INF552 (2023-2024) - PC s06

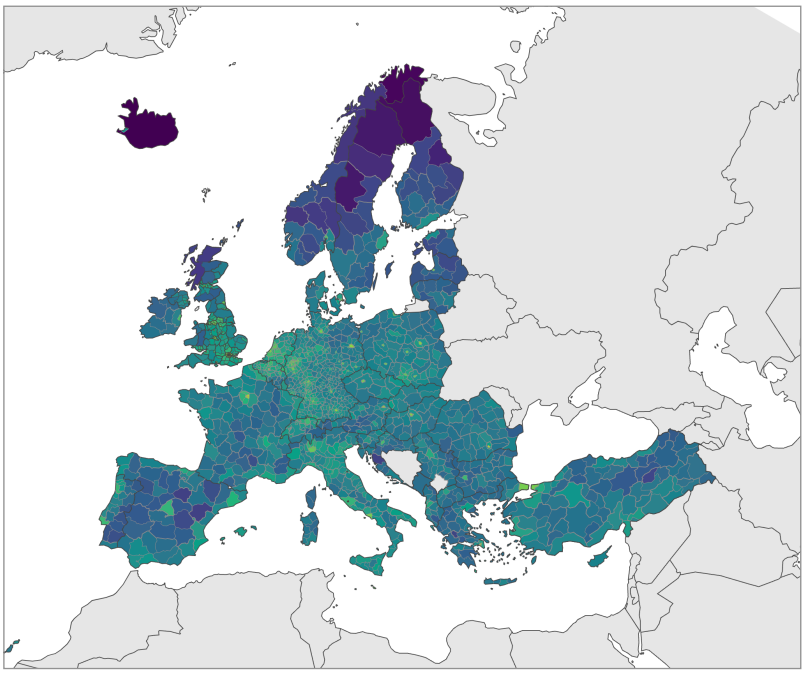
1. Population Density in Europe

Goal: visualize population density, by NUTS region, using D3.

NUTS = *Nomenclature of territorial units for statistics* https://ec.europa.eu/eurostat/web/nuts/background

We will:

- draw countries in the European union, rendering level-3 administrative divisions (NUTS3);
- color those NUTS3 regions to show population density;
- draw countries outside the European union to give geographical context.



1.1. Data Structure

In function `loadData()`, fetch and parse the following 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

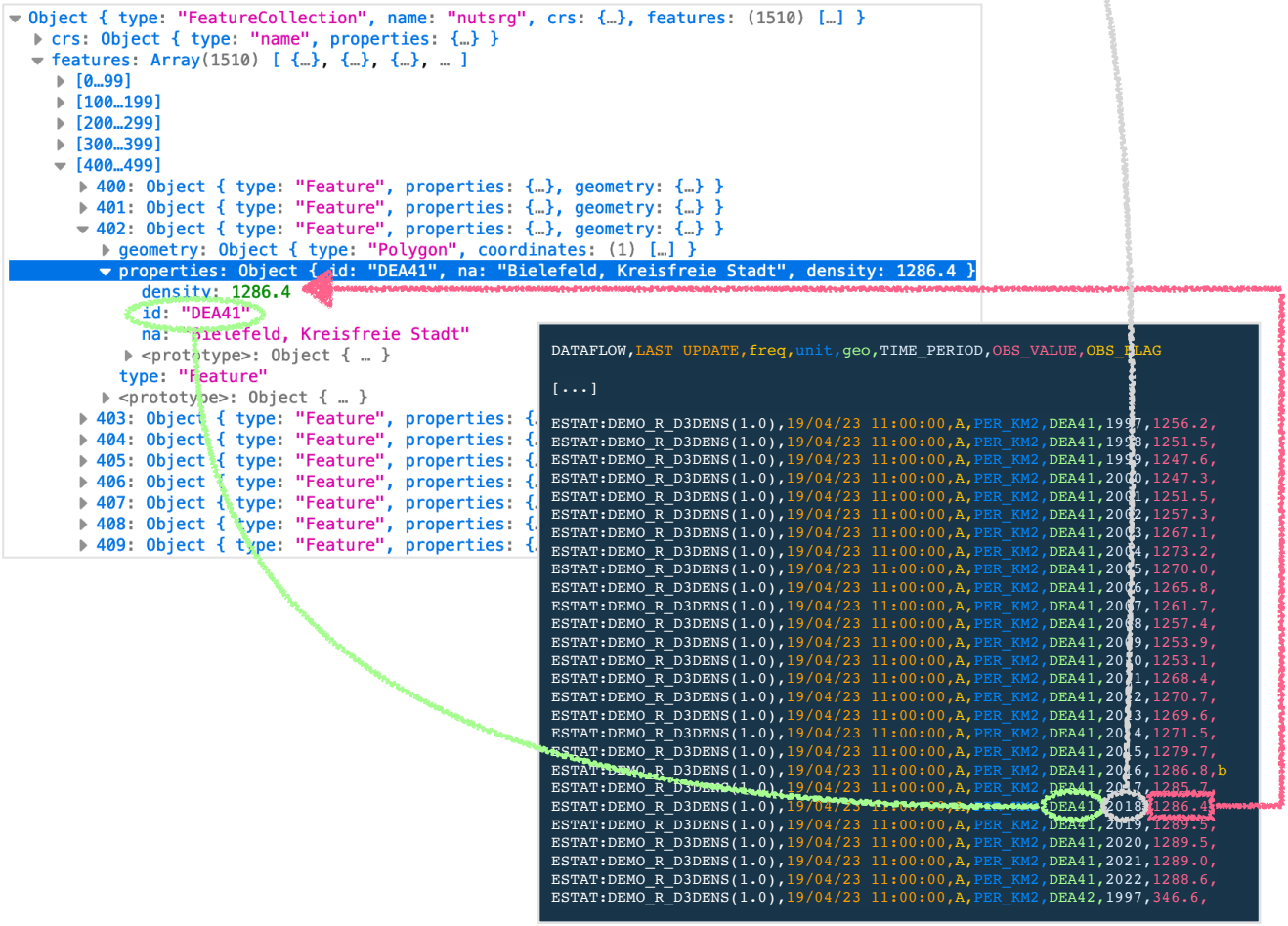
using `Promise.all()` to handle the multiple asynchronous calls to `d3.json()` and `d3.csv()` - see details on the next page.

```
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)
```

Further information:
https://github.com/d3/d3-fetch
https://developer.mozilla.org/docs/Web/JavaScript/Reference/Global_Objects/Promise/all

For each feature in `nutsrg`, retrieve the population density `OBS_VALUE` from `pop_density_nuts3.csv` as follows:

- find the rows corresponding to that NUTS3 region (matching `id` ⟷ `geo`);
- within this subset of rows find the only one corresponding to the year set in `ctx.YEAR` ;
- then add the corresponding OBS_VALUE as an additional item in that NUTS3 regions's properties (call it, *e.g.*, `density`).



1.2. Putting NUTS3 regions on the map

Now that the data structure has been transformed, we are going to draw the map, step by step.

Initialize the projection for Europe as follows (this is just one easy way to do it):

```
ctx.proj = d3.geoIdentity()
             .reflectY(true)
             .fitSize([ctx.SVG_H, ctx.SVG_H], graticule);
// graticule is the data structure parsed from gra.geojson
```

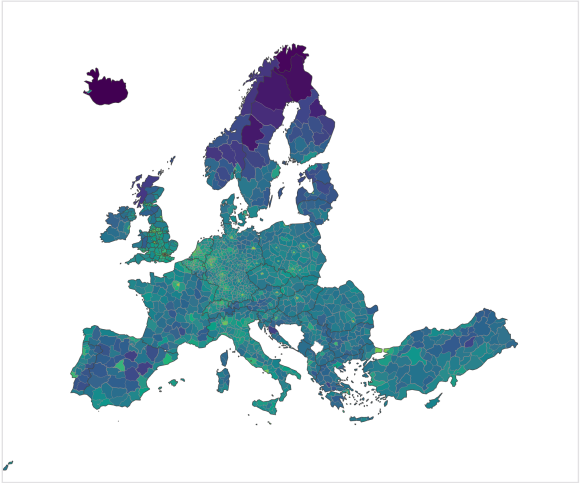
- Create a D3 geo-path generator using that projection, (see slides #3-4 of INF552-2023-PC-slides-s06.pdf)
- Use that geo-path generator to draw areas from the feature array already loaded from `nutsrg.geojson`. Each feature corresponds to one SVG `<path>`, and is thus bound to it using D3's data binding mechanism, as in previous exercises. Put all these `<path>` elements into the same `<g>`. Set `nutsArea` as their class (it is already defined in CSS).
- Do the same for borders, using the same geo-path generator, from the features in `nutsbn.geojson`, setting `nutsBorder` as their class.

1.3. Coloring NUTS3 regions according to population density

Create a color scale as follows:

- log-transform the density data using `d3.scaleLog()` (just set the domain, no need to specify a range);
- create a sequential color scale using `d3.scaleSequential()` that maps density values to colors:
  - using the VIRIDIS interpolator;
  - fed with log-transformed values.

https://d3js.org/d3-scale-chromatic

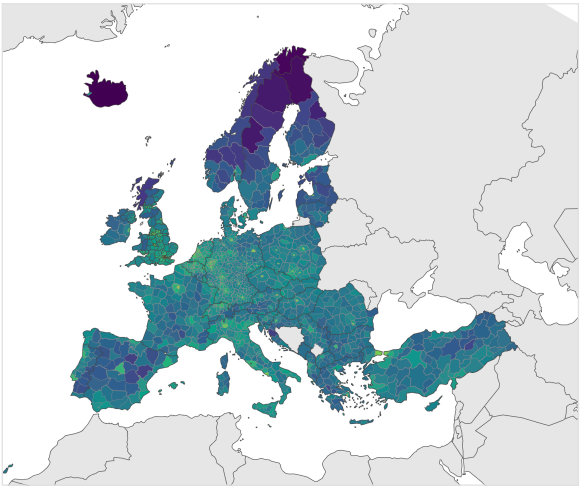


Then use that scale to set the fill color of individual NUTS3 regions based on their own density value, using D3 attribute mappings as we have done in many previous exercises. Since `<path>` elements are already bound to the NUTS3 features, this is very straightforward.

1.4. Putting other countries on the map

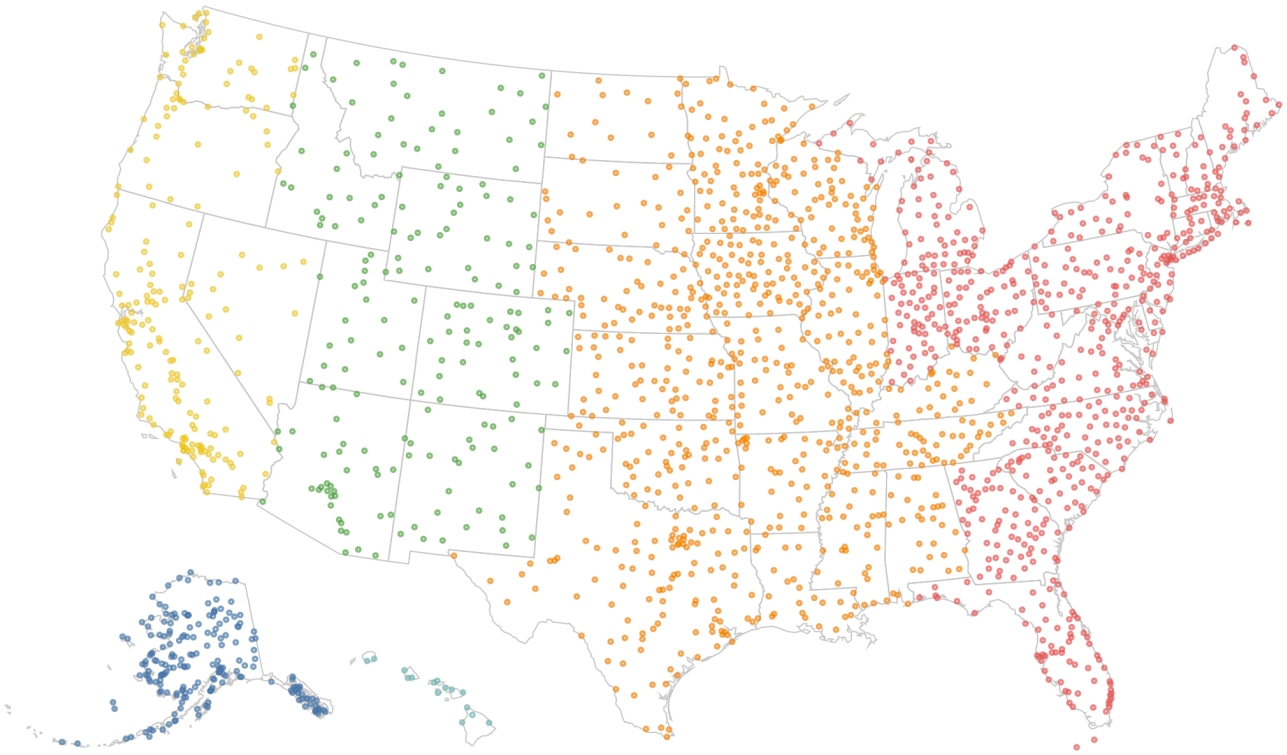
Use the same geo-path generator as in Section 1.2 to draw areas and borders from `cntrg.geojson` and `cntbn.geojson`.

Set classes `countryArea` and `countryBorder` on them, respectively; and put them in two separate `<g>` elements.

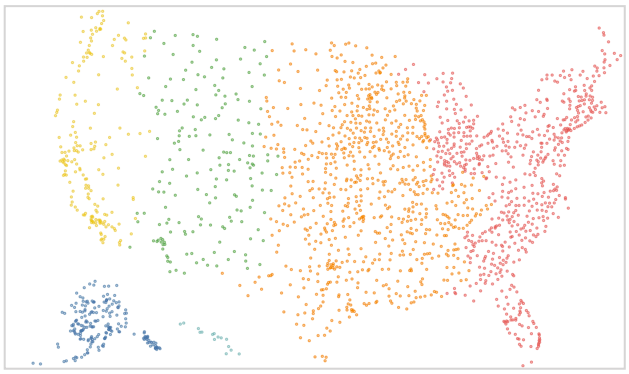


2. Plotting Airports in the USA (optional)

Goal: chart airports in the USA, color-coded by time zone, using Vega-Lite.



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



2.1. 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.

Adapt the shapes' `fill` and `stroke` colors to match the above illustration.

2.2. Airports (top layer)

The same above-referenced example shows how to plot a second layer. Use `point` marks instead of `circle` marks.

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

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"},
```


states_tz.csv


```
State,TimeZone
…
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
```


- then encode that `nominal` attribute using `color` as the encoding channel;

- finally, filter out airports with numbers in their 3-letter IATA code.

Tip: regular expressions `/[0-9]/` or `/\\d/` will return `true` if any of the 3 chars is a number. Create a `filter` which uses the `test(…)` regexp function, accessing the data value with keyword `datum`.
https://vega.github.io/vega-lite/docs/filter.html
https://vega.github.io/vega/docs/expressions/#regexp-functions