Welcome to

instats

The Session Will Begin Shortly

START

Spatial Data Analysis and Visualization in R

Session 7: Working with Vector Data in R Using the sf Package

instats

Introduction

What is **sf**?

- R package for vector spatial data
- Based on the Simple Features standard (ISO 19125)
- Successor to the older **sp** package

Why use sf?

- Simple, consistent syntax
- Stores geometry and attributes in one object
- Compatible with tidyverse tools

Reading spatial data into **sf**

```r
library(sf)
library(tidyverse)
nc <- st_read(system.file("shape/nc.shp", package = "sf"))
```

```
Reading layer `nc' from data source
  `/Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/library/sf/shape/nc.shp'
  using driver `ESRI Shapefile'
Simple feature collection with 100 features and 14 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -84.32385 ymin: 33.88199 xmax: -75.45698 ymax: 36.58965
Geodetic CRS:  NAD27
```

- Supports GeoPackage, GeoJSON, shapefiles, and more

`sf` objects

- `sf` objects are data.frames

```
class(nc)
```

```
[1] "sf"         "data.frame"
```

- it has an additional column that stores the geometries, usually called `"geometry"`.
- this column is also accesible via `st_geometry(nc)`

Getting `sf` objects

- Options:
  - → Read with `st_read`
  - → Create with `st_sf`
  - → Convert with `st_as_sf`
- Make sure:
  - → Geometry column is recognized and set correctly (`st_sf` has argument `sf_column_name`)
  - → CRS is correctly specified (via argumnet `crs`)

Geometry types

- Common geometry types:
  - → POINT, MULTIPOINT
  - → LINESTRING, MULTILINESTRING
  - → POLYGON, MULTIPOLYGON
- Set of geometries:
  - → GEOMETRYCOLLECTION
- Rare geometry types:
  - → CIRCULARSTRING, COMPOUNDCURVE, CURVEPOLYGON, MULTICURVE, and many more
- More on simple features

Alternatives

- **terra** can also handle vector data, but is best known for handling raster data
- **sfheaders** lightweight alternative. Processes `sf` objects, but without requirment of external C++ libraries.

C/C++ Libraries behind sf

Key libraries

- **GEOS**: Geometry Engine
- **GDAL**: Geospatial Data Abstraction Library
- **PROJ**: Coordinate transformation library
- **S2**: Spherical geometry library from Google

GEOS

- Geometry engine written in **C++**
- Provides planar operations:
  - → `st_union()`, `st_intersection()`, `st_buffer()`, etc.
- Assumes flat (projected) coordinates

GDAL

- Stands for **Geospatial Data Abstraction Library**
- Handles reading and writing of raster and vector formats
- Interfaces with hundreds of file types
- Low-level access via dedicated R packages **vapour** and **gdalraster**

PROJ

- Handles **coordinate reference system (CRS)** transformations
- Used to:
  - → Convert between geographic and projected CRS
  - → Perform datum transformations

S2 geometry

- Developed by Google for spherical geometry
- Written in **C++**
- Models geometry on the surface of the Earth

Why S2?

- Planar geometry (GEOS) fails at:
  - → Antimeridian crossings
  - → Polar regions
- S2 works on a **spherical model** of Earth
- Better suited for **global-scale** vector data
- R package interface **s2**

Using S2 in sf

```r
library(sf)
sf_use_s2(TRUE)   # Enables spherical operations
sf_use_s2(FALSE)  # Reverts to planar operations
```

GEOS vs. S2

Feature	GEOS (planar)	S2 (spherical)
Geometry type	Projected coordinates	Geographic coordinates
Language	C++	C++ (Google)
Best for	Local geometry	Global geometry
Handles poles?	❌	✅
Antimeridian?	❌	✅

Summary

- `sf` relies on **GEOS**, **GDAL**, and **PROJ**
- These are all powerful, optimized **C/C++ libraries**
- For global geometry, **S2** provides a more accurate spherical model

Geometry functions

```r
st_geometry(nc)          # Get geometry column
st_geometry_type(nc)          # Get geometry types
st_drop_geometry(nc)            # Drops geometry column geometries
```

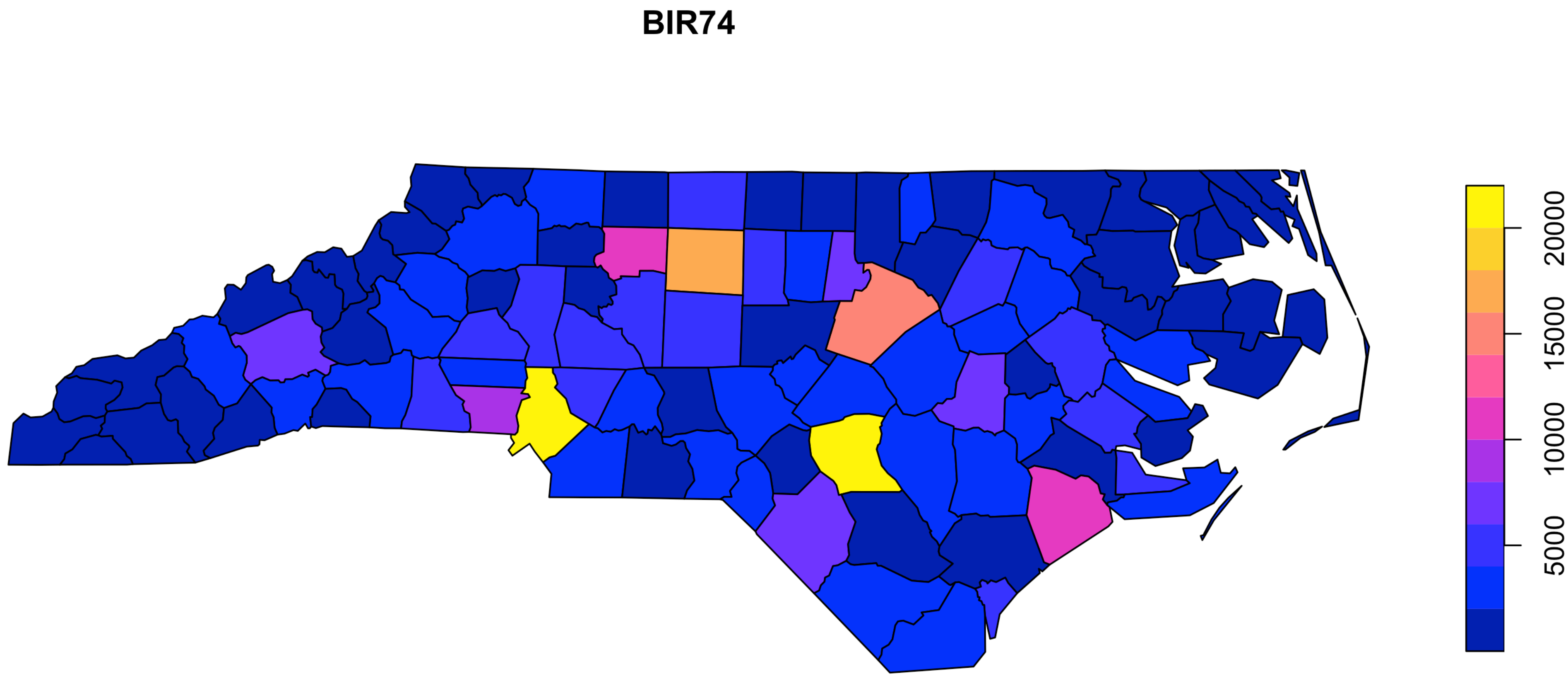
Coordinate Reference Systems

```r
st_crs(nc)                # Get CRS
st_transform(nc, 3857) # Reproject data
```

- CRS = Coordinate Reference System
- EPSG codes identify projections
- Set CRS with `st_set_crs` (note: without transformation)

Plotting with base R

```r
plot(nc["BIR74"])
```



Tidyverse integration

- Tidyverse "verbs" implemented for `sf` objects.

- Common verbs from **dplyr**
  - → `filter`
  - → `group_by`
  - → `mutate`
  - → `select`
  - → `reframe`
  - → `summarize`
  - → etc.

- Join verbs from **dplyr**
  - → `left_join`
  - → `right_join`
  - → `inner_join`
  - → `full_join`

Reading and writing vector data

Common Vector Data Formats

- **Shapefile (.shp)**
  - → Widely used
  - → Several files per dataset
  - → Limited attribute name length (10 chars), no UTF-8
- **GeoPackage (.gpkg)**
  - → Modern, single-file format
  - → Supports multiple layers
  - → Great default choice
- **GeoJSON (.geojson)**
  - → Text-based
  - → Web-friendly
  - → Supports UTF-8

Common Vector Data Formats (cont.)

- **KML (.kml)**
  - → For Google Earth
  - → Limited attribute support
- **CSV with coordinates**
  - → Easy to use
  - → Not a true spatial format

Spatial Data Formats: Pros & Cons

GeoPackage

- ✅ Modern, single-file format
- ✅ Supports vector and raster
- ✅ Good read/write performance

Shapefile

- ✅ Widely supported
- ✅ Good for legacy systems
- ❌ Multiple files per dataset
- ❌ Field name + size limits
- ❌ No support for NULL geometries

GeoJSON

- ✅ Human-readable
- ✅ Great for web mapping
- ✅ Easily handled by `geojsonio`, `sf`
- ❌ Slower for large datasets
- ❌ Only vector data

Reading Data with sf

Using `st_read()`

```r
library(sf)

# Read shapefile
shp = st_read("data/municipalities.shp")

# Read GeoPackage
gpkg = st_read("data/municipalities.gpkg")

# Read GeoJSON
geojson = st_read("data/municipalities.geojson")
```

Additional Notes

- File type inferred from extension
- Use `layer =` to select a specific layer from a GeoPackage
- `dsn =` can point to folders, zipped files, or URLs

Writing Data with sf

Using `st_write()`

```r
# Write to GeoPackage
st_write(shp, "output/municipalities.gpkg")

# Write to GeoJSON
st_write(shp, "output/municipalities.geojson")

# Write to Shapefile
st_write(shp, "output/municipalities.shp")
```

Writing Notes

- File type based on extension
- Some formats (like GeoJSON) may not support certain geometry types
- Shapefiles truncate field names — use GeoPackage if possible

Spatial joins and geometric operations

Example Data

We'll use:

- **Polygon layer**: Zion National Park (`zion`)
- **Point layer**: Observation points (`zion_points`)

```
library(sf)
library(dplyr)
library(tibble)
library(terra)
library(tmap)
library(spDataLarge)

# Load vector data
zion <- read_sf(system.file("vector/zion.gpkg", package = "spDataLarge"))
zion_points <- spDataLarge::zion_points



# Raster (only used to sample data from)
srtm <- rast(system.file("raster/srtm.tif", package = "spDataLarge"))

# Add elevation to points
```

```r
data("zion_points", package = "spDataLarge")
elevation = terra::extract(srtm, zion_points)
zion_points = cbind(zion_points, elevation)
```

Inspect the Data

```
# Spatial data
zion_points
```

```
Simple feature collection with 30 features and 2 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 304669.3 ymin: 4114963 xmax: 333820.2 ymax: 4145046
Projected CRS: UTM Zone 12, Northern Hemisphere
First 10 features:
   ID srtm                geometry
1   1 1802 POINT (329972.4 4118793)
2   2 2433 POINT (314663.3 4140486)
3   3 1886 POINT (320639.5 4133920)
4   4 1370 POINT (326053.6 4123660)
5   5 1452 POINT (323433.6 4119854)
6   6 1635 POINT (333082.3 4117960)
7   7 1380 POINT (318119.4 4123534)
8   8 2032 POINT (314418.6 4135720)
9   9 1830   POINT (319566 4131682)
10 10 1860 POINT (304669.3 4145046)
```

```
# Non-spatial data
point_data
```

```
# A tibble: 30 × 2
      ID category
   <dbl> <chr>
 1     1 B
 2     2 B
```

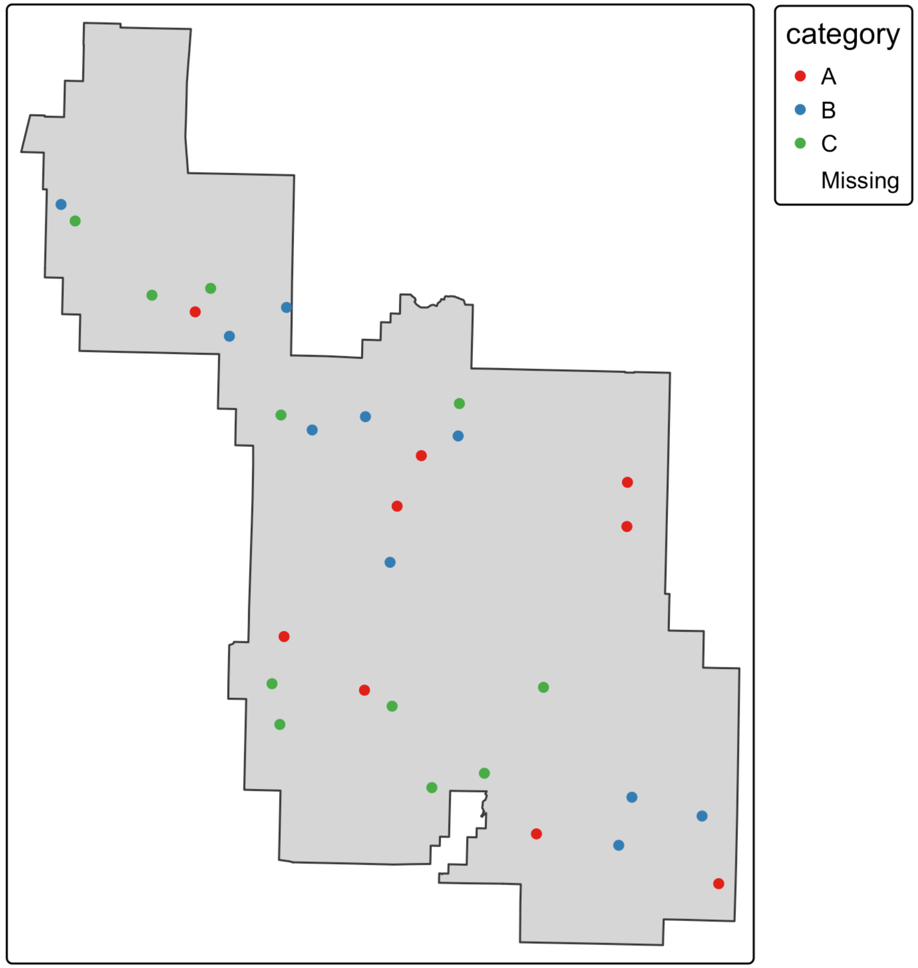
```
 3      3 A
 4      4 C
 5      5 C
 6      6 B
 7      7 A
 8      8 C
 9      9 A
10     10 B
# ℹ 20 more rows
```

Step 1: Attribute Join

```
points_joined <- zion_points %>%
  left_join(point_data, by = "ID")
```

Visualize the Result

```
tm_shape(zion) +
  tm_polygons() +
  tm_shape(points_joined) +
  tm_dots(col = "category", palette = "Set1")
```


category
A
B
C
Missing

Step 2: Spatial Join

```r
st_crs(zion_points) == st_crs(zion)  # Should be TRUE
```

```
[1] TRUE
```

```r
points_with_zion <- st_join(points_joined, zion)
```

Joining Strategies Summary

Type	Function	Based on
Attribute join	`left_join()`	Common column
Spatial join	`st_join()`	Geometric overlap

Common Pitfalls

- CRS mismatch
- Duplicates from one-to-many joins
- Dropping geometry when joining `sf` with `data.frame`

Avoid Dropping Geometry

❌ `left_join(df, sf_object)`
✅ `left_join(sf_object, df)`

Filtering Spatial Features

```
zion_subset <- zion %>%
  filter(grepl("Kolob", UNIT_NAME))
```

Spatial Relationships

```
st_intersects(zion_points, zion)
```

```
Sparse geometry binary predicate list of length 30, where the predicate
was `intersects'
first 10 elements:
 1: 1
 2: 1
 3: 1
 4: 1
 5: 1
 6: 1
 7: 1
 8: 1
 9: 1
 10: 1
```

```
st_within(zion_points, zion)
```

```
Sparse geometry binary predicate list of length 30, where the predicate
was `within'
first 10 elements:
 1: 1
 2: 1
 3: 1
 4: 1
 5: 1
 6: 1
 7: 1
 8: 1
```

```
 9: 1
 10: 1
```

```r
st_distance(zion_points, zion[1, ])
```

```
Units: [m]
      [,1]
 [1,]    0
 [2,]    0
 [3,]    0
 [4,]    0
 [5,]    0
 [6,]    0
 [7,]    0
 [8,]    0
 [9,]    0
[10,]    0
[11,]    0
[12,]    0
[13,]    0
[14,]    0
[15,]    0
[16,]    0
```

Geometry Operations

```
st_intersection(zion, zion)
st_union(zion)
st_difference(zion, zion[1, ])
```

Selecting and Renaming Columns

```
zion_points %>%
  select(id = ID, geometry)
```

Buffer Operation

```r
(point_buffers <- st_buffer(zion_points, dist = 500))
```

```
Simple feature collection with 30 features and 2 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: 304169.3 ymin: 4114463 xmax: 334320.2 ymax: 4145546
Projected CRS: UTM Zone 12, Northern Hemisphere
First 10 features:
   ID srtm                       geometry
1   1 1802 POLYGON ((330472.4 4118793,...
2   2 2433 POLYGON ((315163.3 4140486,...
3   3 1886 POLYGON ((321139.5 4133920,...
4   4 1370 POLYGON ((326553.6 4123660,...
5   5 1452 POLYGON ((323933.6 4119854,...
6   6 1635 POLYGON ((333582.3 4117960,...
7   7 1380 POLYGON ((318619.4 4123534,...
8   8 2032 POLYGON ((314918.6 4135720,...
9   9 1830 POLYGON ((320066 4131682, 3...
10 10 1860 POLYGON ((305169.3 4145046,...
```

Centroids

```r
(zion_centroids <- st_centroid(zion))
```

```
Simple feature collection with 1 feature and 11 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 320403.7 ymin: 4129876 xmax: 320403.7 ymax: 4129876
Projected CRS: UTM Zone 12, Northern Hemisphere
# A tibble: 1 × 12
  UNIT_CODE GIS_Notes        UNIT_NAME DATE_EDIT  STATE REGION GNIS_ID UNIT_TYPE
* <chr>     <chr>            <chr>     <date>     <chr> <chr>  <chr>   <chr>
1 ZION      Lands - http://… Zion Nat… 2017-06-22 UT    IM     1455157 National…
# ℹ 4 more variables: CREATED_BY <chr>, METADATA <chr>, PARKNAME <chr>,
#   geom <POINT [m]>
```

Points in Polygon (Intersection)

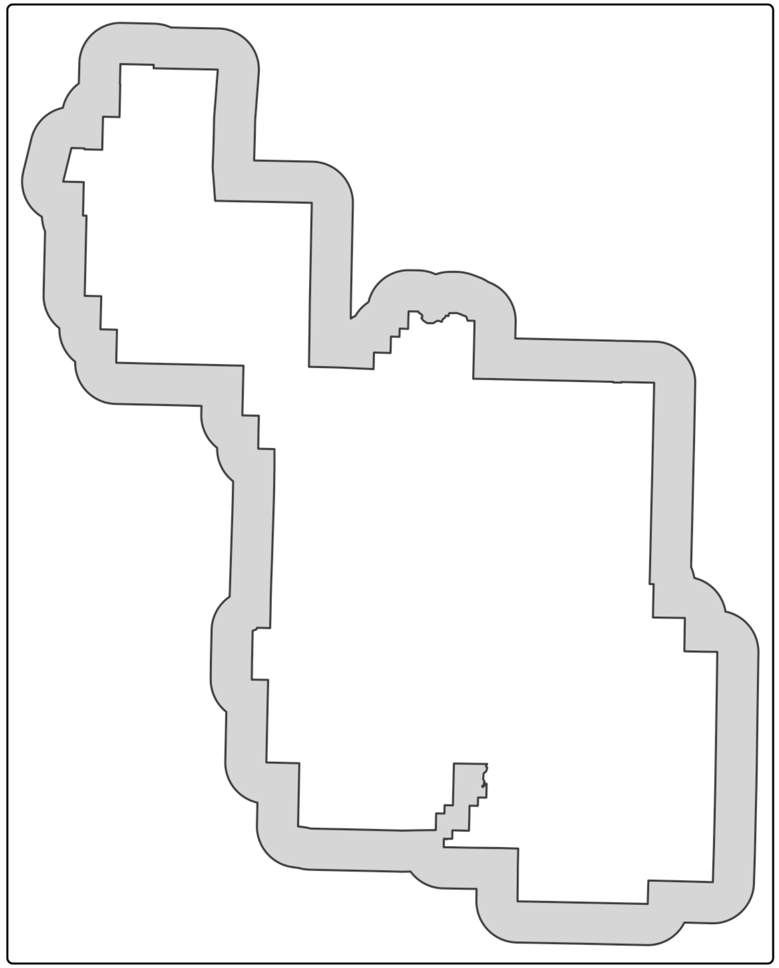
```
(points_in_zion <- st_intersection(zion_points, zion))
```

```
Simple feature collection with 30 features and 13 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 304669.3 ymin: 4114963 xmax: 333820.2 ymax: 4145046
Projected CRS: UTM Zone 12, Northern Hemisphere
First 10 features:
   ID srtm UNIT_CODE
1   1 1802      ZION
2   2 2433      ZION
3   3 1886      ZION
4   4 1370      ZION
5   5 1452      ZION
6   6 1635      ZION
7   7 1380      ZION
8   8 2032      ZION
9   9 1830      ZION
10 10 1860      ZION
                                                                             GIS Notes
```

Union & Difference Example

```
zion_union <- st_union(zion)
buffer_area <- st_buffer(zion_union, dist = 2000)
zion_gap <- st_difference(buffer_area, zion_union)

qtm(zion_gap)
```



Summarizing Spatial Data

```r
zion_points %>%
  summarize(elev = mean(srtm))
```

```
Simple feature collection with 1 feature and 1 field
Geometry type: MULTIPOINT
Dimension:     XY
Bounding box:  xmin: 304669.3 ymin: 4114963 xmax: 333820.2 ymax: 4145046
Projected CRS: UTM Zone 12, Northern Hemisphere
      elev                       geometry
1 1781.233 MULTIPOINT ((304669.3 41450...
```

Coordinate Transformation

```r
st_transform(zion, crs = 26912)  # UTM Zone 12N
```

```
Simple feature collection with 1 feature and 11 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: 302903.1 ymin: 4112244 xmax: 334735.5 ymax: 4153087
Projected CRS: NAD83 / UTM zone 12N
# A tibble: 1 × 12
  UNIT_CODE GIS_Notes        UNIT_NAME DATE_EDIT  STATE REGION GNIS_ID UNIT_TYPE
* <chr>     <chr>            <chr>     <date>     <chr> <chr>  <chr>   <chr>
1 ZION      Lands - http://… Zion Nat… 2017-06-22 UT    IM     1455157 National…
# ℹ 4 more variables: CREATED_BY <chr>, METADATA <chr>, PARKNAME <chr>,
#   geom <POLYGON [m]>
```

Checking and Fixing Geometry

```r
st_is_valid(zion)
```

```
[1] TRUE
```

```r
st_make_valid(zion)
```

```
Simple feature collection with 1 feature and 11 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: 302903.1 ymin: 4112244 xmax: 334735.5 ymax: 4153087
Projected CRS: UTM Zone 12, Northern Hemisphere
# A tibble: 1 × 12
  UNIT_CODE GIS_Notes        UNIT_NAME DATE_EDIT  STATE REGION GNIS_ID UNIT_TYPE
* <chr>     <chr>            <chr>     <date>     <chr> <chr>  <chr>   <chr>
1 ZION      Lands - http://… Zion Nat… 2017-06-22 UT    IM     1455157 National…
# ℹ 4 more variables: CREATED_BY <chr>, METADATA <chr>, PARKNAME <chr>,
#   geom <POLYGON [m]>
```

Recap

- sf is the core package for vector data in R
- Combines geometries and attributes in one tidy object
- Offers tools for reading, transforming, analyzing, and plotting

STOP