Advanced Spatial Methods I

Peter Ganong and Maggie Shi

February 9, 2026

Advanced Spatial Methods I

Skills acquired by the end of this lecture

- Understand how to structure a data-intensive spatial project
- Use `sjoin` to compare/combine two GeoDataFrames
- Use `points_from_xy` to convert regular dataset with xy coordinates into spatial
- Apply principles of data visualization to spatial data

Pre-processing

Pre-processing: roadmap

- Before jumping into spatial content, “meta”-discussion on how this directory is organized
- Discuss structure of `spatial/` folder
- Walk through `preprocessing.py`
- *Note: some of this has been covered in mini-lesson on how to structure your final project. Here we will now go in further detail about why this structure is useful.*

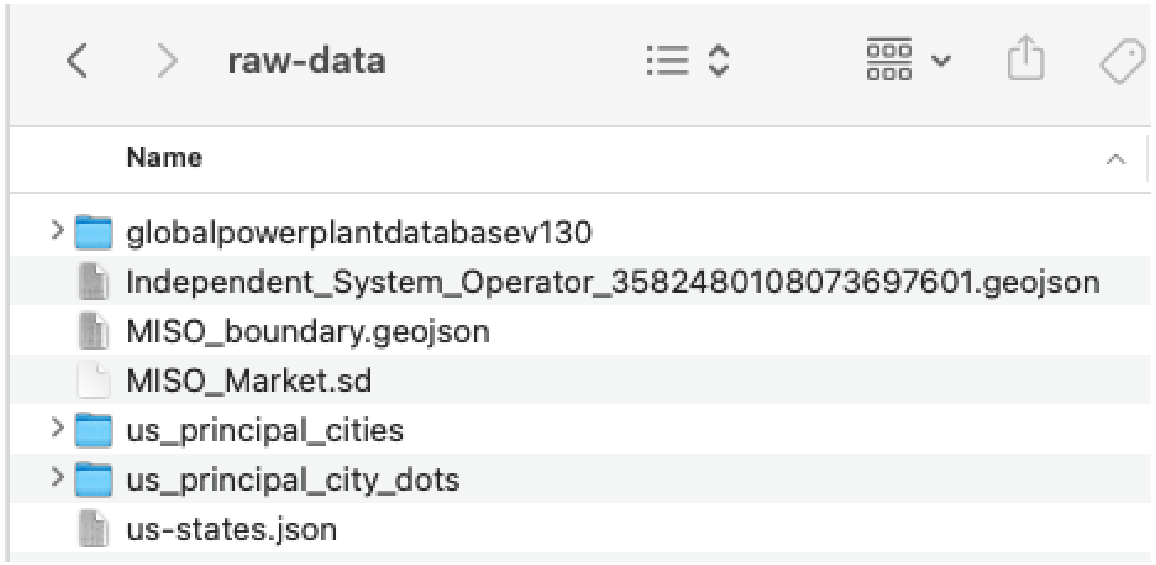
Structure of `spatial/` folder

- In the interest of time, we have pre-processed some of the raw data and shapefiles
- For your **projects** and **upcoming problem sets**, you should structure your data processing in a similar way (for all data, not just spatial)
- Why do this? **Replicability**, **transparency**, and **efficiency**

Structure of `spatial/` folder

- `data/raw-data/`: where raw datasets are stored
- `data/derived-data/`: where derived datasets are stored
- `preprocessing.py`: code to process raw into derived data
- `README.md`: documents original data sources for raw data and explains how they are processed

`spatial/raw-data/` folder



```
globalpowerplantdatabasev130
Independent_System_Operator_3582480108073697601.geojson
MISO_boundary.geojson
MISO_Market.sd
us_principal_cities
us_principal_city_dots
us-states.json
```

- These particular datasets are small enough (<100 MB) that they can be uploaded as part of Github repo
- But for larger datasets, you may have to share a Box/Dropbox/Google Drive folder and direct users to place data into `spatial/raw-data`

`preprocessing.py`: set path

```python
current_wd = os.getcwd()
print(f"Working directory is now: {current_wd}")
```

```
Working directory is now:
/Users/maggieshi/Documents/GitHub/winter2026/lectures/spatial
```

- Automatically detects and sets current working directory
- Allows user to fork + clone your repo and run it directly, rather than having to track down and update all the paths
- Use print statements liberally to check each step

`preprocessing.py`

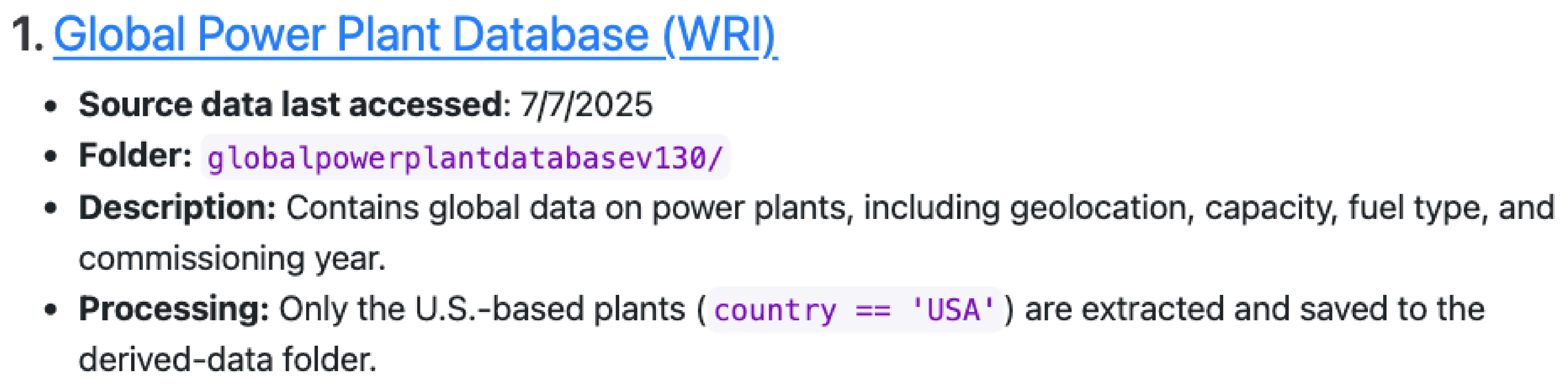
Example of code chunk to create `lower_48_states.geojson`

```python
output_path = "data/derived-data/lower_48_states.geojson"
if not os.path.exists(output_path):
    # code to process lower_48_states.geojson would be here
    print(f"GeoDataFrame successfully exported to: {output_path}")
else:
    print(f"Skipping: {output_path} already exists.")
```

```
Skipping: data/derived-data/lower_48_states.geojson already exists.
```

- **Efficiency**: to reduce runtime and avoid redundancy, check if output already exists in `derived-data/`
- Ensures that we run data processing code *only* if output does not already exist
- If you want to update file, must delete it and re-run

`README.md`



1. Global Power Plant Database (WRI)
   - **Source data last accessed**: 7/7/2025
   - **Folder:** `globalpowerplantdatabasev130/`
   - **Description:** Contains global data on power plants, including geolocation, capacity, fuel type, and commissioning year.
   - **Processing:** Only the U.S.-based plants (`country == 'USA'`) are extracted and saved to the derived-data folder.

- For replication and transparency, always link to source data or cite your source
  - If source data not publicly available: explain how to get access
  - URLs change often – including "last accessed" helps a future user troubleshoot why they can't find the data
- Include description of source data and processing to get derived data

Pre-processing: summary

- When you are working with more complex tasks and data, structure your project directory to carefully
- **Efficiency**: do computationally-intensive tasks once in `preprocessing.py`, and build checks to reduce redundancy
- **Replicability** and **transparency**: clear and cohesive `README.md`

Basic Spatial: Review

Roadmap: basic spatial review

- Introduce context: U.S. electricity markets
- Spatial review: packages, loading data, basic visualization

Intro to policy context + data: ISOs

- **Independent Systems Operators (ISOs)** are independent non-profit organizations that manage and coordinate the electric grid in the U.S.
  - Oversee and ensure reliable electricity transmission
  - Facilitate regional wholesale markets
- Each ISO covers a different geographic area, but not all their geographic boundaries are well-defined/follow state lines
- And not all regions in the U.S. covered by an ISO – only 2/3 of U.S. electricity load is served by an ISO

Basic spatial review: loading packages and data

```python
import geopandas as gpd
import matplotlib.pyplot as plt
from matplotlib import patheffects as pe # used later to format text
import shapely
iso_gdf = gpd.read_file('data/derived-data/Independent_System_Operator.geoj
```

- `geopandas` package can read in `.geojson`’s, in the same way that it reads in `.shp` and `.gpkg`
- `geojson`: JSON data type, but with spatial features like `geometry` and `coordinates`
- `iso_gdf` is a `GeoDataFrame`: a dataframe with spatial features + methods

Basic spatial review: load data and inspect

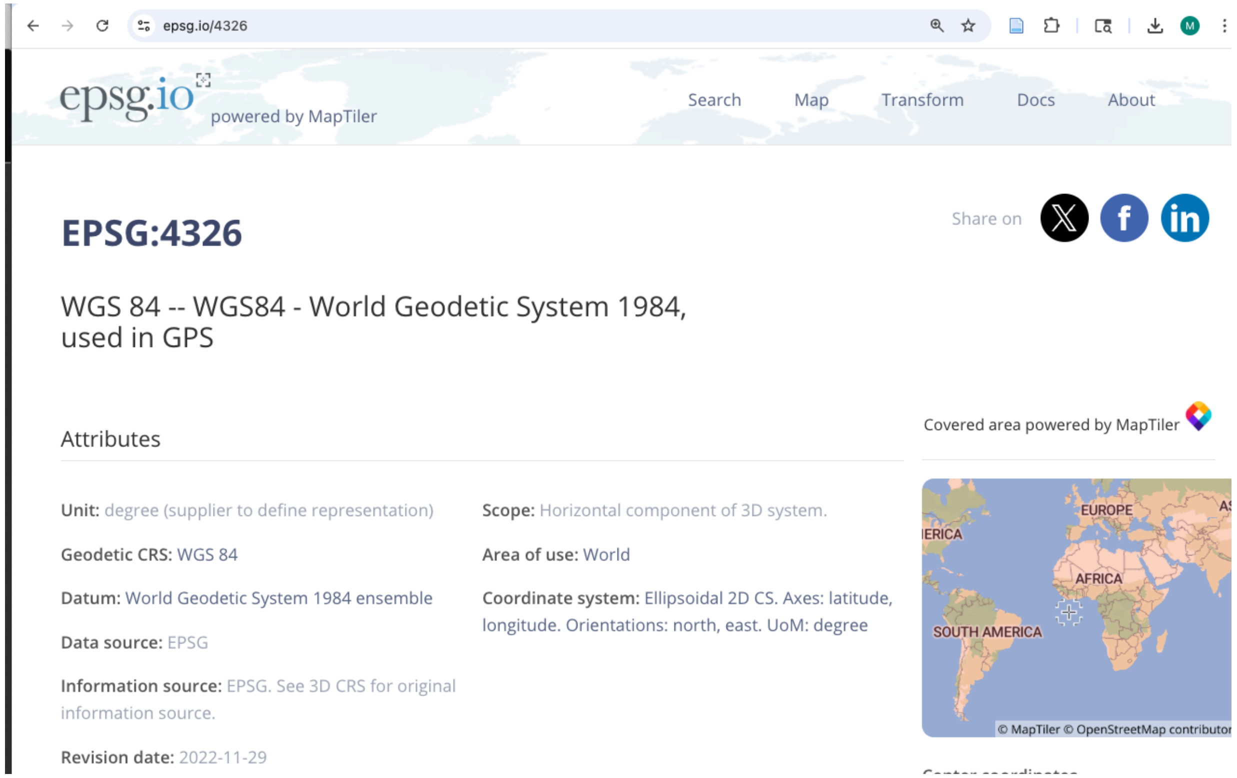
```python
# count geometry by type
print("Geometry type:", iso_gdf.geom_type)

# count total number of features
print("\n N features:", len(iso_gdf))
```

```
Geometry type: 0    MultiPolygon
1    MultiPolygon
2    MultiPolygon
3    MultiPolygon
4    MultiPolygon
5    MultiPolygon
6    MultiPolygon
dtype: object

 N features: 7
```

Basic spatial review: CRS

```python
# get CRS
print("\n Coordinate reference system:", iso_gdf.crs)
```

```
 Coordinate reference system: EPSG:4326
```


epsg.io/4326
epsg.io powered by MapTiler
Search
Map
Transform
Docs
About
Share on
Covered area powered by MapTiler
EUROPE
AFRICA
SOUTH AMERICA
© MapTiler © OpenStreetMap contributors

EPSG:4326

WGS 84 -- WGS84 - World Geodetic System 1984, used in GPS

Attributes

Unit: degree (supplier to define representation)

Geodetic CRS: WGS 84

Datum: World Geodetic System 1984 ensemble

Data source: EPSG

Information source: EPSG. See 3D CRS for original information source.

Revision date: 2022-11-29

Scope: Horizontal component of 3D system.

Area of use: World

Coordinate system: Ellipsoidal 2D CS. Axes: latitude, longitude. Orientations: north, east. UoM: degree

Source: epsg.io/4326

Basic spatial review: CRS

- Recall that **Coordinate Reference Systems (CRS)** tells GIS packages how to project the data into 2D representation
- With `.geojson`, the CRS is embedded directly into the file
- Later as we combine different files together, have to remember to make sure the CRS's are all aligned

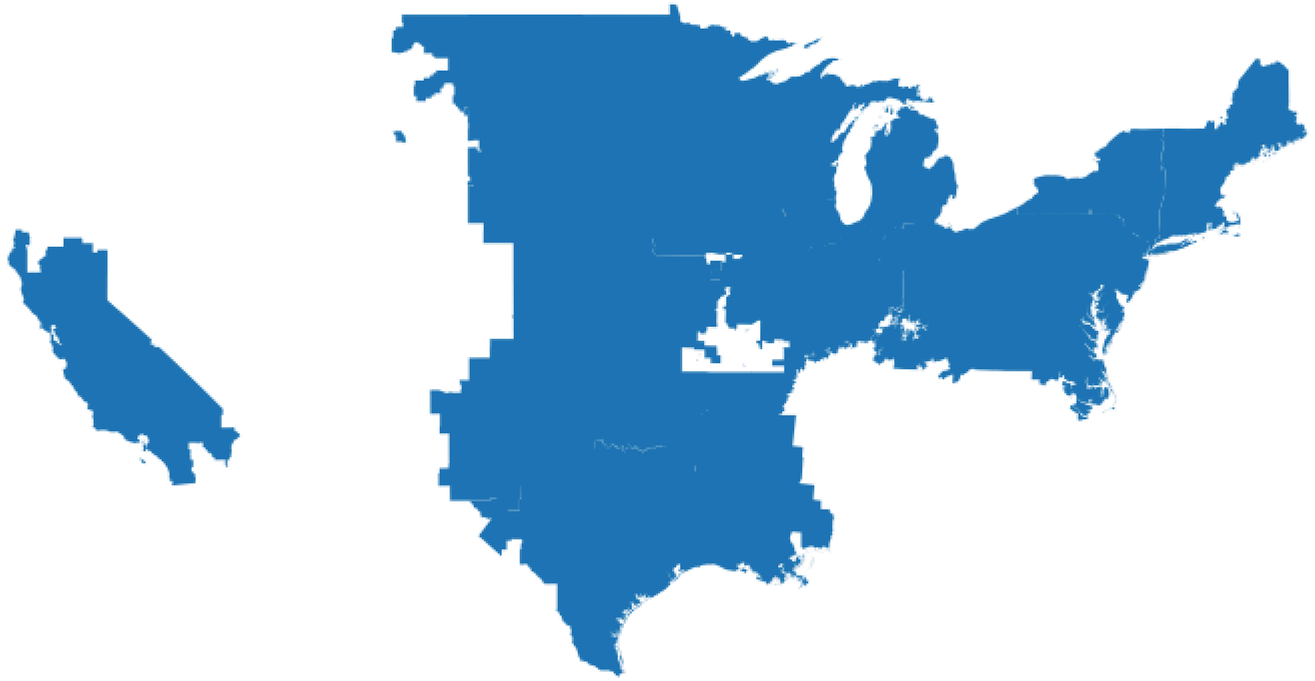
Basic spatial review: load data and inspect

```
print(iso_gdf.head())
```

```
   OBJECTID     ID                                               NAME  \
0         1  56669  Midcontinent Independent Transmission System O...
1         2  59504                                Southwest Power Pool
2         3  14725                            Pjm Interconnection, Llc
3         4   5723          Electric Reliability Council Of Texas, Inc.
4         5   2775              California Independent System Operator

                     ADDRESS         CITY STATE    ZIP       TELEPHONE  \
0  720 WEST CITY CENTER DRIVE       CARMEL    IN  46032  (317) 249-5400
1          201 WORTHEN DRIVE  LITTLE ROCK    AR  72223  (501) 614-3200
2       2750 MONROE BOULEVARD      AUDUBON    PA  19403  (610) 666-8980
3     7620 METRO CENTER DRIVE       AUSTIN    TX  78744  (512) 225-7000
4        250 OUTCROPPING WAY       FOLSOM    CA  95630  (916) 351-4400
```

Basic spatial review: basic visualization

```
iso_gdf.plot().set_axis_off()
```



Basic spatial review: summary

- Work with spatial data using `geopandas`
  - Load spatial file: `gpd.read_file()`
  - `.plot()` for basic visualization
- `.geojson` data type is a similar alternative to `.shp` and `.gpkg`
- Coordinate Reference System: tells `geopandas` how to project data

Advanced Spatial Methods: roadmap

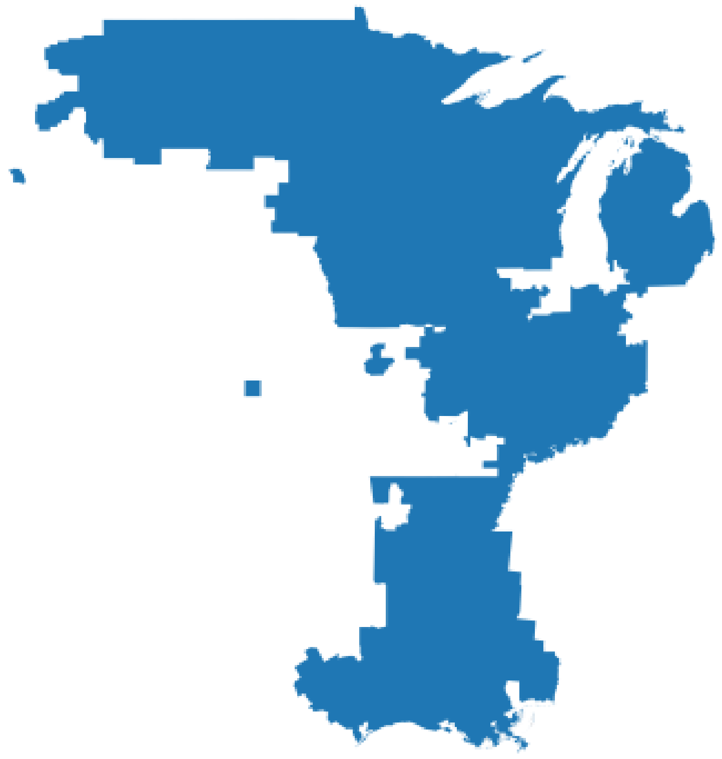
- Geometry objects like `GeoDataFrame` have a variety of methods that can…
  - Transform the object into something else: `centroid`, `union`
  - Make calculations: `length`, `area`
  - Combined separate objects together: `sjoin`
- The rest of the spatial lectures will be structured as case studies that explore use cases for these methods
- Next week, additional spatial package for dashboards: `pydeck`

Use case 1: characterize spatial relationships – roadmap

- Zoom in on ISO covering Midwest: **MISO**
- **Motivating question**: which states does MISO cover (at least) part of?
- `sjoin`

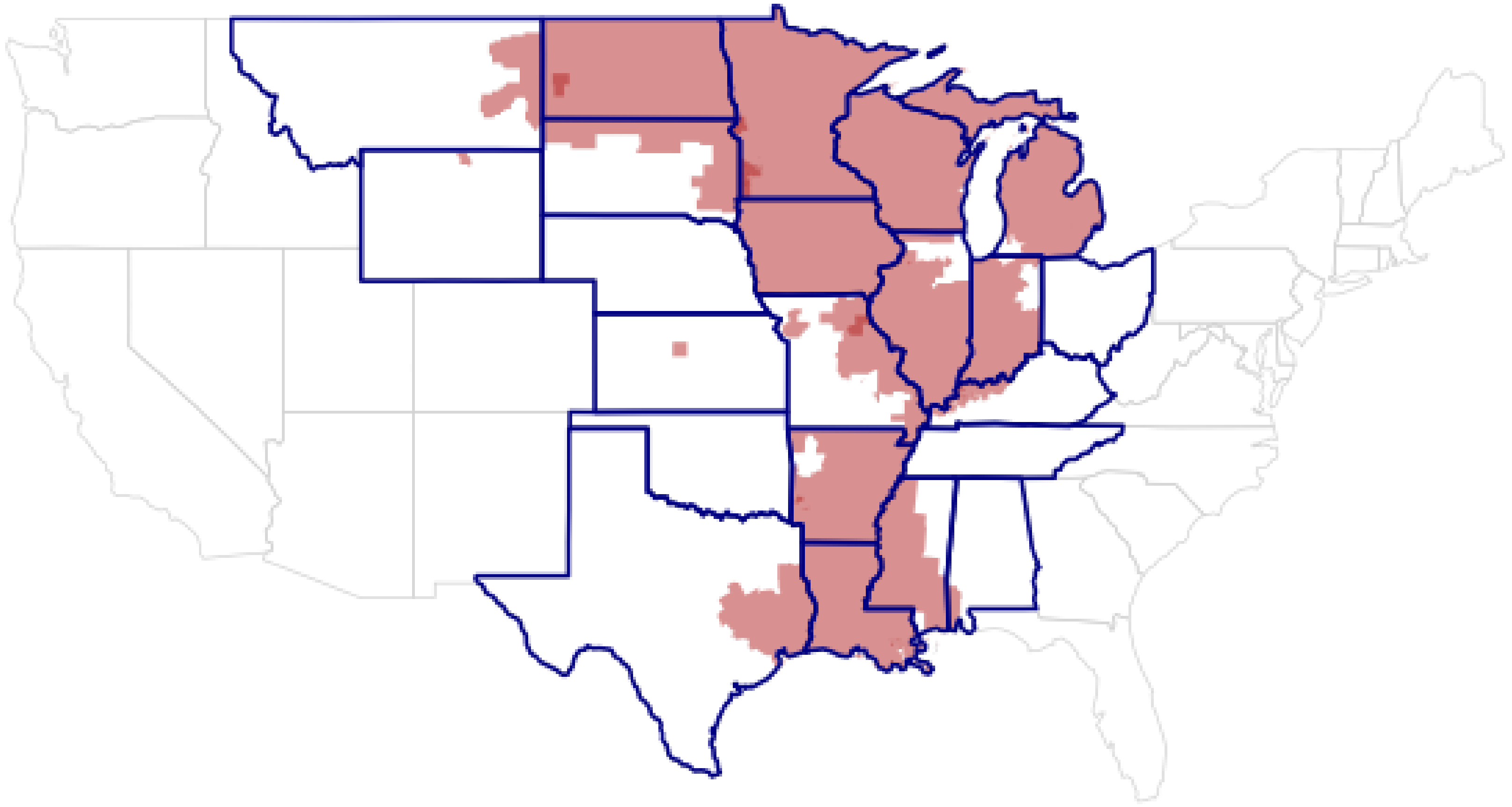
Zooming in on MISO

```python
# subset to MISO
miso_gdf = iso_gdf[iso_gdf["NAME"] ==
    "Midcontinent Independent Transmission System Operator, Inc.."]
miso_gdf.plot().set_axis_off()
```



Which states does MISO cover part of?

End goal:



Use case 1: characterize spatial relationships

- We can use spatial methods to compare two maps: MISO and state maps
- Want to know: **which states does MISO cover (at least) part of?**

Use case 1: characterize spatial relationships

First, let's load and inspect the state map we want to join with MISO: `lower_48_states.geojson`

```python
us_states_gdf = gpd.read_file('data/derived-data/lower_48_states.geojson')
us_states_gdf.crs = iso_gdf.crs
print(us_states_gdf.head())
```

```
   id        name  density                                           geometry
0  01     Alabama    94.65  POLYGON ((-87.3593 35.00118, -85.60668 34.9847...
1  04     Arizona    57.05  POLYGON ((-109.0425 37.00026, -109.04798 31.33...
2  05    Arkansas    56.43  POLYGON ((-94.47384 36.50186, -90.15254 36.496...
3  06  California   241.70  POLYGON ((-123.23326 42.00619, -122.37885 42.0...
4  08    Colorado    49.33  POLYGON ((-107.91973 41.00391, -105.72895 40.9...
```

- The states are denoted by `name`
- `us_states_gdf.crs = iso_gdf.crs` sets the two CRS's to match

Which states does MISO cover? – `sjoin`

- `sjoin` takes two GeoDataFrames and *creates a new one*
- Resulting gdf has variables from both dataframes
- Inputs to `sjoin`:
  - Left `GeoDataFrame`, right `GeoDataFrame`
  - `how`: `left`, `right`, `inner`
  - `predicate`: `intersects` (default), `contains`, `within`, `overlaps`…

`sjoin`: `how`

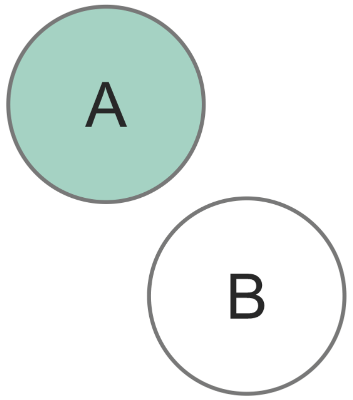
`how`: determines *which* rows of *which* gdf survive

- `left`: all rows of **left** gdf survive, regardless of if they successfully join
- `inner`: *only* rows of **left** gdf that successfully join survive
- `right`: all rows of **right** gdf survive, regardless of if they successfully join

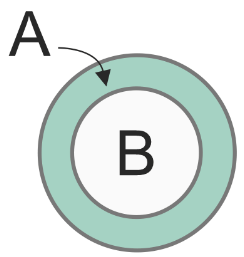
Note: if there is a 1:m join, the `sjoin` will duplicate rows

`sjoin`: `predicate`

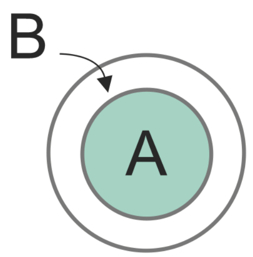
`predicate` captures the type of spatial join



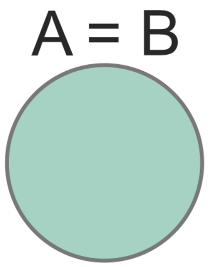
disjoint
"A and B are fully disconnected"



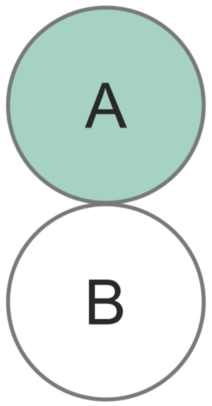
contains
"A contains B"



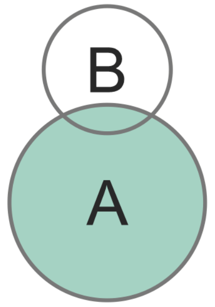
within
"A is within B"



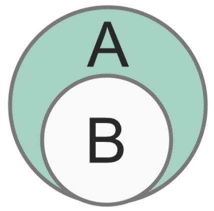
equals
"A and B are topologically equal"



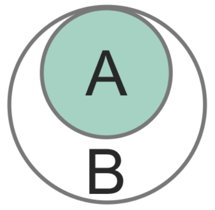
touches
"A and B touch at border"



overlaps
"A and B overlap each other"



covers
"A covers B"



covered by
"A is covered by B"

Which states does MISO cover?

`intersects` is a more generalized version of `overlaps` that allows for different types of object types

```python
states_with_any_miso = gpd.sjoin(us_states_gdf,
    miso_gdf,
    how='inner',
    predicate='intersects')

print("Object type:", type(states_with_any_miso))
print("\nGeometry type:", states_with_any_miso.geom_type.value_counts())
```

```
Object type: <class 'geopandas.geodataframe.GeoDataFrame'>

Geometry type: Polygon         21
MultiPolygon     1
Name: count, dtype: int64
```

Which states does MISO cover?

```
print(states_with_any_miso.head())
```

```
   id      name  density                                           geometry  
\
0  01   Alabama    94.65  POLYGON ((-87.3593 35.00118, -85.60668 34.9847...
2  05  Arkansas    56.43  POLYGON ((-94.47384 36.50186, -90.15254 36.496...
11 17  Illinois   231.50  POLYGON ((-90.63998 42.51006, -88.78878 42.493...
12 18   Indiana   181.70  POLYGON ((-85.99006 41.75972, -84.80704 41.759...
13 19      Iowa    54.81  POLYGON ((-91.36842 43.50139, -91.21506 43.501...

   index_right  OBJECTID     ID  \
0            0         1  56669
2            0         1  56669
11           0         1  56669
12           0         1  56669
13           0         1  56669
```

```
print("States covered by MISO (any amount):", states_with_any_miso["name"].
```

```
States covered by MISO (any amount): ['Alabama', 'Arkansas', 'Illinois',
'Indiana', 'Iowa', 'Kansas', 'Kentucky', 'Louisiana', 'Michigan', 'Minnesota',
'Mississippi', 'Missouri', 'Montana', 'Nebraska', 'North Dakota', 'Ohio',
'Oklahoma', 'South Dakota', 'Tennessee', 'Texas', 'Wisconsin', 'Wyoming']
```

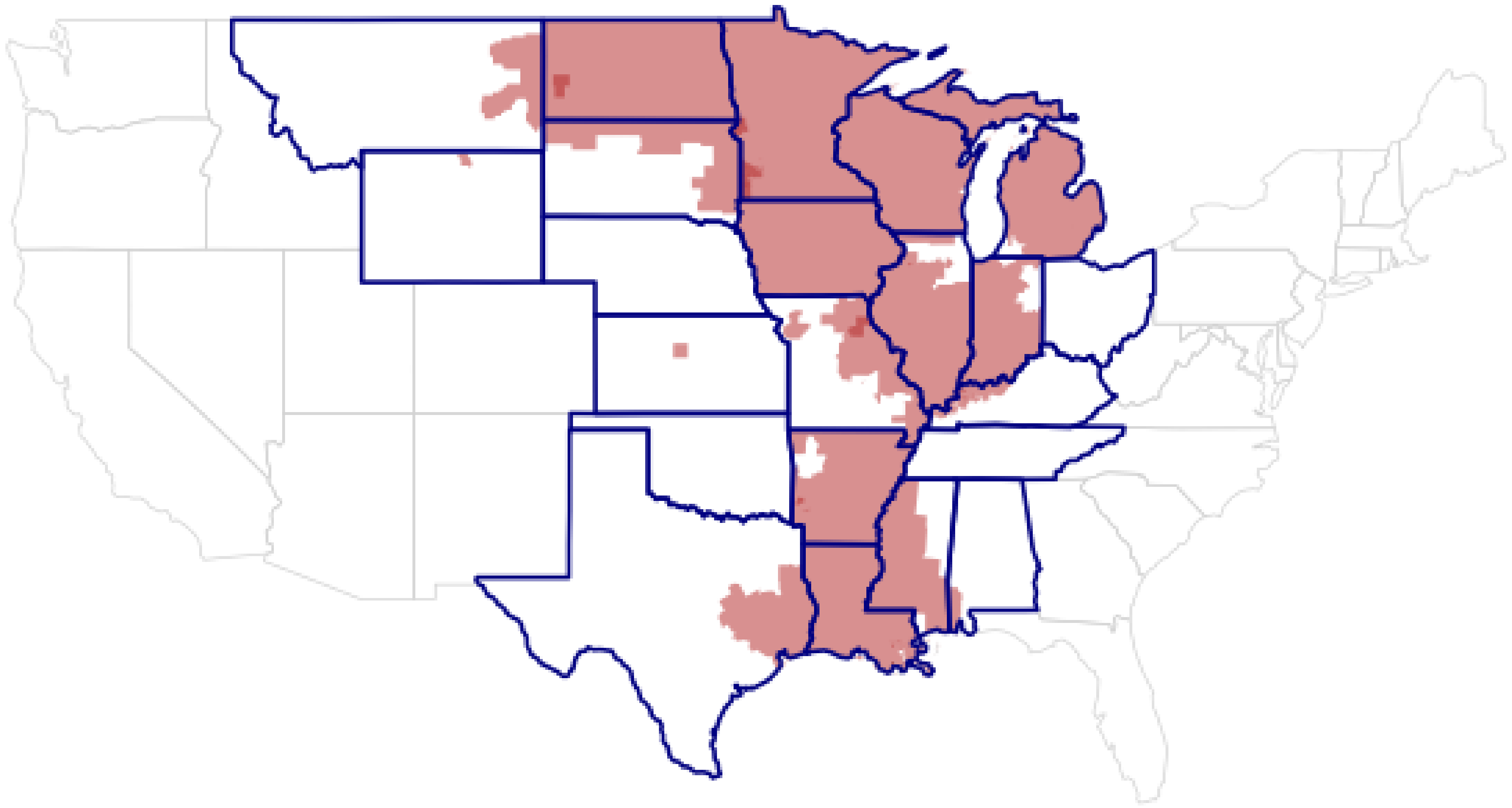
Plot for visual confirmation

For more complicated/layered spatial plots, use `matplotlib`

```python
# set up plot
fig, ax = plt.subplots(figsize=(8, 6))

# plot MISO in red
miso_gdf.plot(ax=ax, color='firebrick', alpha=0.5)

# plot all states in gray boundary
us_states_gdf.boundary.plot(ax=ax, color='lightgray', linewidth=0.5)

# plot inner join states in navy boundary
states_with_any_miso.boundary.plot(ax=ax, edgecolor='navy', linewidth=1.2)

ax.set_axis_off()
plt.show()
```

We will cover more on making production-quality maps next lecture

Plot for visual confirmation



Use case 1: characterize spatial relationships – summary

- Can use spatial methods to compare different maps and characterize spatial relationships in a systematic way
- Example: which states does MISO cover (at least) part of?
- Method: `sjoin`
- Use `matplotlib` for more complicated spatial maps

Use Case 2: communicating spatial patterns

Use case 2: communicating spatial patterns – roadmap

- **Motivating Question:** what is the spatial distribution of power plants by fuel type in the U.S.?
- Load new power plant dataset from **CSV**
- Convert to GeoDataframe using coordinates: `points_from_xy`
- Explore different ways of encoding categorical information to answer question

Context: power plants

- We will next introduce a dataset on U.S. power plants: location, fuel type, and electricity generation
- Most power plants use a **primary fuel type** – natural gas, coal, nuclear, etc. – to generate electricity
- Power generation is measured in **gigawatt-hours** (GWh)
    - 1 GWh roughly powers 700 homes for a year

A new dataset: US power plants

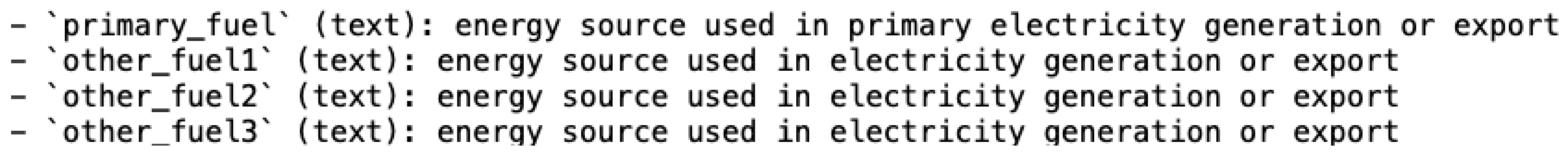
```python
us_powerplant_df = pd.read_csv('data/derived-data/us_gppd.csv')
print(us_powerplant_df.columns)
```

Note that this is *not* a spatial file type, but just a CSV

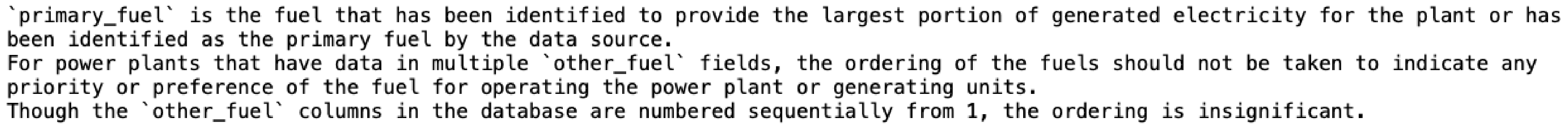
```
Index(['country', 'country_long', 'name', 'gppd_idnr', 'capacity_mw',
       'latitude', 'longitude', 'primary_fuel', 'other_fuel1', 'other_fuel2',
       'other_fuel3', 'commissioning_year', 'owner', 'source', 'url',
       'geolocation_source', 'wepp_id', 'year_of_capacity_data',
       'generation_gwh_2013', 'generation_gwh_2014', 'generation_gwh_2015',
       'generation_gwh_2016', 'generation_gwh_2017', 'generation_gwh_2018',
       'generation_gwh_2019', 'generation_data_source',
       'estimated_generation_gwh_2013', 'estimated_generation_gwh_2014',
       'estimated_generation_gwh_2015', 'estimated_generation_gwh_2016',
       'estimated_generation_gwh_2017', 'estimated_generation_note_2013',
       'estimated_generation_note_2014', 'estimated_generation_note_2015',
       'estimated_generation_note_2016', 'estimated_generation_note_2017'],
      dtype='object')
```

Exploring us_gppd.csv

We are interested in fuel types. From the README.txt:



```
- `primary_fuel` (text): energy source used in primary electricity generation or export
- `other_fuel1` (text): energy source used in electricity generation or export
- `other_fuel2` (text): energy source used in electricity generation or export
- `other_fuel3` (text): energy source used in electricity generation or export
```



```
`primary_fuel` is the fuel that has been identified to provide the largest portion of generated electricity for the plant or has
been identified as the primary fuel by the data source.
For power plants that have data in multiple `other_fuel` fields, the ordering of the fuels should not be taken to indicate any
priority or preference of the fuel for operating the power plant or generating units.
Though the `other_fuel` columns in the database are numbered sequentially from 1, the ordering is insignificant.
```

So we are interested in studying primary_fuel

```python
print(us_powerplant_df["primary_fuel"].unique())
```

```
['Solar' 'Gas' 'Oil' 'Hydro' 'Wind' 'Coal' 'Biomass' 'Waste' 'Storage'
 'Cogeneration' 'Geothermal' 'Petcoke' 'Nuclear' 'Other']
```

Exploring `us_gppd.csv`

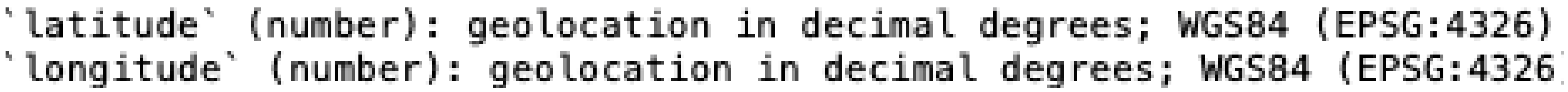
```python
summary = (
    us_powerplant_df
    .groupby("primary_fuel", as_index=False)["generation_gwh_2019"]
    .sum()
    .rename(columns={"generation_gwh_2019": "total_generation_gwh_2019"})
    .sort_values(by="total_generation_gwh_2019", ascending=False)
)

print(summary)
```

```
   primary_fuel  total_generation_gwh_2019
3           Gas               1.577508e+06
1          Coal               9.357724e+05
6       Nuclear               8.071456e+05
13         Wind               2.946884e+05
5         Hydro               2.804984e+05
10        Solar               7.142216e+04
12        Waste               4.378694e+04
0       Biomass               2.570438e+04
4    Geothermal               1.554570e+04
7           Oil               1.483142e+04
9       Petcoke               6.967579e+03
```

```
2   Cogeneration                   3.888141e+03
8          Other                   1.155774e+03
```

Loading `us_gppd.csv` as a `GeoDataFrame`

- Note that we’ve only imported this CSV dataset as a `pandas` dataframe up until now
- But the dataset contains spatial coordinates: `latitude` and `longitude`
- From the `README.txt`:



```
`latitude` (number): geolocation in decimal degrees; WGS84 (EPSG:4326)
`longitude` (number): geolocation in decimal degrees; WGS84 (EPSG:4326
```

Loading `us_gppd.csv` as a `GeoDataFrame`

```python
us_powerplant_gdf = gpd.GeoDataFrame(
    geometry=gpd.points_from_xy(
        us_powerplant_df.longitude,
        us_powerplant_df.latitude,
        crs="EPSG:4326"),
    data=us_powerplant_df)

print("Geometry type:", us_powerplant_gdf.geom_type.value_counts())
```

```
Geometry type: Point    9581
Name: count, dtype: int64
```

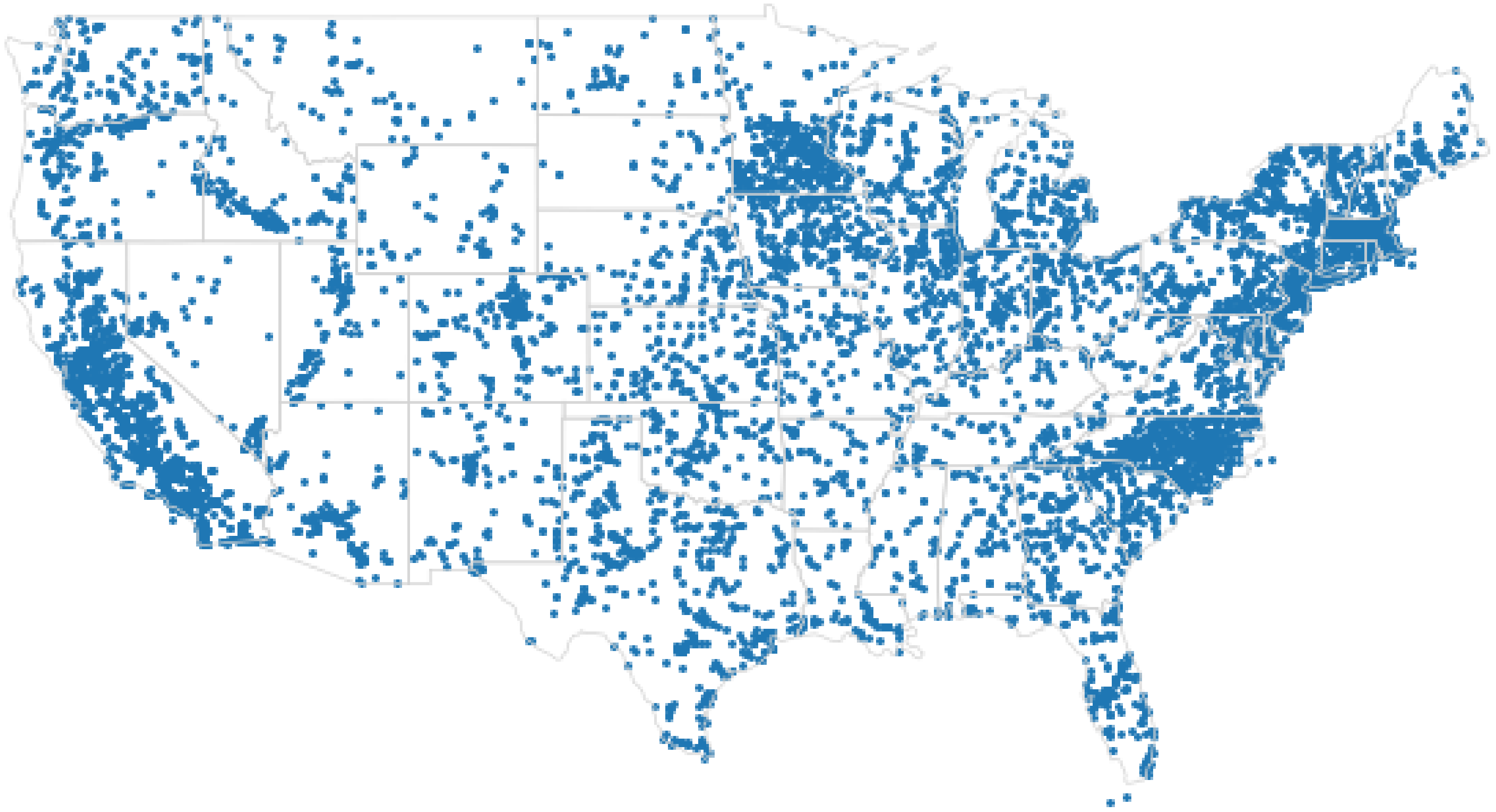
Visualizing `us_powerplant_gdf`

- Once we confirm that `us_powerplant_gdf` is on the same CRS as `us_states_gdf`
- We can plot one directly on the other

```python
assert(us_states_gdf.crs == us_powerplant_gdf.crs)
print("I've asserted that the CRS's match.")

fig, ax = plt.subplots(figsize=(8, 6))
us_states_gdf.boundary.plot(ax=ax, color='lightgray', linewidth=0.5)
us_powerplant_gdf.plot(ax=ax, markersize=1.5)
ax.set_axis_off()
plt.show()
```

Visualizing `us_powerplant_gdf`

```
I've asserted that the CRS's match.
```



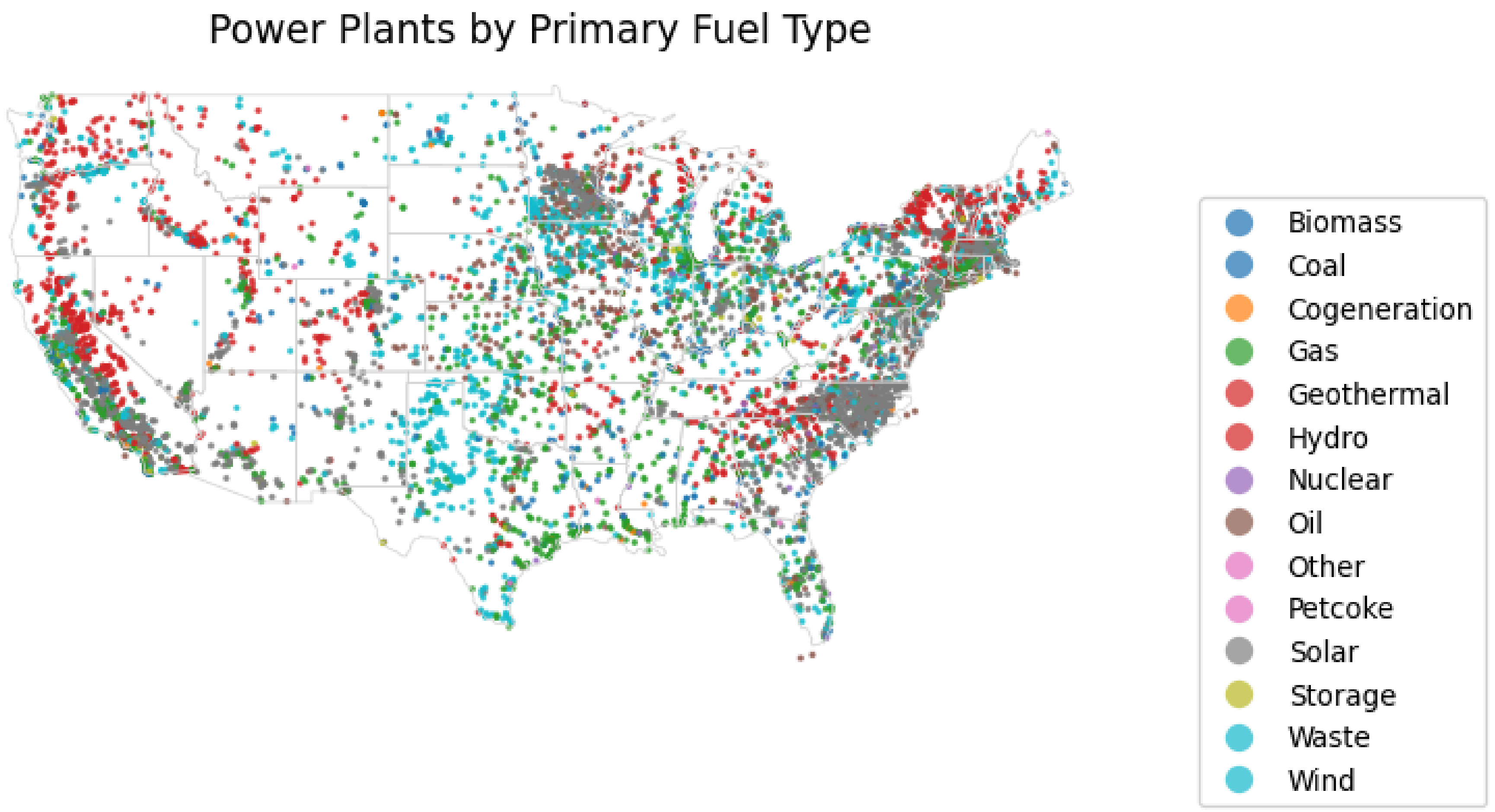
Visualizing `us_powerplant_gdf`

- Recall our **motivating question**: what is the spatial distribution of power plants by fuel type in the U.S.?
- We have the ingredients to answer this question now:
  - `latitude` and `longitude` of power plants (converted into `GeoDataFrame`)
  - `primary_fuel`: categorical attribute
- As a first pass: let’s encode `primary_fuel` in different colors and plot

Visualizing `us_powerplant_gdf`

```python
fig, ax = plt.subplots(figsize=(8, 6))
us_states_gdf.boundary.plot(ax=ax, color='lightgray', linewidth=0.5)

us_powerplant_gdf.plot(
    ax=ax,
    column='primary_fuel',
    categorical=True,
    legend=True,
    markersize=2,
    alpha=0.7
)
ax.set_axis_off()
ax.set_title("Power Plants by Primary Fuel Type", fontsize=14)
plt.tight_layout()
plt.show()
```

Take 1: encoding fuels with color



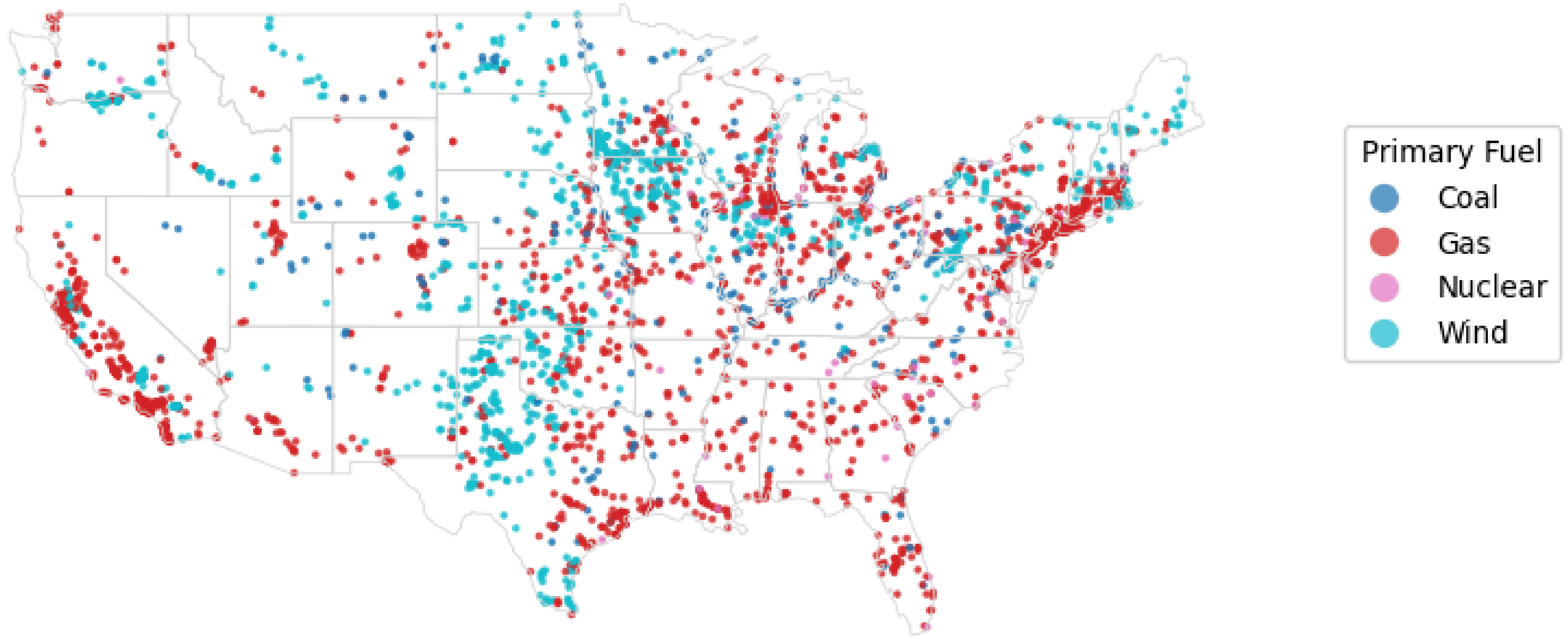
Question: what is the headline of this map? Sub-messages?

Take 2: encoding top 4 fuels with color

The map on the previous slide is too cluttered. Let’s restrict just to the top 4 fuel types.

```
# restrict to top 4 fuels by total gwh
top_fuels = summary["primary_fuel"].head(4)
filtered_powerplant_gdf = us_powerplant_gdf[us_powerplant_gdf["primary_fuel

fig, ax = plt.subplots(figsize=(8, 6))
us_states_gdf.boundary.plot(ax=ax, color='lightgray', linewidth=0.5)

# loop through each fuel type and plot
for fuel_type, group in filtered_powerplant_gdf.groupby("primary_fuel"):
    group.plot(ax=ax, markersize=3, label=fuel_type,alpha=0.7)

# define legend
ax.legend(title="Primary Fuel", fontsize=8, title_fontsize=8, loc="lower ri
ax.set_axis_off()
plt.tight_layout()
plt.show()
```

Take 2: encoding top 4 fuels with color



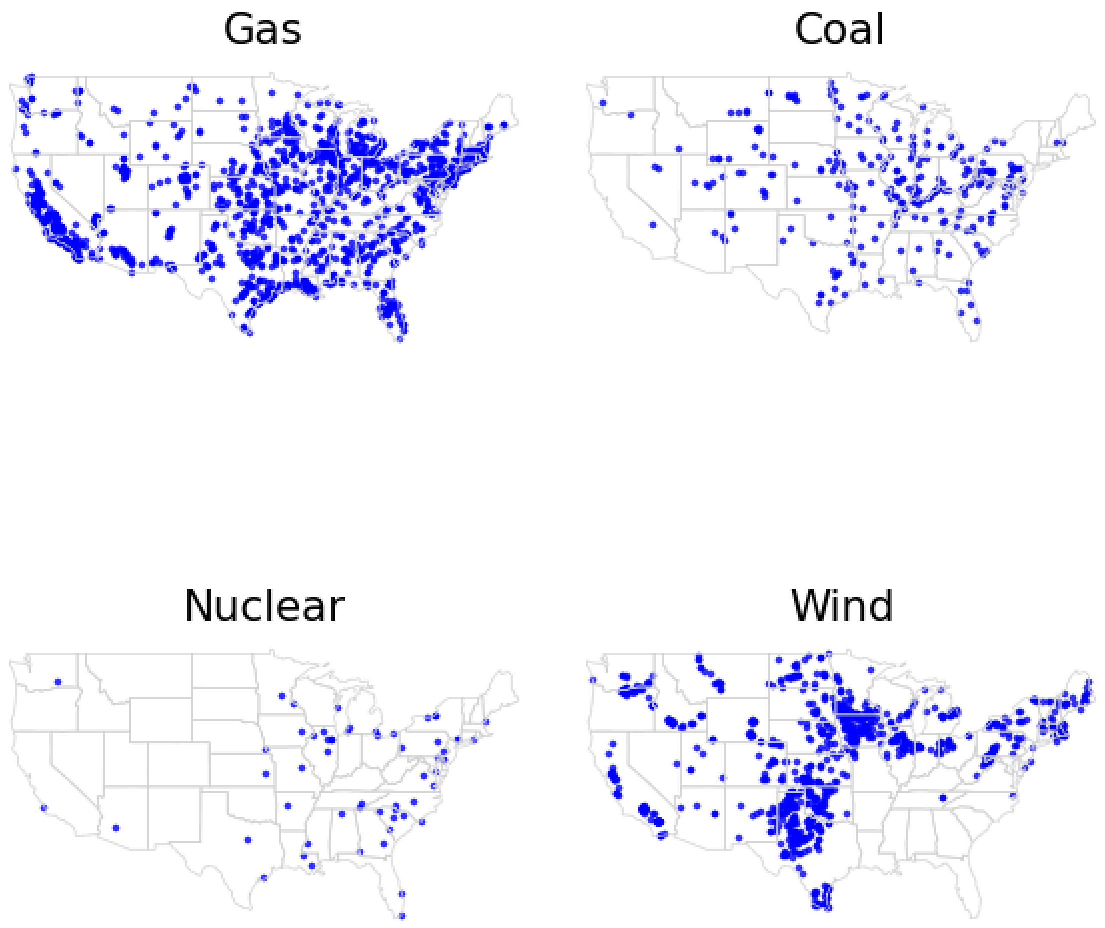
Discussion question: what headlines/submessages come out now?

Take 3: encoding with small multiples

Recall from visualization lectures: an alternative to color is *small multiples*:

```python
fig, axes = plt.subplots(2, 2, figsize=(6, 6), constrained_layout=True) # 2
axes = axes.flatten() # flatten for iteration

for ax, fuel_type in zip(axes, summary["primary_fuel"].head(4)):
    group = filtered_powerplant_gdf[filtered_powerplant_gdf["primary_fuel"]
    group.plot(ax=ax, markersize=3, alpha=0.7, color='blue')
    us_states_gdf.boundary.plot(ax=ax, color='lightgray', linewidth=0.5)
    ax.set_title(fuel_type, fontsize=16)
    ax.set_axis_off()
plt.show()
```

Take 3: encoding with small multiples



Discussion questions: what are the headlines and submessages now?

Do-pair-share

Motivating question: create a small-multiples plot to depict the spatial distribution of power plants by fuel type (top 4) in *Illinois*.

Hints:

- Subset `us_states_gdf` to just Illinois
- Then use `.sjoin` from previous case study

```
print(us_states_gdf.head())
```

```
   id        name  density                                           geometry
0  01     Alabama    94.65  POLYGON ((-87.3593 35.00118, -85.60668 34.9847...
1  04     Arizona    57.05  POLYGON ((-109.0425 37.00026, -109.04798 31.33...
2  05    Arkansas    56.43  POLYGON ((-94.47384 36.50186, -90.15254 36.496...
3  06  California   241.70  POLYGON ((-123.23326 42.00619, -122.37885 42.0...
4  08    Colorado    49.33  POLYGON ((-107.91973 41.00391, -105.72895 40.9...
```

Do-pair-share: starter code

Code to plot small-multiples of top 4 fuel types nationally:

spatial1_dps.qmd

```python
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import patheffects as pe # used later to format text
import shapely

# load geodata
us_states_gdf = gpd.read_file('data/derived-data/lower_48_states.geojson')
us_powerplant_df = pd.read_csv('data/derived-data/us_gppd.csv')
us_powerplant_gdf = gpd.GeoDataFrame(
    geometry=gpd.points_from_xy(
    us_powerplant_df.longitude,
    us_powerplant_df.latitude,
    crs="EPSG:4326"),
    data=us_powerplant_df)

# summarize top fuels by gwh in 2019
summary = (
```

Use case 2: communicating spatial patterns – summary

- `points_from_xy` converts datasets with `latitude` and `longitude` into spatial data
- There are many ways to visualize spatial patterns – a lot the rules and concepts from standard data viz apply here too
  - Plotting as much information as possible is not necessarily optimal
  - Experiment with different ways of encoding information