Data Visualization

INF552 - 2023 - Session 06 - exercices
Geovisualization with D3 and Vega-lite

ÉCOLE POLYTECHNIQUE

Inria

emmanuel.pietriga@inria.fr

olivier.gladin@inria.fr

D3 - Defining a symbol and using it

- Instead of tediously drawing shapes with SVG elements,
  use D3 symbols for circles, squares, triangles, crosses, *etc*.

```
let circleGenerator = d3.symbol().type(d3.symbolCircle)
                        .size(6);

d3.selectAll("path")
  .data(someData)
  .enter()
  .append("path")
  .attr("d", circleGenerator());
```

- The above symbols would be positioned in `(0,0)` by default.
  Use affine transforms to move them to the write place.

```
  .attr("transform", function(d){return "translate(" + d.x + "," + d.y + ")";});
```

https://github.com/d3/d3-shape/blob/master/README.md#symbol

- Syntactic sugar for strings:

```
(d) => (`translate(${d.x}, ${d.x})`)
```

d3-geo

D3 symbol generators seen in s#02:

```
let triangleGen = d3.symbol().type(d3.symbolTriangle).size(12);

d3.selectAll("path")
  .data(...)
  .enter()
  .append("path")
  .attr("d", triangleGen())
  .attr("transform", function(d){
      return "translate(" + d.x + "," + d.y + ")";
  })
  .attr("stroke", "#DDD");
```

`d3.geoPath()`: geographic path generator

```
let geoPathGen = d3.geoPath();

// where geoData is a geojson file parsed with d3.json()
d3.selectAll("path")
  .data(geoData.features)
  .enter()
  .append("path")
  .attr("d", geoPathGen)
  .attr("stroke", "#DDD");
```

d3-geo

- Combined into a single FeatureCollection, drawn as a single **`<path>`**:

```
someElement.append("path")
           .datum({type: "FeatureCollection", features: features})
           .attr("d", d3.geoPath());
```

- Many useful **`geoPath`** methods: **`area()`**, **`bounds()`**, **`centroid()`**, *etc.*

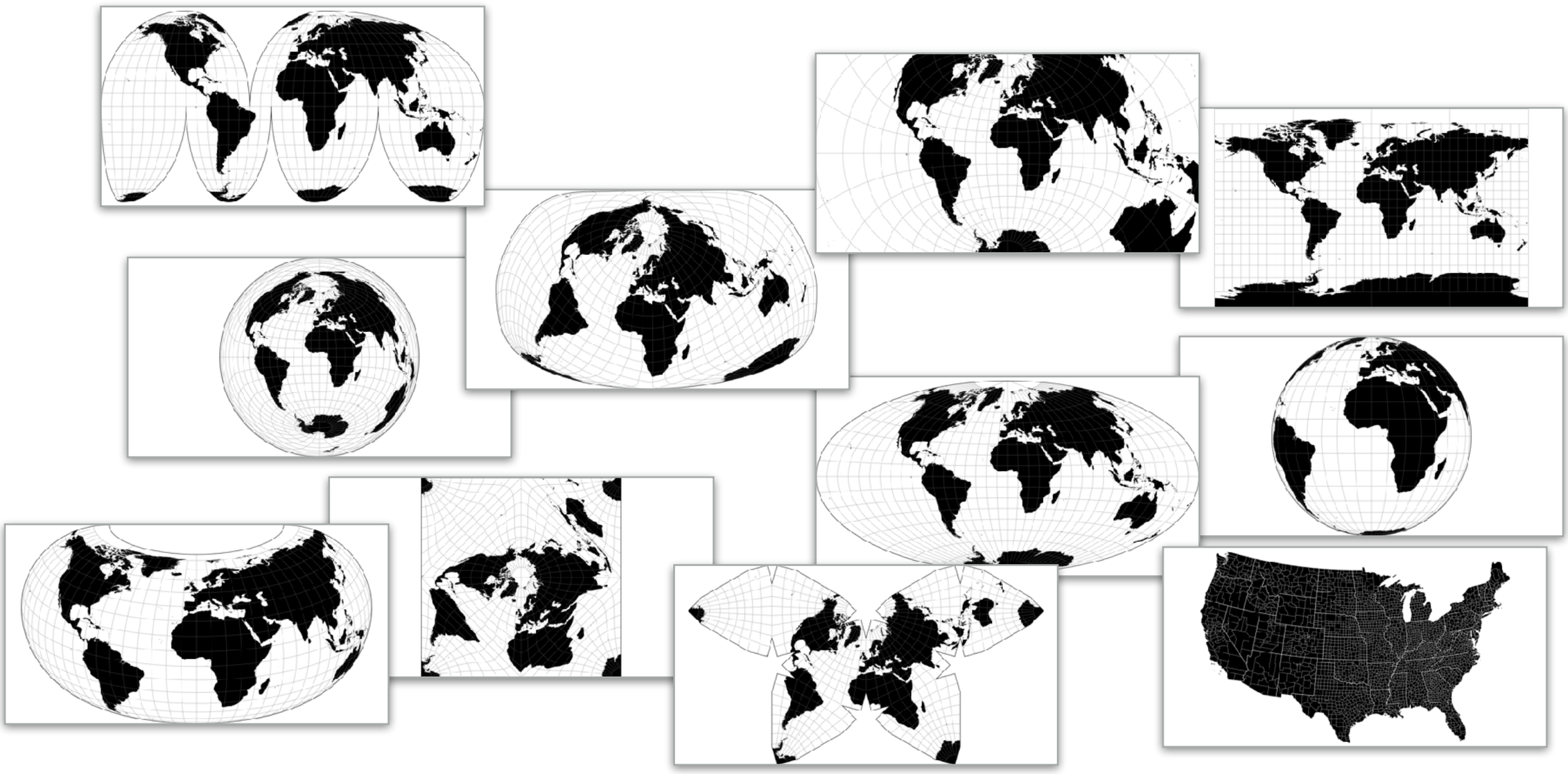
- Change map projection by setting the projection of the **`geoPath()`** generator:

```
let myProj = d3.geoEquirectangular().scale(MAP_HEIGHT / Math.PI);
let geoPathGen = d3.geoPath()
                   .projection(myProj);
```

d3-geo

https://github.com/d3/d3-geo
https://github.com/d3/d3-geo-projection

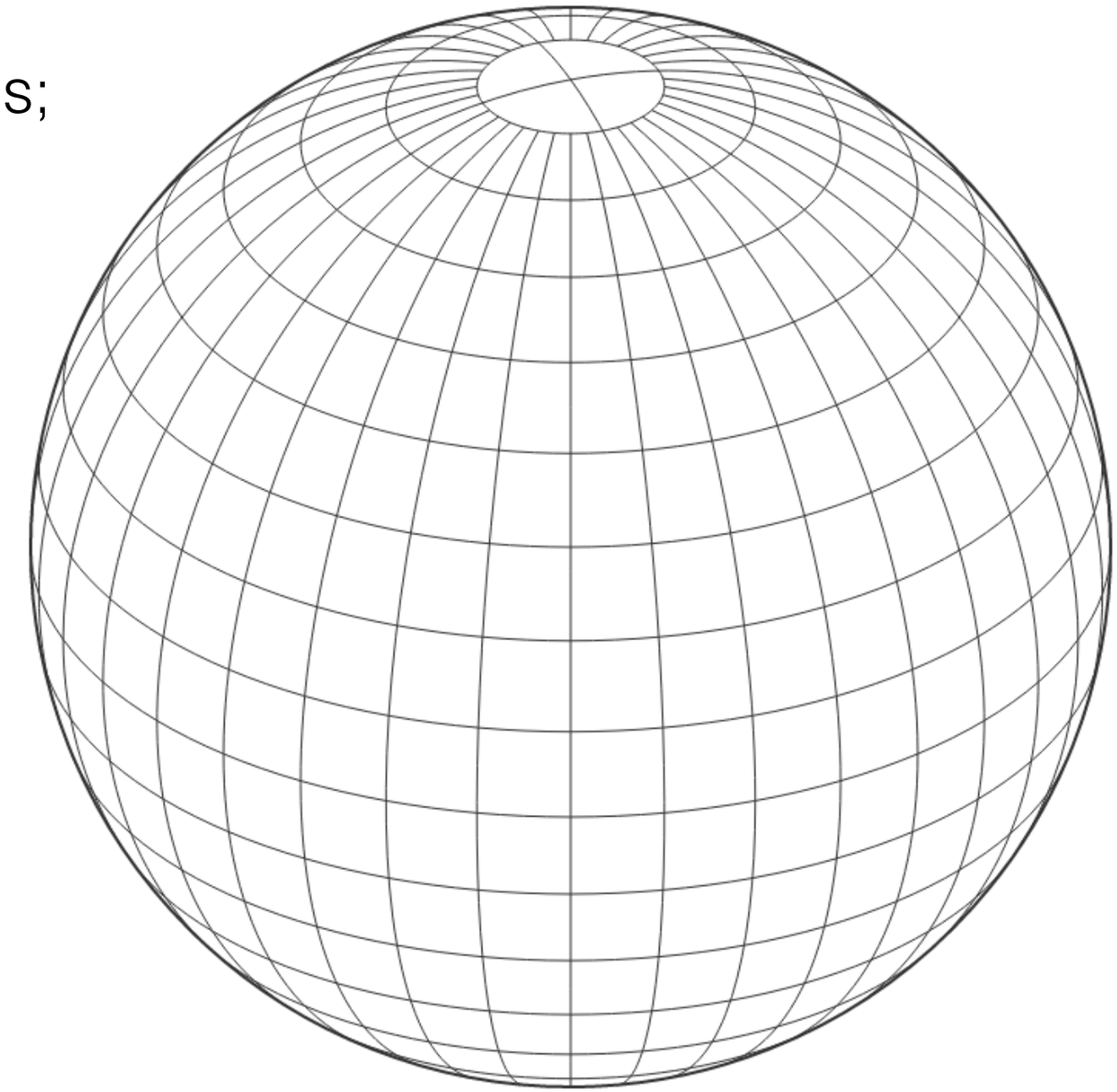
Numerous projections available, highly customizable:



d3-geo

Graticule generator:

- displays meridians and parallels;
- customizable;
- helps understand distortion.



https://github.com/d3/d3-geo#geoGraticule

d3-geo - Shapefile conversion, TopoJSON

On-the-fly conversion Shapefile -> GeoJSON:

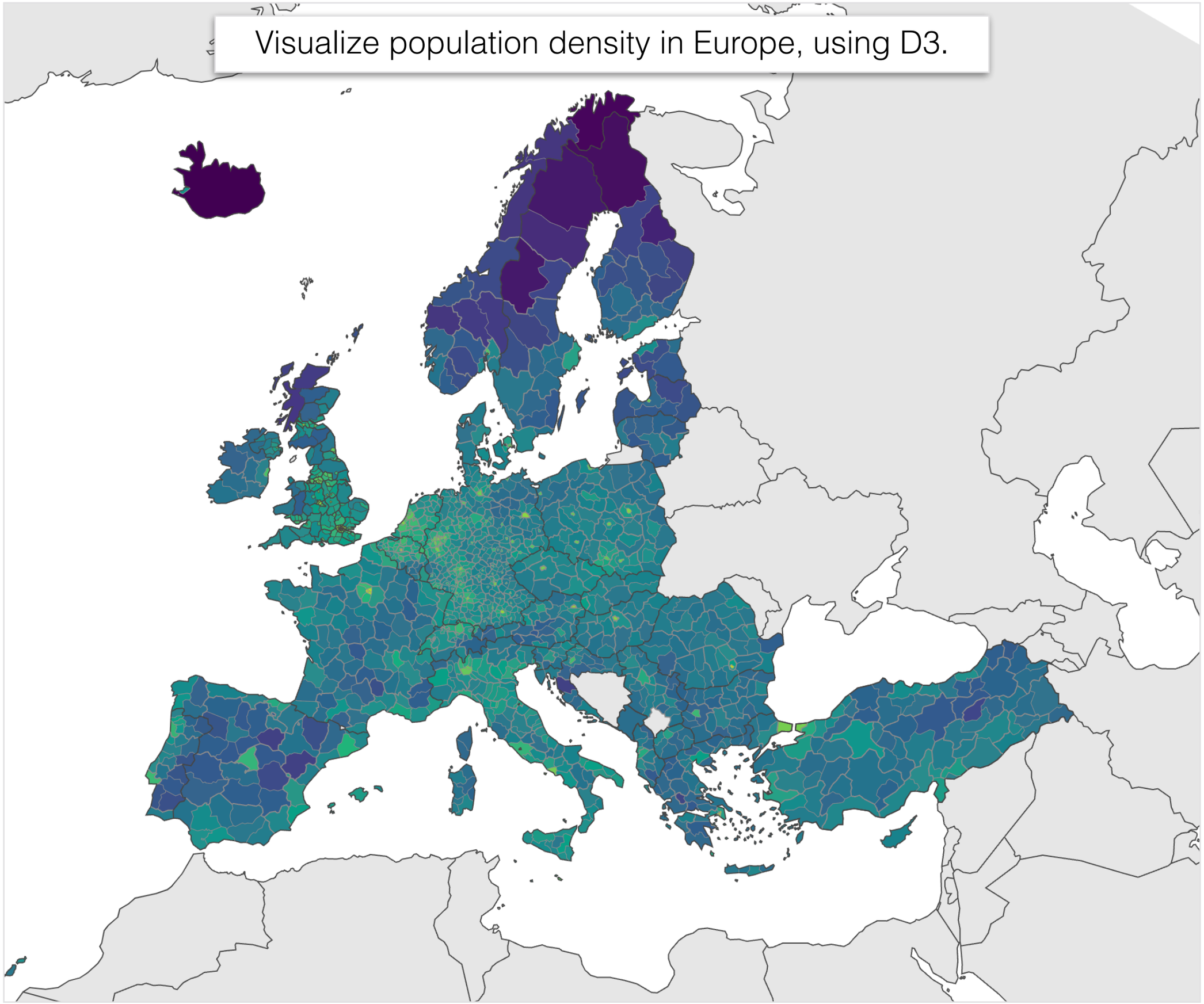
```
shapefile.open("https://cdn.rawgit.com/mbostock/shapefile/master/test/points.shp")
  .then(source => source.read()
    .then(function log(result) {
      if (result.done) return;
      console.log(result.value);
      return source.read().then(log);
    }))
  .catch(error => console.error(error.stack));
```

https://github.com/mbostock/shapefile

TopoJSON: encodes topology, enabling the stitching of geometries, thus eliminating redundancies, reducing file size.

https://github.com/topojson/topojson

PC s06-1



PC s06-1

- The input data is split in six files:

File	Description
`gra.geojson`	graticule
`nutsrg.geojson`	NUTS3 areas
`nutsbn.geojson`	NUTS3 borders
`cntrg.geojson`	country areas (outside EU)
`cntbn.geojson`	country borders (outside EU)
`pop_density_nuts3.csv`	population density, per NUTS3 region, per year

- Load all files, using Javascript function **`Promise.all()`** to handle the async' calls:

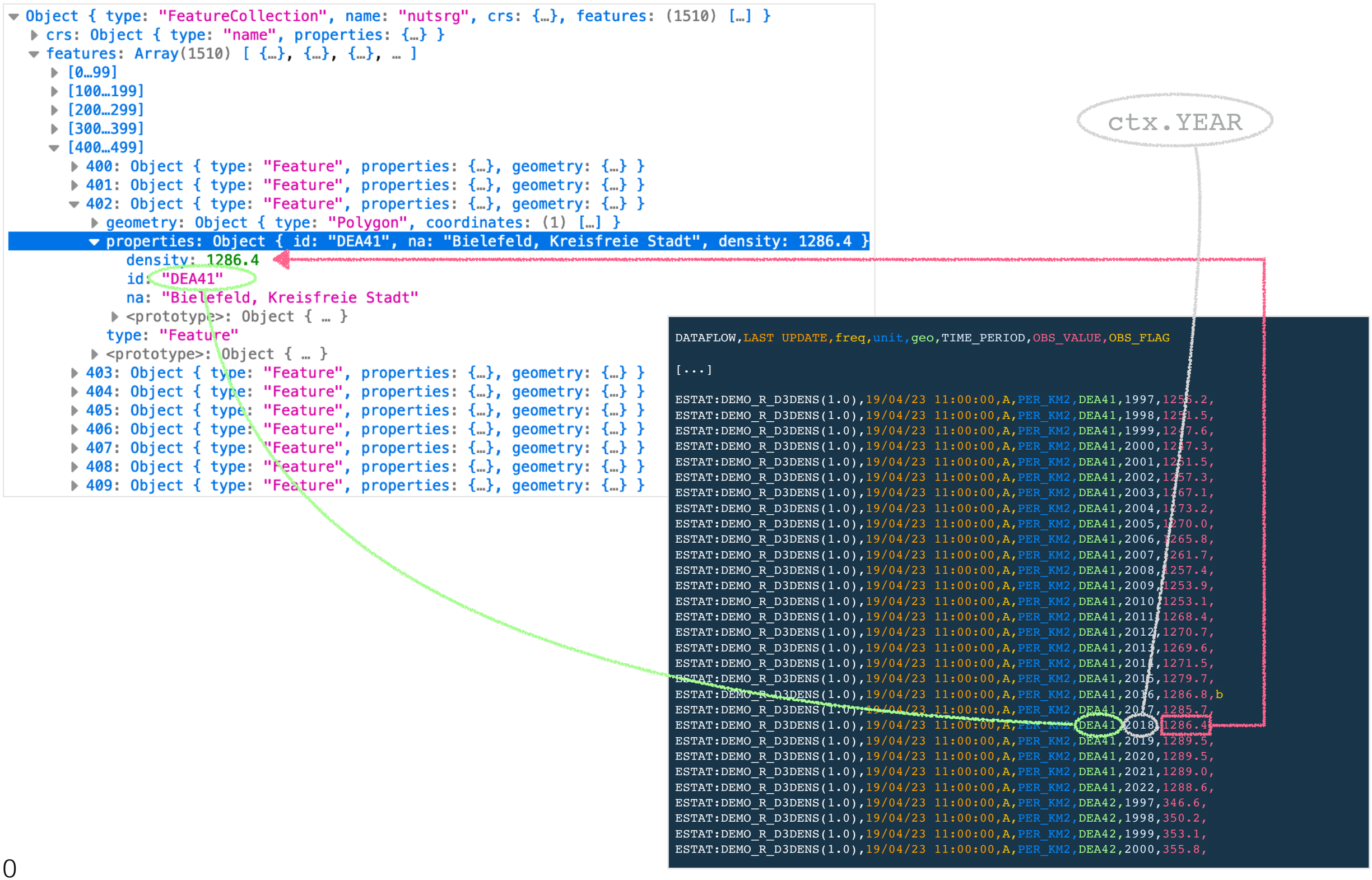
```
d3.json(...) // like d3.csv(), returns a Promise.
             // We called then(...) directly on that promise in previous D3 exercises.

// Here we want to load multiple resources before proceeding to the creation
// of the visualization. To wait for multiple promises to be completed, use:
Promise.all(...).then(function(data){/* Callback code */})

// This runs all promises given as input (array) to all(),
// and executes the code in then() only once
// all promises have been completed
// (in our case, fetching and parsing all CSV and JSON files)
```

https://github.com/d3/d3-fetch

https://developer.mozilla.org/docs/Web/JavaScript/Reference/Global_Objects/Promise/all

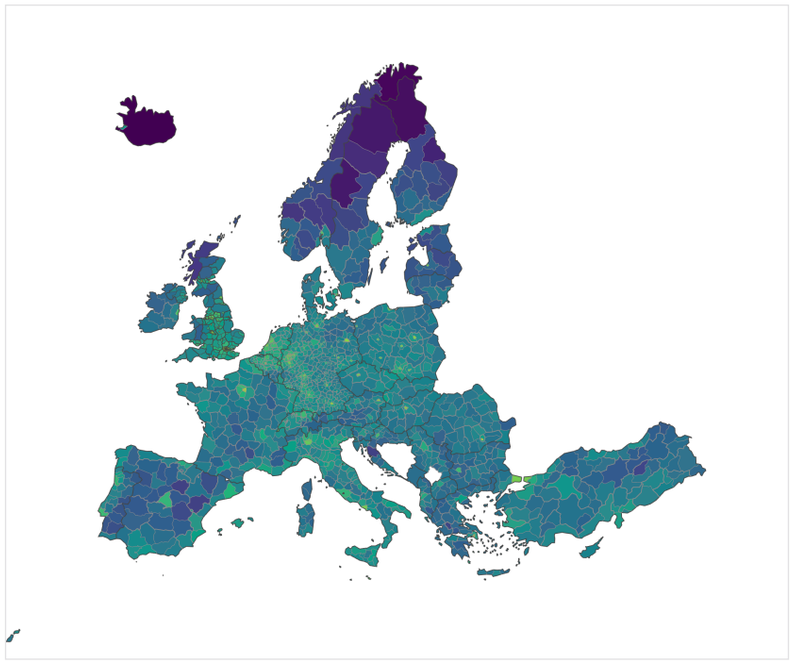
PC s06-1

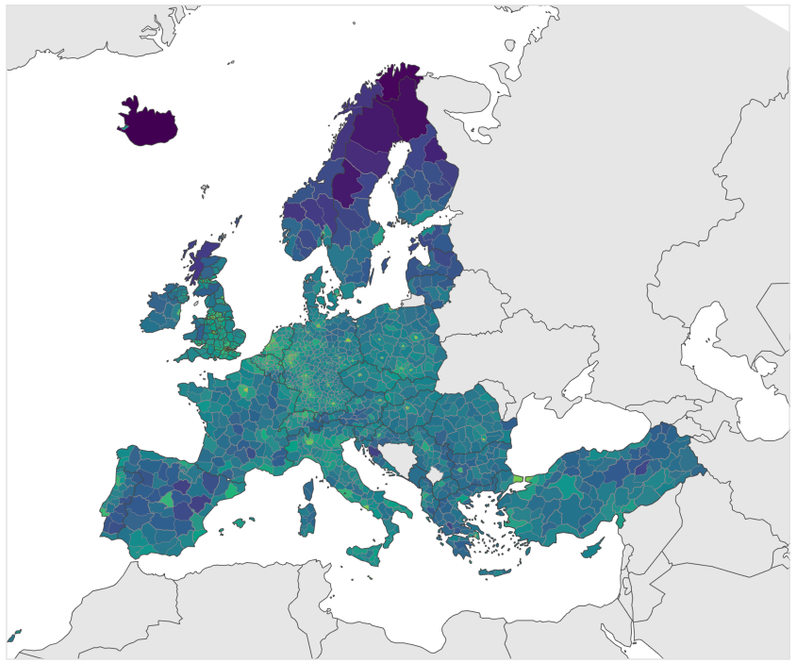
- Data structure



PC s06-1

- Map drawing





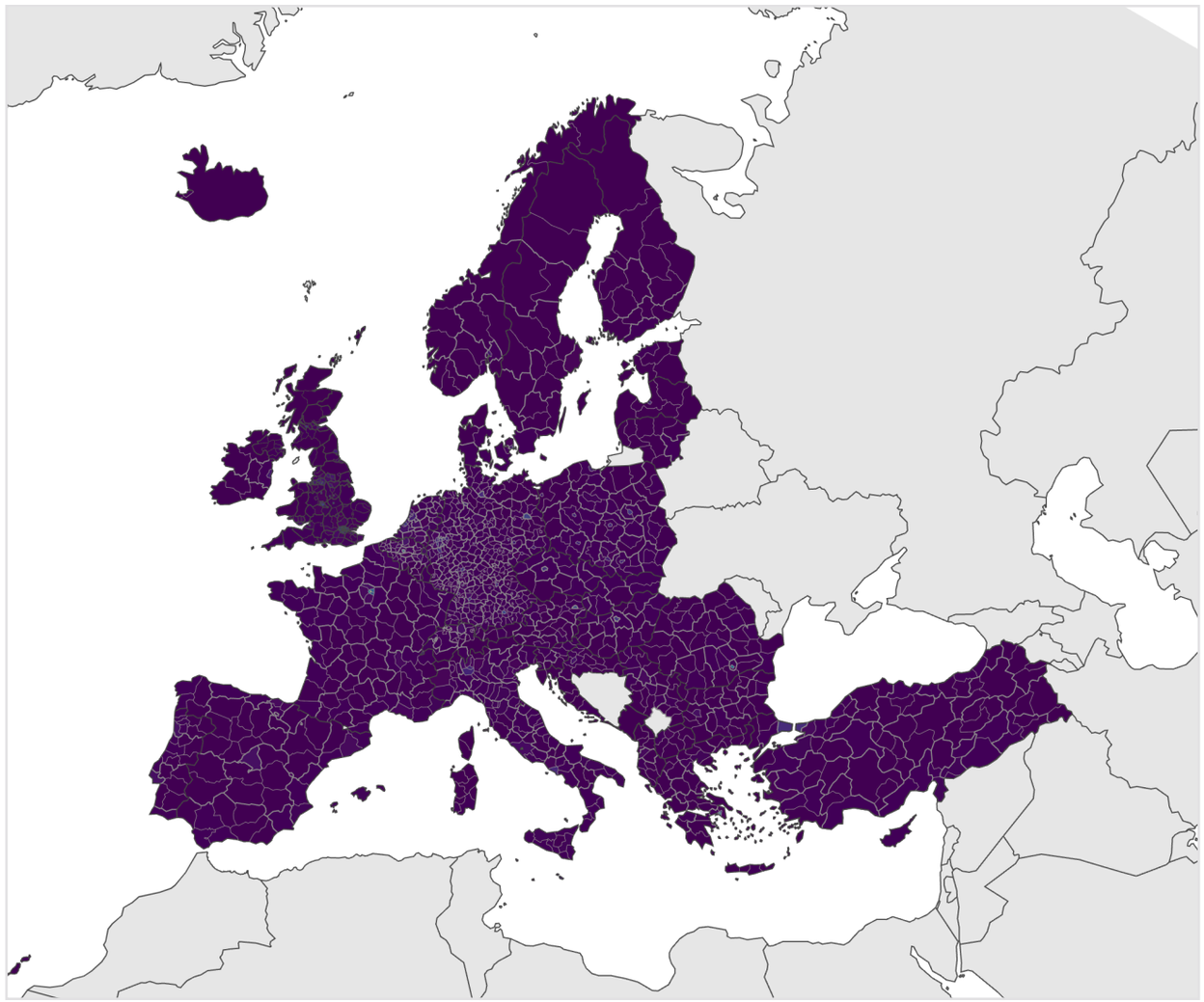
- Log-transformed density values mapped to the Viridis scale

https://d3js.org/d3-scale-chromatic/sequential#interpolateViridis

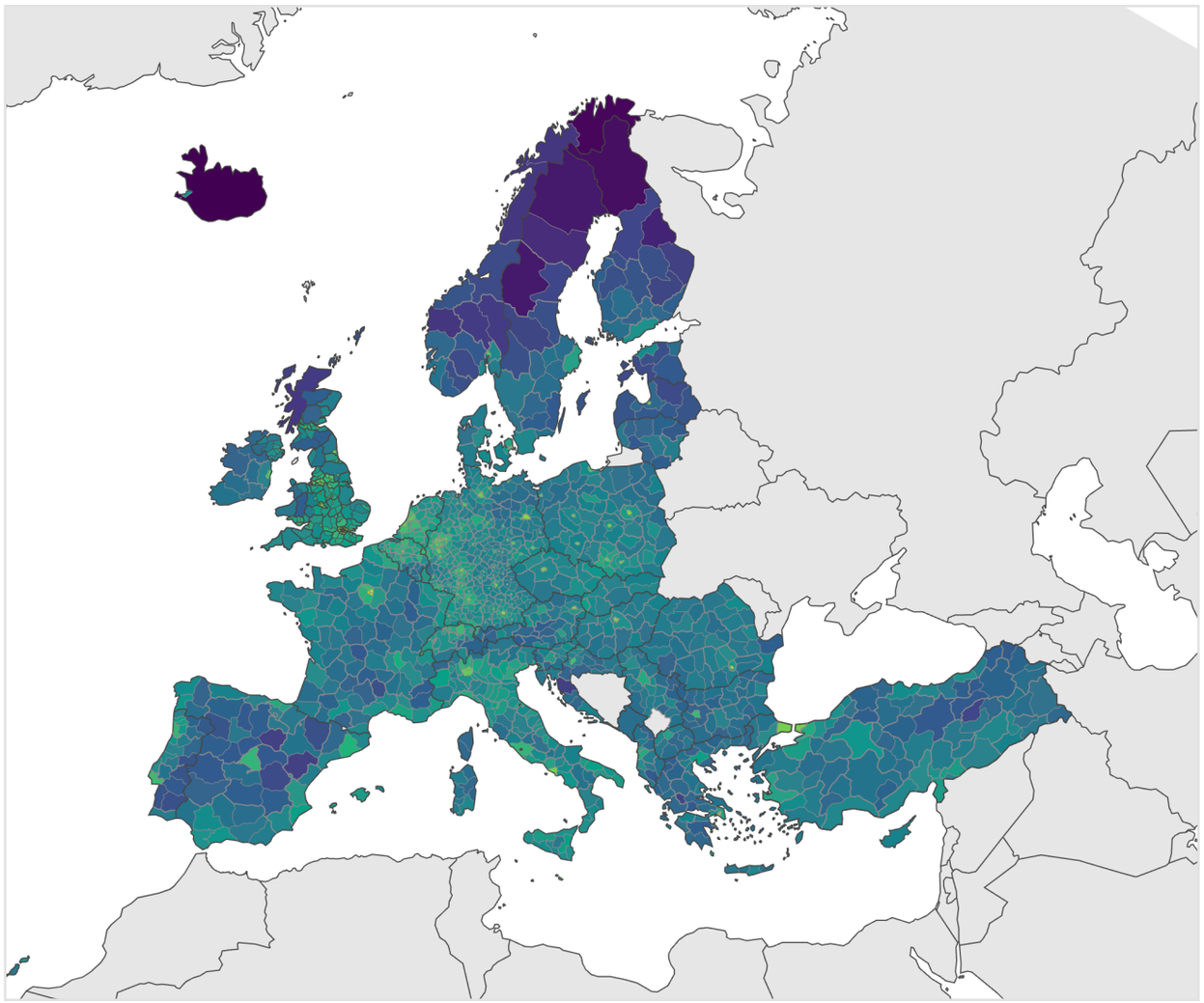
interpolateViridis(t)

Source · Given a number *t* in the range [0,1], returns the corresponding color from the "viridis" perceptually-uniform color scheme designed by van der Walt, Smith and Firing for matplotlib, represented as an RGB string.

PC s06-1



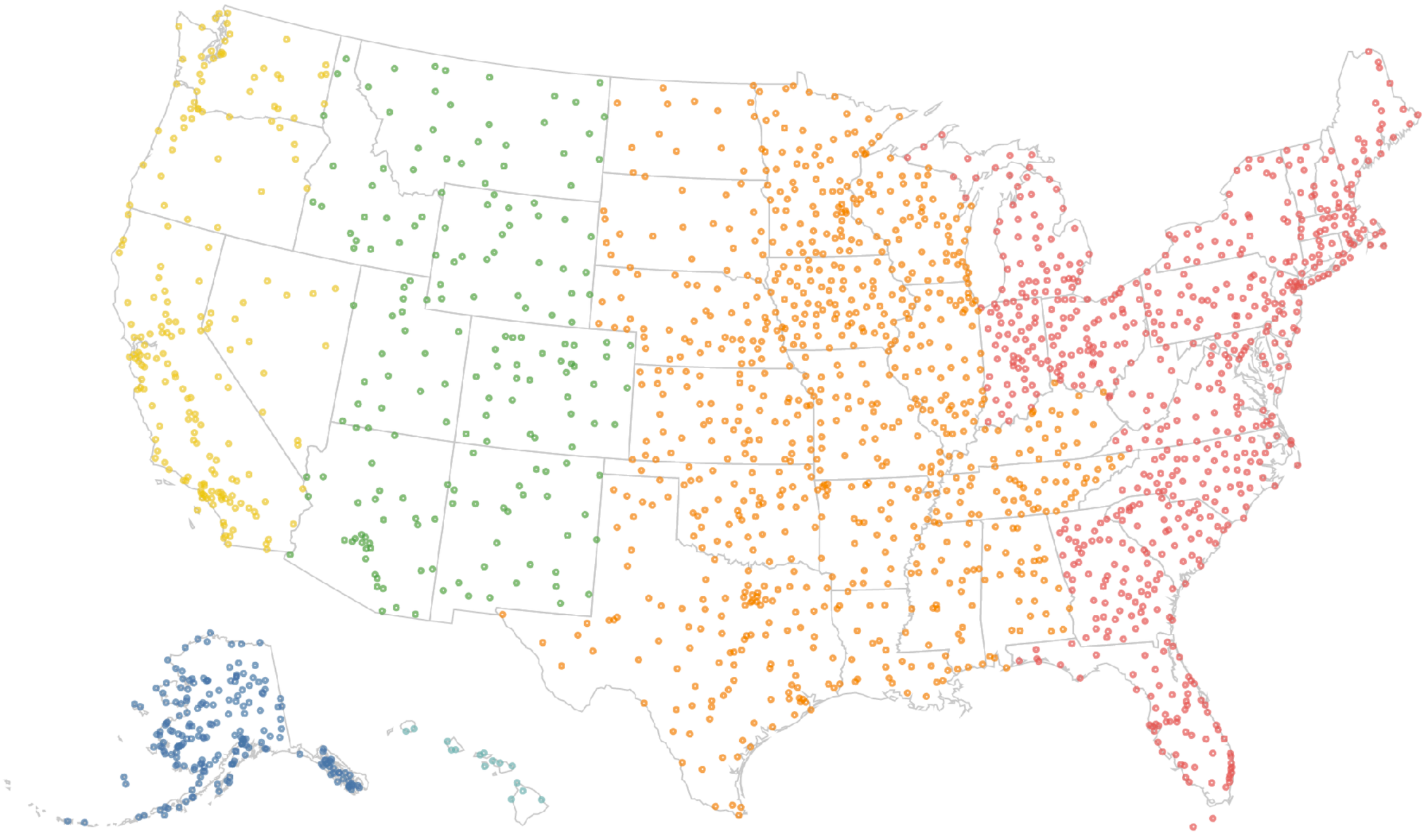
linear



log-transformed

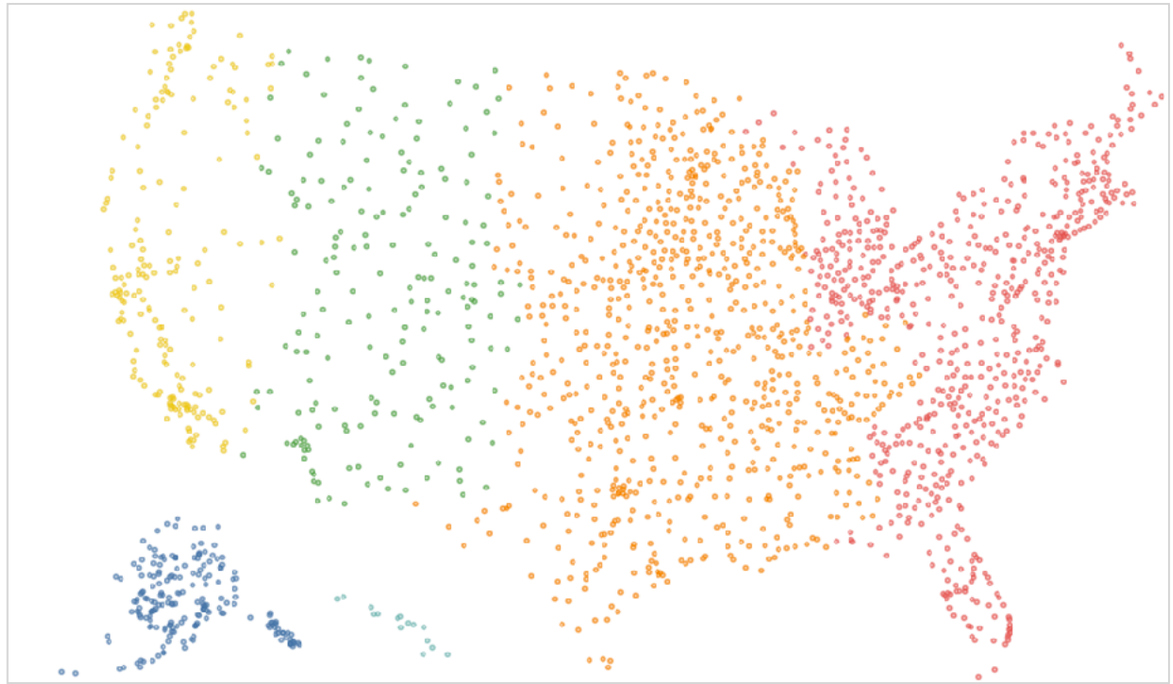
PC s06-2 (optional)

- Vega-lite: chart airports in the USA, color-coded by time zone:



PC s06-2 (optional)

- The visualization will be composed of two superimposed layers
  https://vega.github.io/vega-lite/docs/layer.html
  (we already did this for the line plot in s#03)



PC s06-2 (optional)

- Base map (bottom layer):
  - Take inspiration from https://vega.github.io/vega-lite/examples/geo_layer.html to draw the states' borders from GeoJSON file **`us-10m.json`** using Albers projection.
  - and adapt the shapes' **`fill`** and **`stroke`** colors to match this:

PC s06-2 (optional)

- Airports (top layer):
  - The same example `https://vega.github.io/vega-lite/examples/geo_layer.html` shows how to plot the second layer.
  - Use **`point`** marks instead of **`circle`** marks.

PC s06-2 (optional)

airports.json


```
[{"city": "Bay Springs", "country": "USA", "iata": "00M", "latitude": 31.95376472,
"longitude": -89.23450472, "name": "Thigpen", "state": "MS"},
 {"city": "Livingston", "country": "USA", "iata": "00R", "latitude": 30.68586111,
"longitude": -95.01792778, "name": "Livingston Municipal", "state": "TX"},
 {"city": "Colorado Springs", "country": "USA", "iata": "00V", "latitude":
38.94574889, "longitude": -104.5698933, "name": "Meadow Lake", "state": "CO"},
 {"city": "Perry", "country": "USA", "iata": "01G", "latitude": 42.74134667,
"longitude": -78.05208056, "name": "Perry-Warsaw", "state": "NY"},
 {"city": "Hilliard", "country": "USA", "iata": "01J", "latitude": 30.6880125,
"longitude": -81.90594389, "name": "Hilliard Airpark", "state": "FL"},
 {"city": "Belmont", "country": "USA", "iata": "01M", "latitude": 34.49166667,
"longitude": -88.20111111, "name": "Tishomingo County", "state": "MS"},
 {"city": "Clanton", "country": "USA", "iata": "02A", "latitude": 32.85048667,
"longitude": -86.61145333, "name": "Gragg-Wade", "state": "AL"},
 {"city": "Brookfield", "country": "USA", "iata": "02C", "latitude": 43.08751,
"longitude": -88.17786917, "name": "Capitol", "state": "WI"},
 {"city": "East Liverpool", "country": "USA", "iata": "02G", "latitude": 40.67331278,
"longitude": -80.64140639, "name": "Columbiana County", "state": "OH"},
 {"city": "Memphis", "country": "USA", "iata": "03D", "latitude": 40.44725889,
"longitude": -92.22696056, "name": "Memphis Memorial", "state": "MO"},
 {"city": "Pittsboro", "country": "USA", "iata": "04M", "latitude": 33.93011222,
"longitude": -89.34285194, "name": "Calhoun County", "state": "MS"},
 {"city": "Hawley", "country": "USA", "iata": "04Y", "latitude": 46.88384889,
"longitude": -96.35089861, "name": "Hawley Municipal", "state": "MN"},
 {"city": "Griffith", "country": "USA", "iata": "05C", "latitude": 41.51961917,
"longitude": -87.40109333, "name": "Griffith-Merrillville", "state": "IN"},
 {"city": "Gatesville", "country": "USA", "iata": "05F", "latitude": 31.42127556,
"longitude": -97.79696778, "name": "Gatesville - City/County", "state": "TX"},
 {"city": "Eureka", "country": "USA", "iata": "05U", "latitude": 39.60416667,
"longitude": -116.0050597, "name": "Eureka", "state": "NV"},
 {"city": "Tuskegee", "country": "USA", "iata": "06A", "latitude": 32.46047167,
"longitude": -85.68003611, "name": "Moton  Municipal", "state": "AL"},
```


states_tz.csv


```
State,TimeZone
AL,CST
AK,AKST
AZ,MST
AR,CST
CA,PST
CO,MST
CT,EST
DE,EST
FL,EST
GA,EST
HI,HST
ID,MST
IL,CST
IN,EST
IA,CST
KS,CST
KY,CST
LA,CST
ME,EST
MD,EST
MA,EST
MI,EST
MN,CST
MS,CST
MO,CST
MT,MST
NE,CST
NV,PST
NH,EST
NJ,EST
NM,MST
```

PC s06-2 (optional)

- Airports (top layer):
  - Color code airports based on the time zone of the parent state:
    - For each airport, lookup the time zone in **`states_tz.csv`** and add it as a new attribute of that airport, using a Vega-Lite **`transform`**
      Take inspiration from example at `https://vega.github.io/vega-lite/docs/lookup.html`
    - Then encode that **`nominal`** attribute using **`color`**

PC s06-2 (optional)

- Airports (top layer):
  - Filter out airports with numbers in their 3-letter IATA code (as we did in s#04)
    Hint: regular expressions `/[0-9]/` or `/\\d/` will return `true` if any of the 3 chars is a number

```
https://vega.github.io/vega-lite/docs/filter.html
https://vega.github.io/vega/docs/expressions/#regexp-functions
```