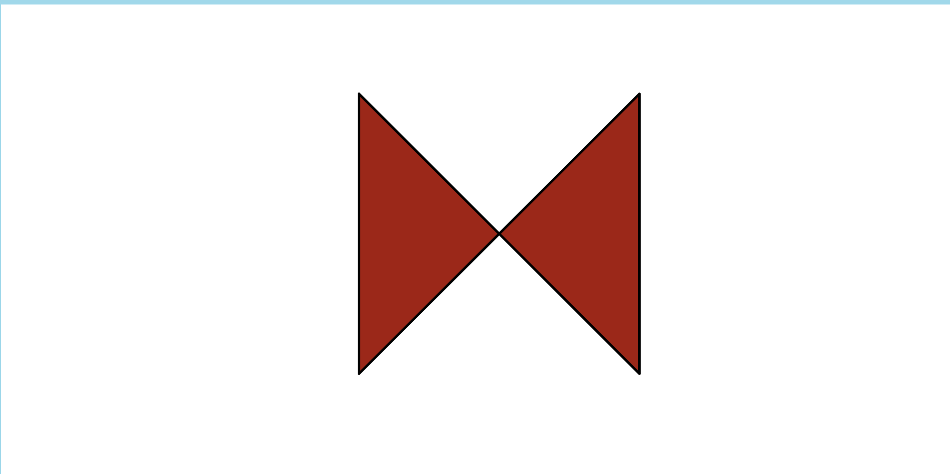


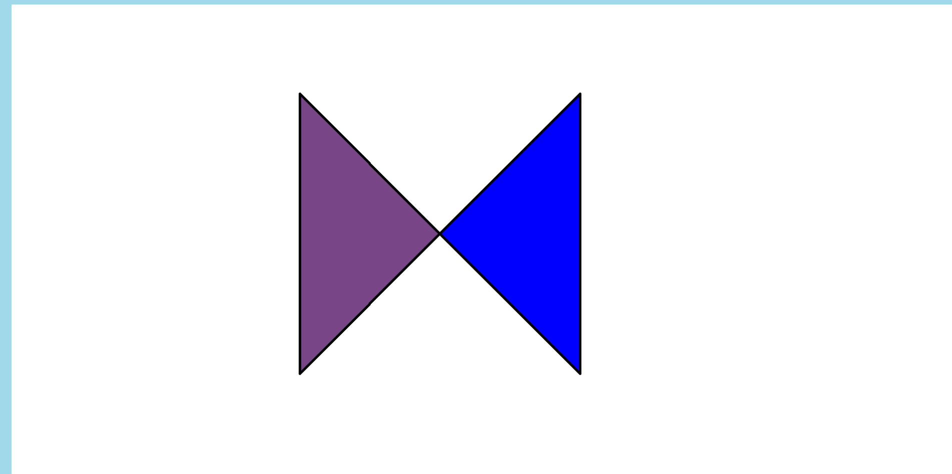
Remembering Valid Geometries

- Valid polygons
  - Are closed (i.e., the last vertex equals the first)
  - Have holes (inner rings) that inside the the exterior boundary
  - Have holes that touch the exterior at no more than one vertex (they don’t extend across a line)
    - For multipolygons, adjacent polygons touch only at points
  - Do not repeat their own path

Remembering Valid Geometries

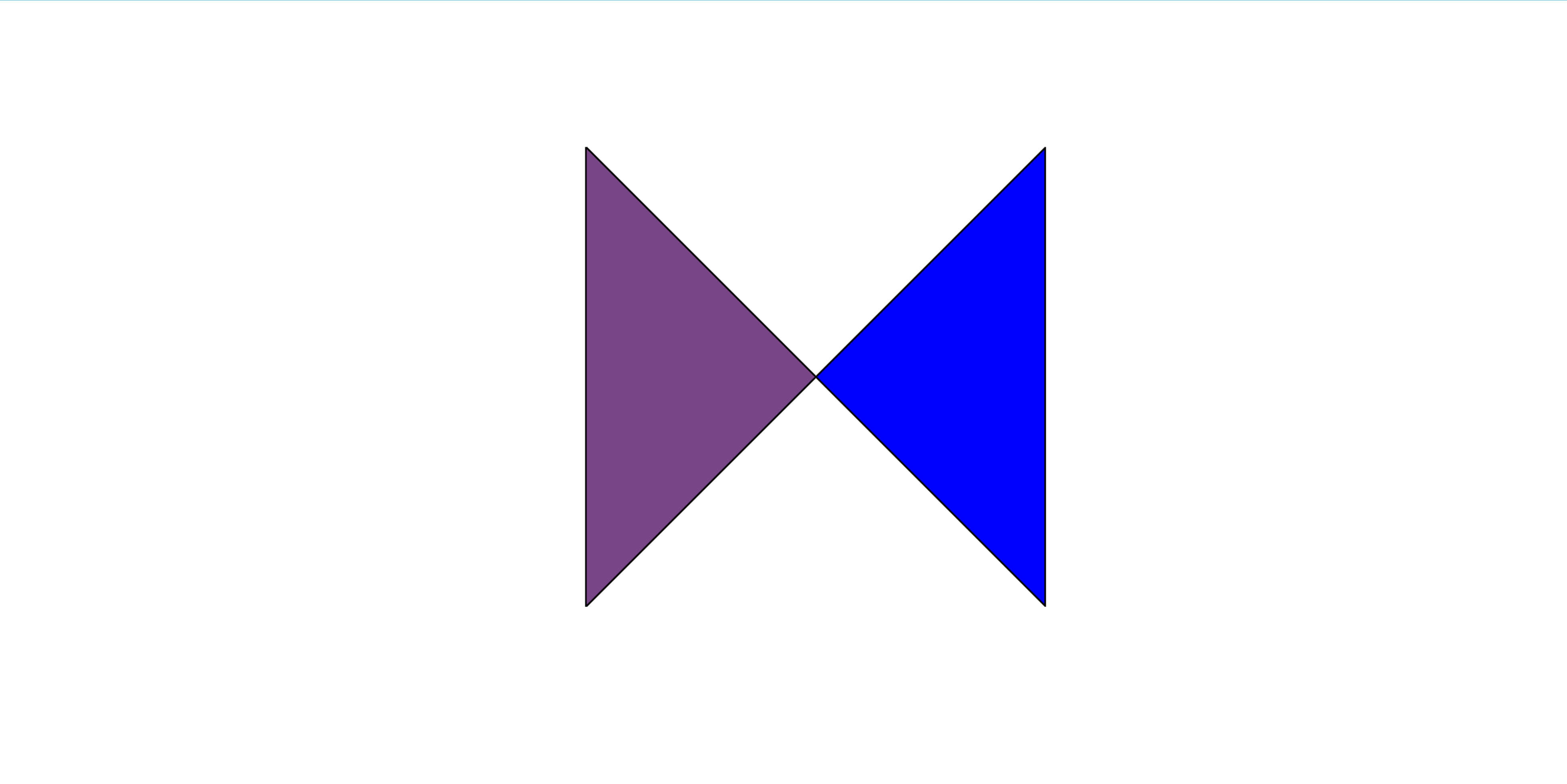
```
p1 = st_as_sfc("POLYGON((0 0, 0 10, 10 0, 10 10, 0 0))")
p2 = st_as_sfc("POLYGON((0 0, 0 10, 5 5,  0 0))")
p3 = st_as_sfc("POLYGON((5 5, 10 10, 10 0, 5 5))")
```





Remembering Valid Geometries

```
p4 = st_as_sfc(c("POLYGON((0 0, 0 10, 5 5,  0 0))", "POLYGON((5 5,
plot(p4, col=c( "#7C4A89", "blue"))
```



Empty Geometries

- Empty geometries arise when an operation produces `NULL` outcomes (like looking for the intersection between two non-intersecting polygons)
- `sf` allows empty geometries to make sure that information about the data type is retained
- Similar to a `data.frame` with no rows or a `list` with `NULL` values
- Most vector operations require simple, valid geometries

Predicates

Using Unary Predicates

- Unary predicates accept single geometries (or geometry collections)
- Provide helpful ways to check whether your data is ready to analyze
- Use the `st_` prefix and return `TRUE/FALSE`

predicate	asks…
`is_simple`	is the geometry self-intersecting (i.e., simple)?
`is_valid`	is the geometry valid?
`is_empty`	is the geometry column of an object empty?
`is_longlat`	does the object have geographic coordinates? (`FALSE` if coords are projected, `NA` if no `crs`)
`is(geometry, class)`	is the geometry of a particular class?

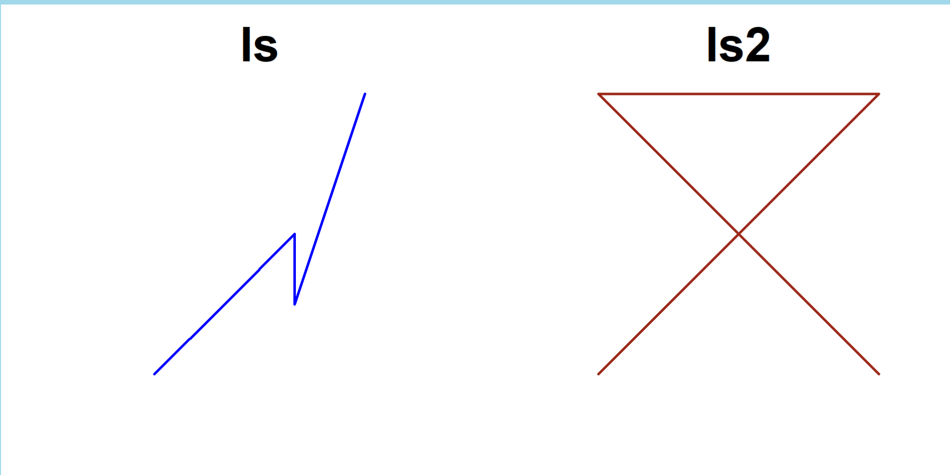
Checking Geometries With Unary Predicates

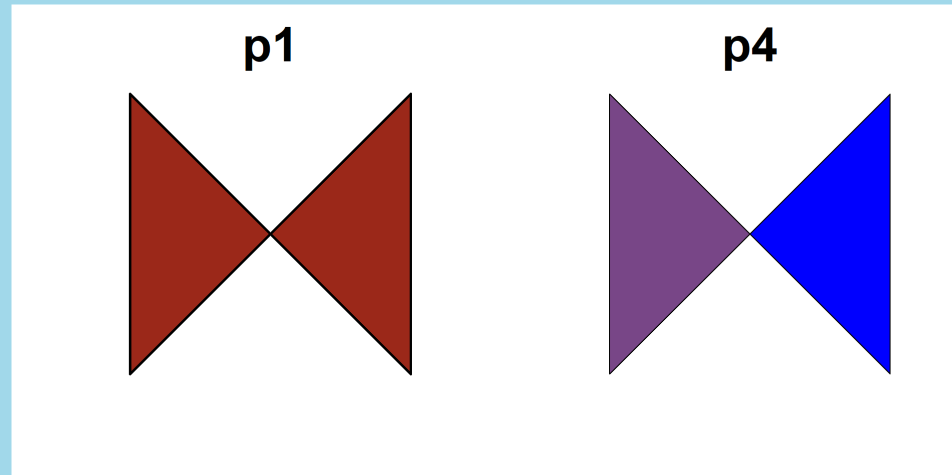
- Before conducting costly analyses, it's worth checking for:

1. empty geometries, using `any(st_is_empty(x))`
2. corrupt geometries, using `any(is.na(st_is_valid(x)))`
3. invalid geometries, using `any(na.omit(st_is_valid(x)) == FALSE)`; in case of corrupt and/or invalid geometries,
4. in case of invalid geometries, query the reason for invalidity by `st_is_valid(x, reason = TRUE)`

Invalid geometries will require **transformation** (next week!)

Checking Geometries With Unary Predicates






```
st_is_simple(ls)
```

[1] TRUE

```
st_is_simple(ls2)
```

[1] FALSE

```
st_is_valid(p1)
```

[1] FALSE

```
st_is_valid(p4)
```

[1] TRUE TRUE

Unary Predicates and Real Data

```
library(tigris)
id.cty <- countie

st_crs(id.cty)$inp
```

```
[1] "NAD83"
```

```
st_is_longlat(id.c
```

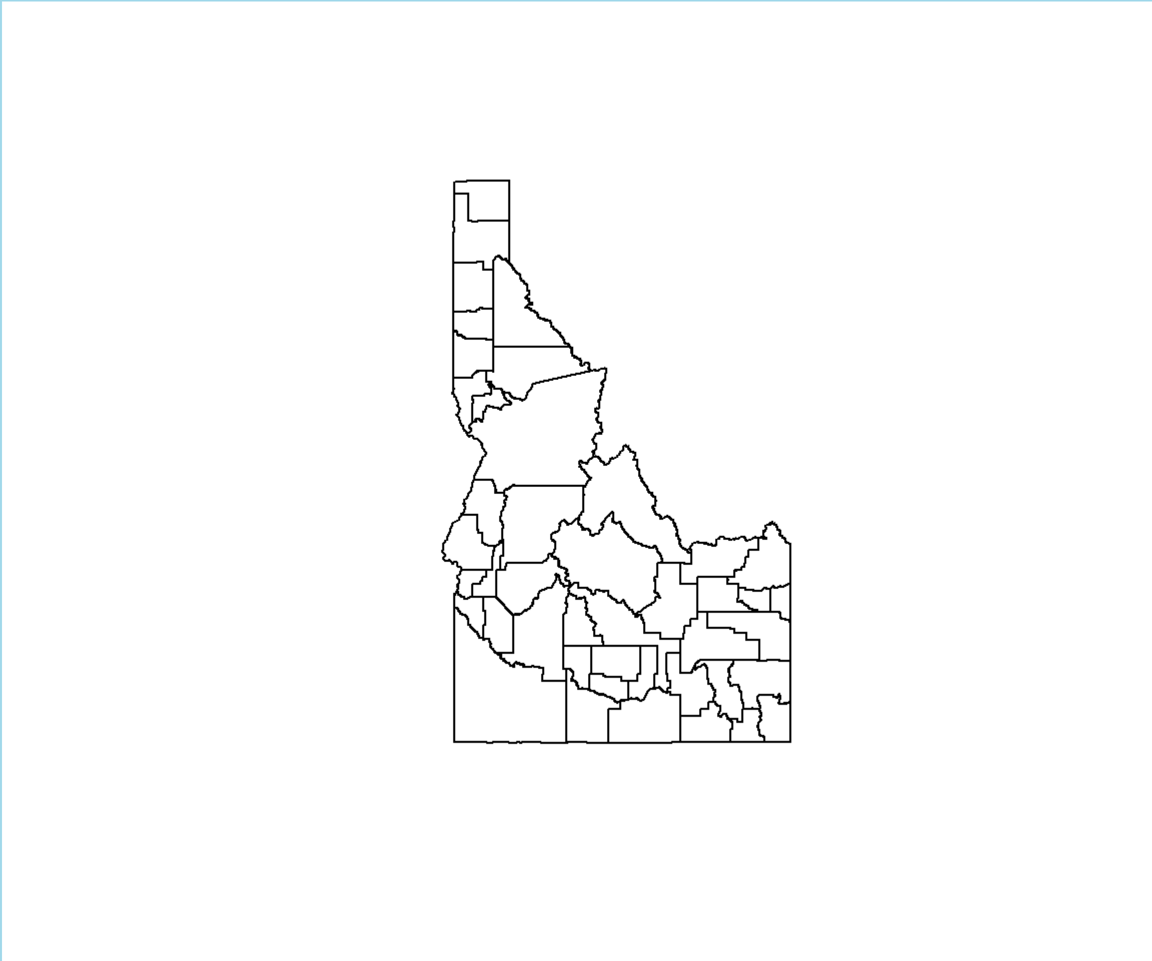
```
[1] TRUE
```

```
st_is_valid(id.cty
```

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

```
all(st_is_valid(id
```

```
[1] TRUE
```



Binary Predicates

Binary Predicates

- Accept exactly two geometries (or collections)
- Also return `logical` outcomes
- Based on the Dimensionally Extended 9-Intersection Model (DE-9IM)

predicate	meaning	inverse of
`contains`	None of the points of A are outside B	`within`
`contains_properly`	A contains B and B has no points in common with the boundary of A	
`covers`	No points of B lie in the exterior of A	`covered_by`
`covered_by`	Inverse of `covers`	
`crosses`	A and B have some but not all interior points in common	
`disjoint`	A and B have no points in common	`intersects`
`equals`	A and B are topologically equal: node order or number of nodes may differ; identical to A contains B AND A within B	
`equals_exact`	A and B are geometrically equal, and have identical node order	
`intersects`	A and B are not disjoint	`disjoint`
`is_within_distance`	A is closer to B than a given distance	
`within`	None of the points of B are outside A	`contains`
`touches`	A and B have at least one boundary point in common, but no interior points	
`overlaps`	A and B have some points in common; the dimension of these is identical to that of A and B	
`relate`	given a mask pattern, return whether A and B adhere to this pattern	

Binary Predicates

```
id <- states(progr
  filter(STUSPS ==
or <- states(progr
  filter(STUSPS ==
ada.cty <- id.cty
  filter(NAME == '
```

```
st_covers(id, ada.cty)
```

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

```
st_covers(id, ada.cty, sparse=FALS
```

```
     [,1]
[1,] TRUE
```

```
st_within(ada.cty, or)
```

```
Sparse geometry binary predicate list of
length 1, where the predicate
was `within'
 1: (empty)
```

```
st_within(ada.cty, or, sparse=FALS
```

```
      [,1]
[1,] FALSE
```

Measures

Measures

Unary Measures

- Return quantities of individual geometries

measure	returns
`dimension`	0 for points, 1 for linear, 2 for polygons, possibly `NA` for empty geometries
`area`	the area of a geometry
`length`	the length of a linear geometry

Unary Measures

```
st_area(id)
```

```
2.15994e+11 [m^2]
```

```
st_area(id.cty[1:5,])
```

```
Units: [m^2]
[1] 2858212132 3380630278 1459359818 1726660462 1223521586
```

```
st_dimension(id.cty[1:5,])
```

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

Binary Measures

- `st_distance` returns the distance between pairs of geometries

```r
kootenai.cty <- id.cty %>%
  filter(NAME == "Kootenai")
st_distance(kootenai.cty, ada.cty)
```

```
Units: [m]
         [,1]
[1,] 396433.8
```

```r
st_distance(id.cty)[1:5, 1:5]
```

```
Units: [m]
         [,1]     [,2]     [,3]     [,4]     [,5]
[1,]      0.0 467635.7 277227.0 132998.0      0.0
[2,] 467635.7      0.0 319706.4 656056.0 514306.9
[3,] 277227.0 319706.4      0.0 377105.4 336146.8
[4,] 132998.0 656056.0 377105.4      0.0 133045.5
[5,]      0.0 514306.9 336146.8 133045.5      0.0
```