Coordinates and Geometries

HES 505 Fall 2023: Session 7

Matt Williamson

Today's Plan

Objectives

- Understand the linkage between location, coordinates, coordinate reference systems, and geometry

- Access and manipulate geometries in R with `sf` (and `terra`)

- Define `geometry` in the context of vector objects and troubleshoot common problems

- Change the CRS for vectors and rasters (and understand the implications)

But first…

Getting more acquainted with R

- Objects, classes, functions, oh my…

- Intuition for the `tidyverse`

- Getting used to pipes (`%>%` or `|>`)

- Learning to prototype

Getting help

2 Kinds of Errors

- **Syntax Errors**: Your code won't actually run

- **Semantic Errors**: Your code runs without error, but the result is unexpected

Asking better questions, getting better answers

- Places to get help (Google, Slack, Stack Overflow, Github Issue pages)

- What are you trying to do? (the outcome you want/expect)

- What isn't working? (the code and steps you've tried so far)

- Why aren't common solutions working? (proof that you've done your due diligence)

Reproducible examples

- Don't require someone to have your data or your computer

- Minimal amount of information and code to reproduce your error

- Includes both code and your operating environment info

- More info

- An example with spatial data

Coordinates and Geometries

Reference Systems

- To locate an object or quantity, we need:

  - A fixed *origin* (or **datum**) to measure distances to/from

  - A *measurement unit* (or **scale**) that defines the units of distance

  - **Datum** + **scale** = **reference system**

Coordinate Reference Systems

- Map the location on an object to earth (geodetic) or flat (projected) surfaces

- **Coordinate System** - the mathematical rules that specify how coordinates are assigned to points

- **Datum** - the parameter or set of parameters that define the position of the origin, the scale, and the orientation of a coordinate system

- **Coordinate Reference Systems** - a coordinate system that is related to an object by a datum

Accessing CRS with R

- `sf::st_crs()` for vector data

- `terra::crs()` for raster data

- stored in WKT, epsg, or proj4string (deprecated)

- The EPSG website is a great reference for getting projection info

Accessing CRS with R

```r
1  dir.for.files <- "/Users/mattwilliamson/Library/CloudStorage/GoogleDrive-ma
2  vector.data <- sf::st_read(dsn = paste0(dir.for.files, "cejst_nw.shp"), qui
3  sf::st_crs(x = vector.data)$input
```

```
[1] "WGS 84"
```

```r
1  sf::st_crs(x = vector.data)$proj4string
```

```
[1] "+proj=longlat +datum=WGS84 +no_defs"
```

```r
1  sf::st_crs(x = vector.data)$wkt
```

```
[1] "GEOGCRS[\"WGS 84\",\n    DATUM[\"World Geodetic System 1984\",\n
ELLIPSOID[\"WGS 84\",6378137,298.257223563,\n
LENGTHUNIT[\"metre\",1]]],\n    PRIMEM[\"Greenwich\",0,\n
ANGLEUNIT[\"degree\",0.0174532925199433]],\n    CS[ellipsoidal,2],\n
AXIS[\"latitude\",north,\n            ORDER[1],\n
ANGLEUNIT[\"degree\",0.0174532925199433]],\n        AXIS[\"longitude\",east,\n
ORDER[2],\n            ANGLEUNIT[\"degree\",0.0174532925199433]],\n
ID[\"EPSG\",4326]]"
```

Accessing CRS with R

```
1  raster.data <- terra::rast(x = paste0(dir.for.files, "wildfire_hazard_agg.t
2  terra::crs(raster.data, describe=TRUE, proj=TRUE)
```

```
        name authority code area          extent
1 unnamed        <NA> <NA> <NA> NA, NA, NA, NA

proj
1 +proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0
+datum=NAD83 +units=m +no_defs
```

What if you don't know the CRS?

- Sometimes you receive data that is missing the projection

- You can assign it (**with caution**)

- You can guess it using `crsuggest::guess_crs()`

```r
1   library(sf)
2   library(mapview)
3   locations <- data.frame(
4     X = c(1200822.97857801, 1205015.51644983, 1202297.44383987, 1205877.68696
5           1194763.21511923, 1195463.42403192, 1199836.01037452, 1207081.96500
6           1201924.15986897),
7     Y = c(1246476.31475063, 1248612.72571423, 1241479.45996392, 1243898.58428
8           1246033.7550009, 1241827.7730307, 1234691.50899912, 1251125.6780848
9           1252188.4333016),
10    id = 1:9
11  )
12
13  locations_sf <- st_as_sf(locations, coords = c("X", "Y"))
```
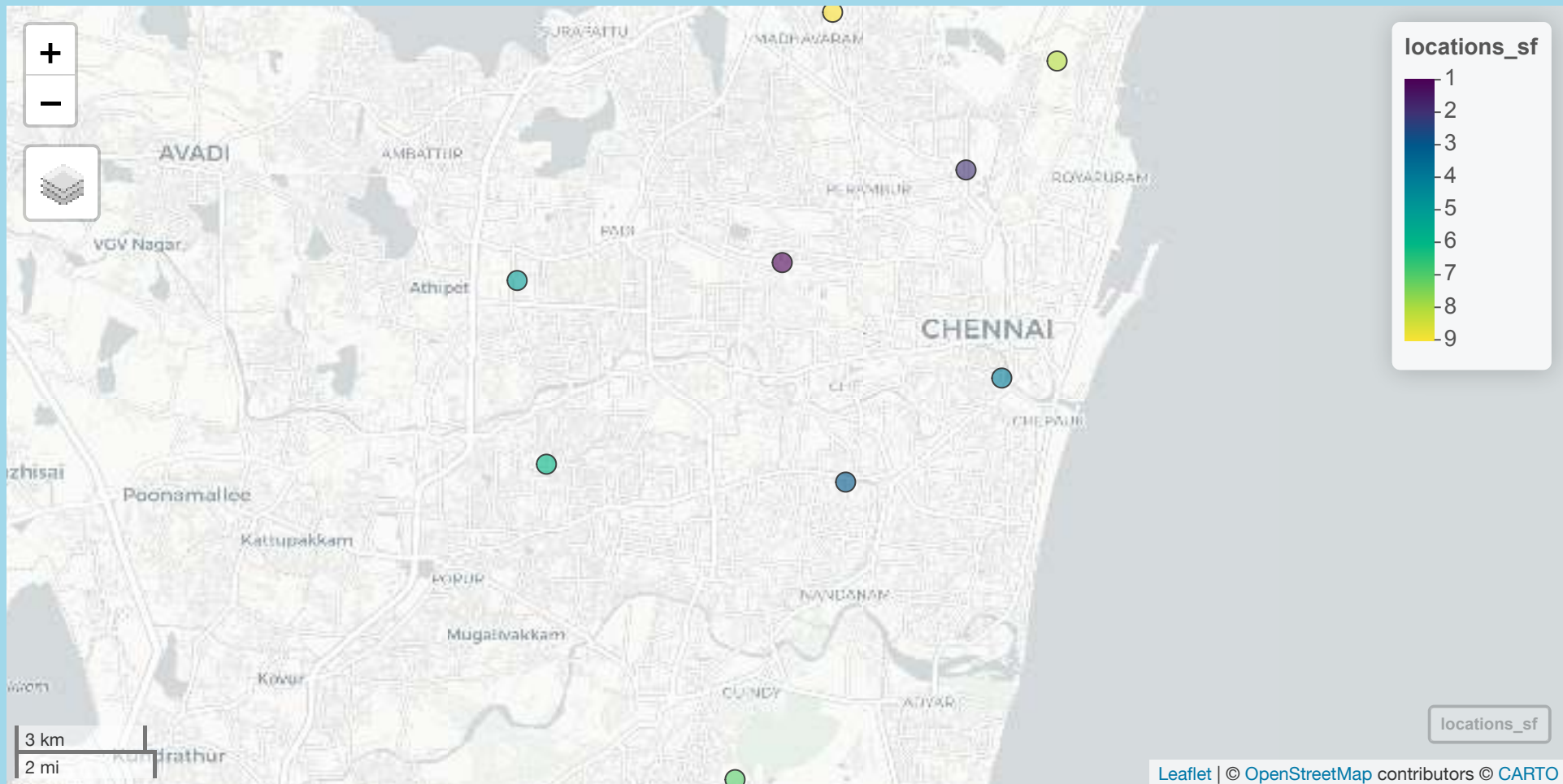
locations_sf

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9

locations_sf

Leaflet

21
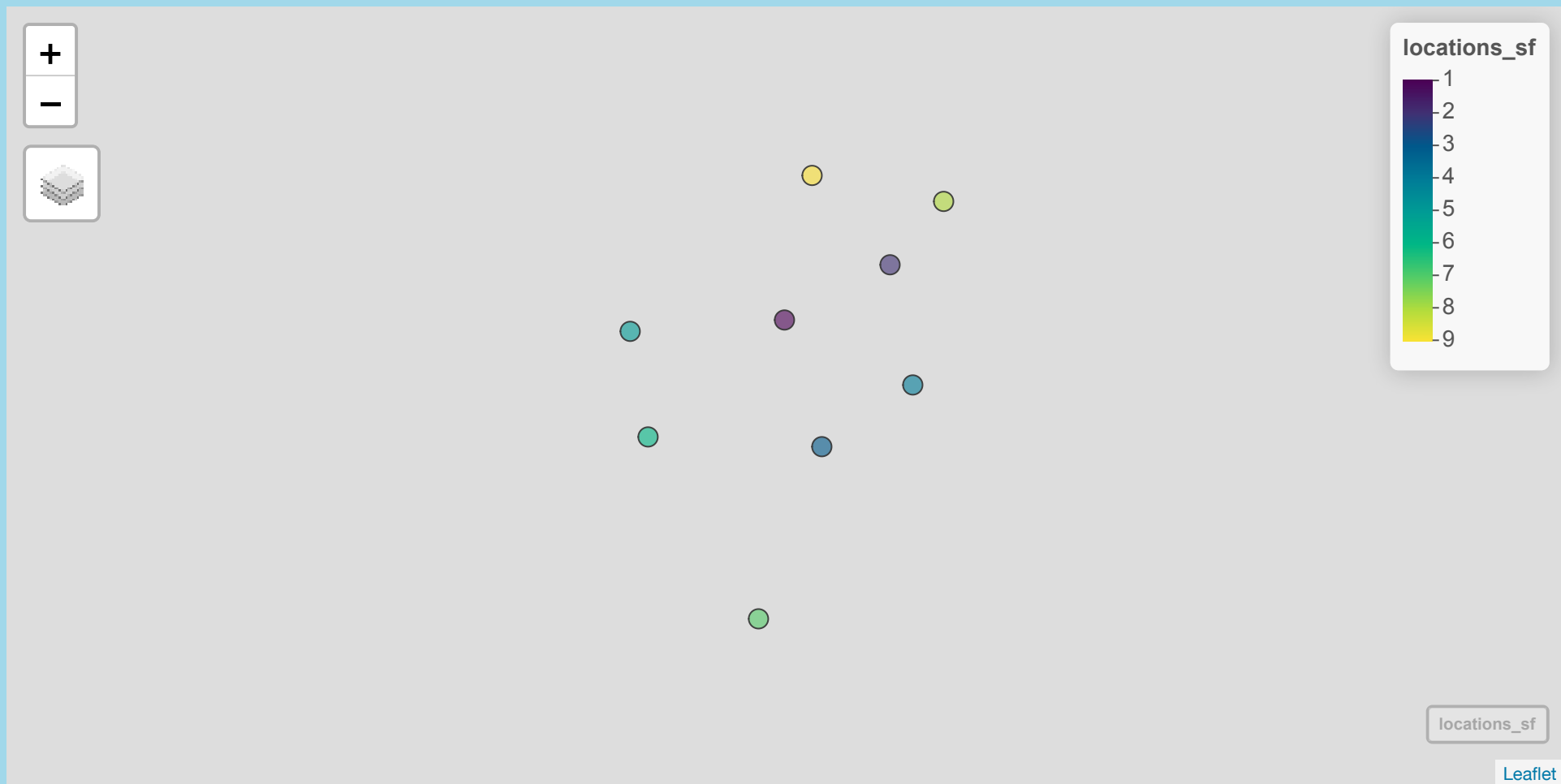
Guessing CRS

```r
1  library(crsuggest)
2  guess_crs(locations_sf, "Chennai, India", n_return = 5)
```

```
# A tibble: 5 × 2
  crs_code dist_km
  <chr>      <dbl>
1 7785        4.07
2 24344      806.
3 32644      806.
4 32244      806.
5 32444      806.
```

```r
1  st_crs(locations_sf) <- 7785
```

locations_sf

1
2
3
4
5
6
7
8
9

locations_sf

Leaflet | © OpenStreetMap contributors © CARTO

23

All spatial analysis and mapping requires that your data is aligned correctly.

Changing the CRS

- Requires recomputing coordinates

- Coordinate Conversion - No change to the datum; lossless

- Coordinate Transformation - New datum; relies on models; some error involved

Changing the CRS in R

- `sf::st_transform` for vectors

- `terra::project` for rasters

- **Projecting Rasters Causes Distortion**

Changing the CRS in R

```r
1  vector.data.proj <- vector.data %>%
2    sf::st_transform(., crs = 3083)
3  st_crs(vector.data.proj)$input
```

```
[1] "EPSG:3083"
```

```r
1  vector.data.proj.rast <- vector.data %>%
2    sf::st_transform(., crs = crs(raster.data))
3  st_crs(vector.data.proj.rast)$proj4string
```

```
[1] "+proj=aea +lat_0=23 +lon_0=-96 +lat_1=29.5 +lat_2=45.5 +x_0=0 +y_0=0
+datum=NAD83 +units=m +no_defs"
```

Changing the CRS in **R**

```
1  raster.data.proj <- project(x = raster.data, y = "EPSG:3083")
2  crs(raster.data.proj, describe=TRUE, proj=TRUE)
```

```
                              name authority code
1 NAD83 / Texas Centric Albers Equal Area      EPSG 3083
                          area                            extent
1 United States (USA) - Texas -106.66, -93.50, 36.50, 25.83

proj
1 +proj=aea +lat_0=18 +lon_0=-100 +lat_1=27.5 +lat_2=35 +x_0=1500000
+y_0=6000000 +datum=NAD83 +units=m +no_defs
```

```
1  raster.data.proj.vect <- project(x = raster.data, y = vect(vector.data))
2  crs(raster.data.proj.vect, describe=TRUE, proj=TRUE)
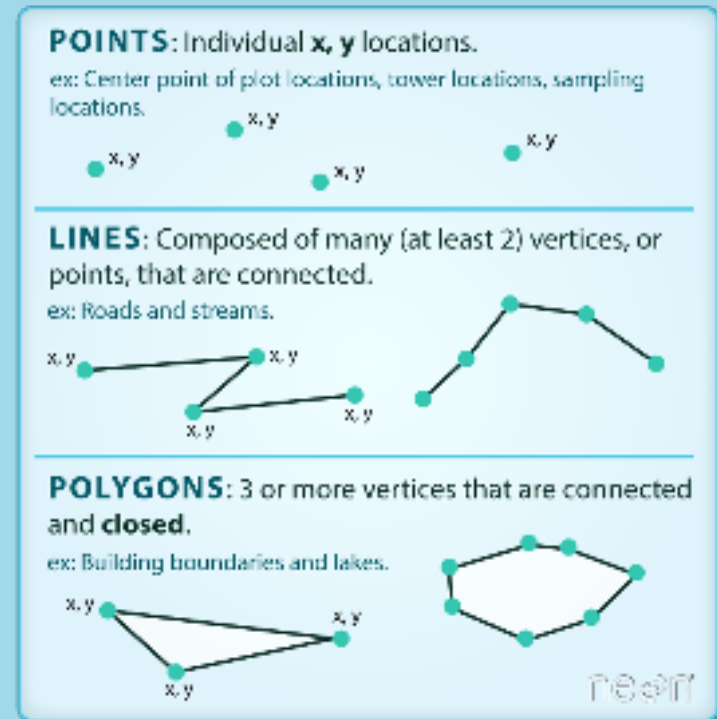```

```
    name authority code area   extent
proj
1 WGS 84   EPSG 4326 <NA> NA, NA, NA, NA +proj=longlat +datum=WGS84
+no_defs
```
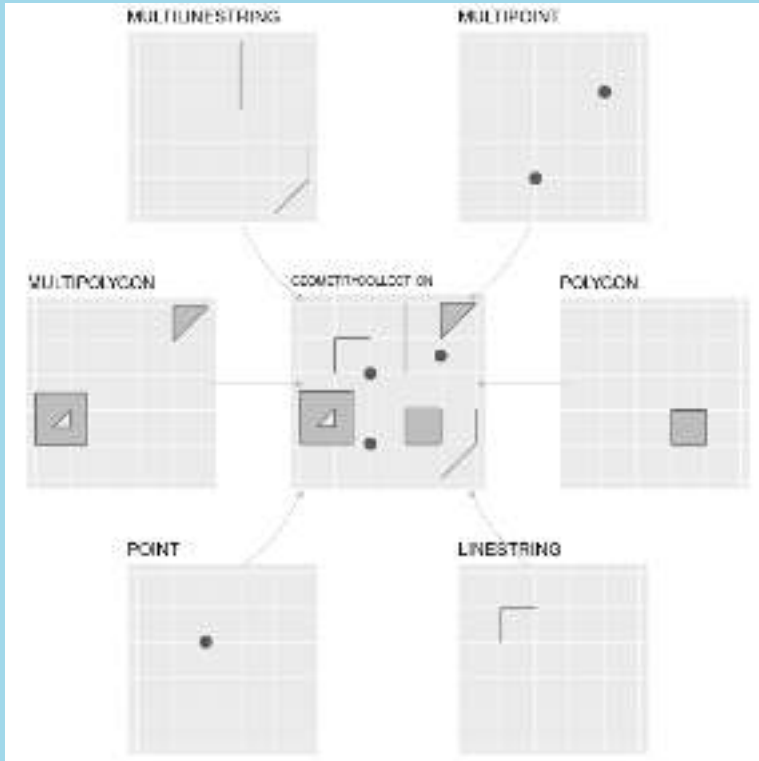
# Geometries

# The Vector Data Model

- Coordinates define the **Vertices** (i.e., discrete x-y locations) that comprise the geometry

- The organization of those vertices define the *shape* of the vector

- General types: points, lines, polygons



**POINTS**: Individual **x, y** locations.
ex: Center point of plot locations, tower locations, sampling locations.

**LINES**: Composed of many (at least 2) vertices, or points, that are connected.
ex: Roads and streams.

**POLYGONS**: 3 or more vertices that are connected and **closed**.
ex: Building boundaries and lakes.

# Representing vector data in R



From Lovelace et al.

- sf hierarchy reflects increasing complexity of geometry
  - st_point, st_linestring, st_polygon for single features
  - st_multi∗ for multiple features of the same type
  - st_geometrycollection for multiple feature types
  - st_as_sfc creates the geometry list column for many sf operations

33

# Points

```
1  library(sf)
2  proj <- st_crs('+proj=longlat +datum=WGS84')
3  long <- c(-116.7, -120.4, -116.7, -113.5, -115.5, -120.8, -119.5, -113.7, -
4  lat <- c(45.3, 42.6, 38.9, 42.1, 35.7, 38.9, 36.2, 39, 41.6, 36.9)
5  st_multipoint(cbind(long, lat)) %>% st_sfc(., crs = proj)
```

```
Geometry set for 1 feature
Geometry type: MULTIPOINT
Dimension:     XY
Bounding box:  xmin: -120.8 ymin: 35.7 xmax: -110.7 ymax: 45.3
Geodetic CRS:  +proj=longlat +datum=WGS84
```
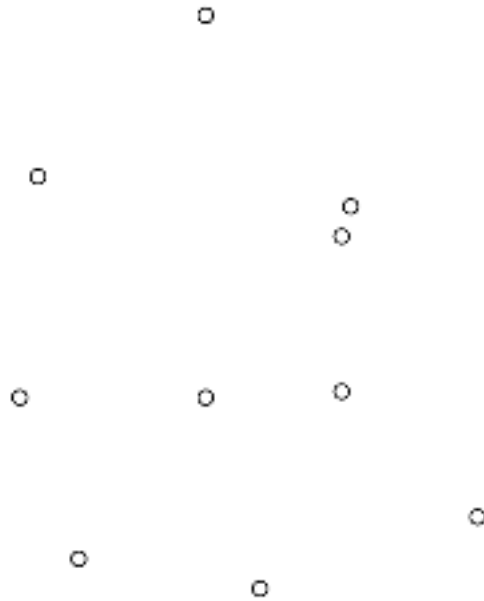
Points

```
1  plot(st_multipoint(cbind(long, lat)) %>%
2                     st_sfc(., crs = proj))
```

Lines

```
1  lon <- c(-116.8, -114.2, -112.9, -111.9, -114.2, -115.4, -117.7)
2  lat <- c(41.3, 42.9, 42.4, 39.8, 37.6, 38.3, 37.6)
3  lonlat <- cbind(lon, lat)
4  pts <- st_multipoint(lonlat)
5
6  sfline <- st_multilinestring(list(pts[1:3,], pts[4:7,]))
7  str(sfline)
```

```
List of 2
 $ : num [1:3, 1:2] -116.8 -114.2 -112.9 41.3 42.9 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:2] "lon" "lat"
 $ : num [1:4, 1:2] -111.9 -114.2 -115.4 -117.7 39.8 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:2] "lon" "lat"
 - attr(*, "class")= chr [1:3] "XY" "MULTILINESTRING" "sfg"
```

Lines

```
1  plot(st_multilinestring(list(pts[1:3,], pts[4:7,])))
```
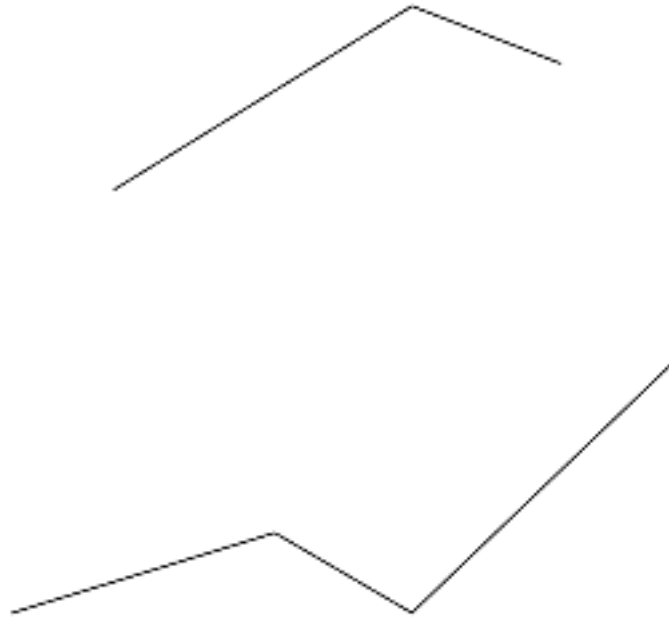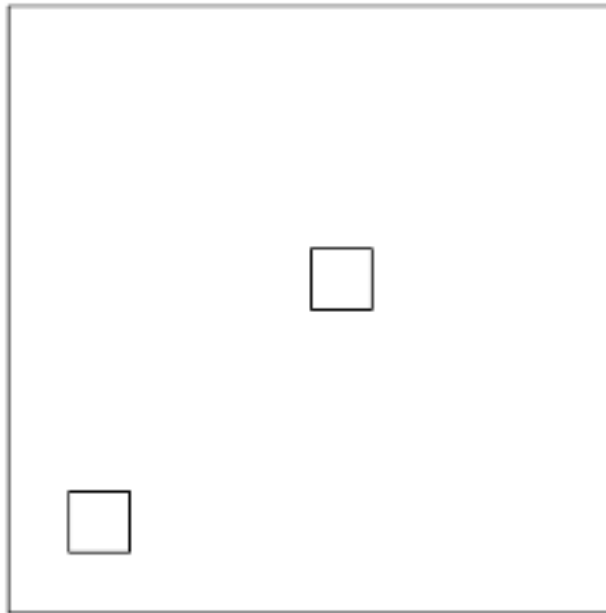
Polygons

```r
1  outer = matrix(c(0,0,10,0,10,10,0,10,0,0),ncol=2, byrow=TRUE)
2  hole1 = matrix(c(1,1,1,2,2,2,2,1,1,1),ncol=2, byrow=TRUE)
3  hole2 = matrix(c(5,5,5,6,6,6,6,5,5,5),ncol=2, byrow=TRUE)
4  coords = list(outer, hole1, hole2)
5  pl1 = st_polygon(coords)
```
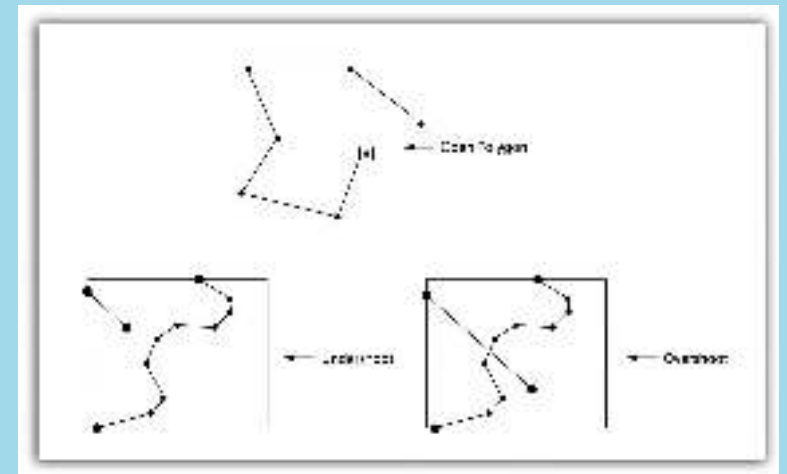
Polygons

```
1  plot(pl1)
```

Common Problems with Vector Data

- Vectors and scale

- Slivers and overlaps

- Undershoots and overshoots

- Self-intersections and rings

Topology Errors - Saylor Acad.

We'll use `st_is_valid()` to check this, but fixing can be tricky

Fixing Problematic Topology

- `st_make_valid()` for simple cases

- `st_buffer` with `dist=0`

- More complex errors need more complex approaches

A Note on Vectors

Moving forward we will rely primarily on the `sf` package for vector manipulation. Some packages require objects to be a different class. `terra`, for example, relies on `SpatVectors`. You can use `as()` to coerce objects from one type to another (assuming a method exists). You can also explore other packages. Many packages provide access to the 'spatial' backbones of `R` (like `geos` and `gdal`), they just differ in how the "verbs" are specified. For `sf` operations the `st_` prefix is typical. For `rgeos` operations, the `g` prefix is common.