an open-source JavaScript library
for mobile-friendly interactive maps

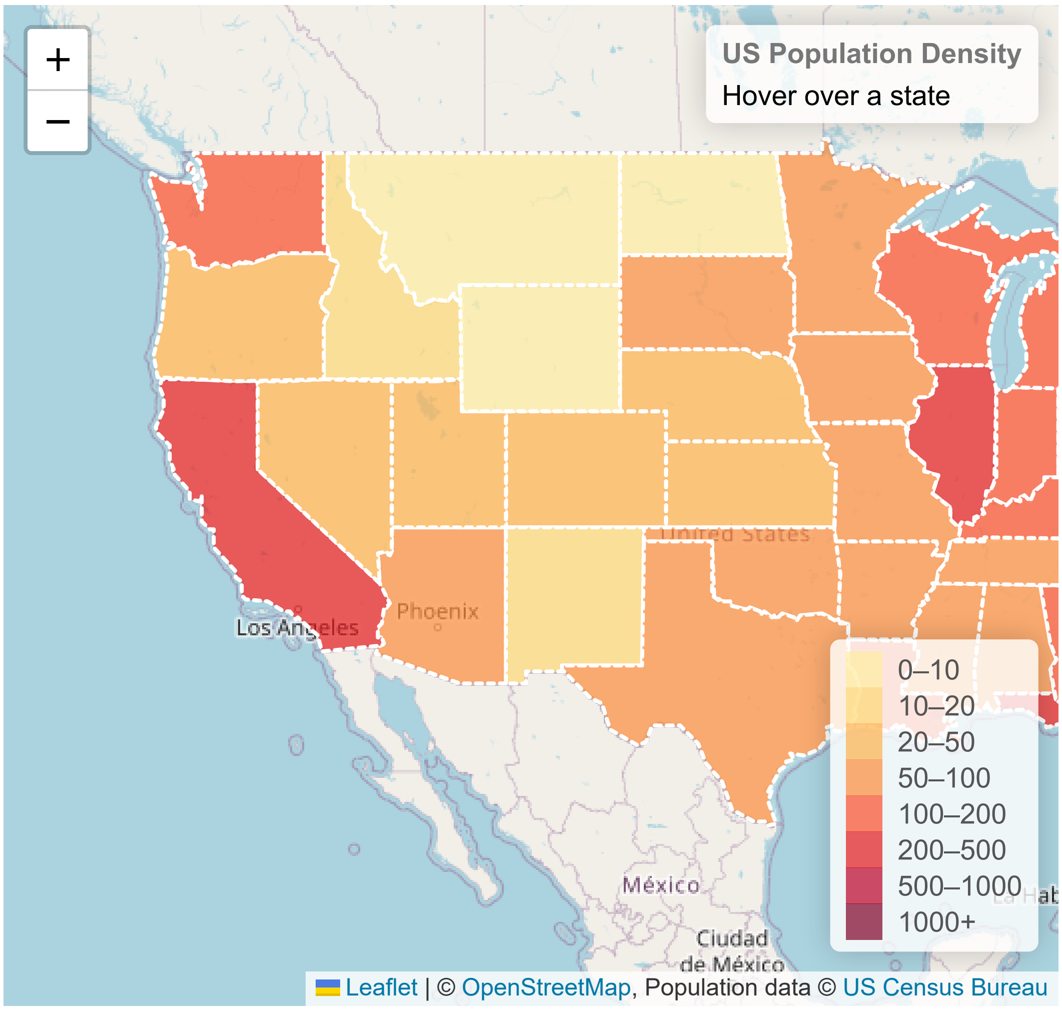
Overview Tutorials Docs Download Plugins Blog

← Tutorials

Interactive Choropleth Map

This is a case study of creating a colorful interactive choropleth map of US States Population Density with the help of GeoJSON and some custom controls (that will hopefully convince all the remaining major news and government websites that do not use Leaflet yet to start doing so).

The tutorial was inspired by the Texas Tribune US Senate Runoff Results map (also powered by Leaflet), created by Ryan Murphy.



See this example stand-alone.

Data Source

We'll be creating a visualization of population density per US state. As the amount of data (state shapes and the density value for each state) is not very big, the most convenient and simple way to store and then display it is GeoJSON.

Each feature of our GeoJSON data (us-states.js) will look like this:

```
{
    "type": "Feature",
    "properties": {
        "name": "Alabama",
        "density": 94.65
    },
    "geometry": ...
    ...
}
```

The GeoJSON with state shapes was kindly shared by Mike Bostock of D3 fame, extended with density values from this Wikipedia article based on July 1st 2011 data from US Census Bureau and assigned to `statesData` JS variable.

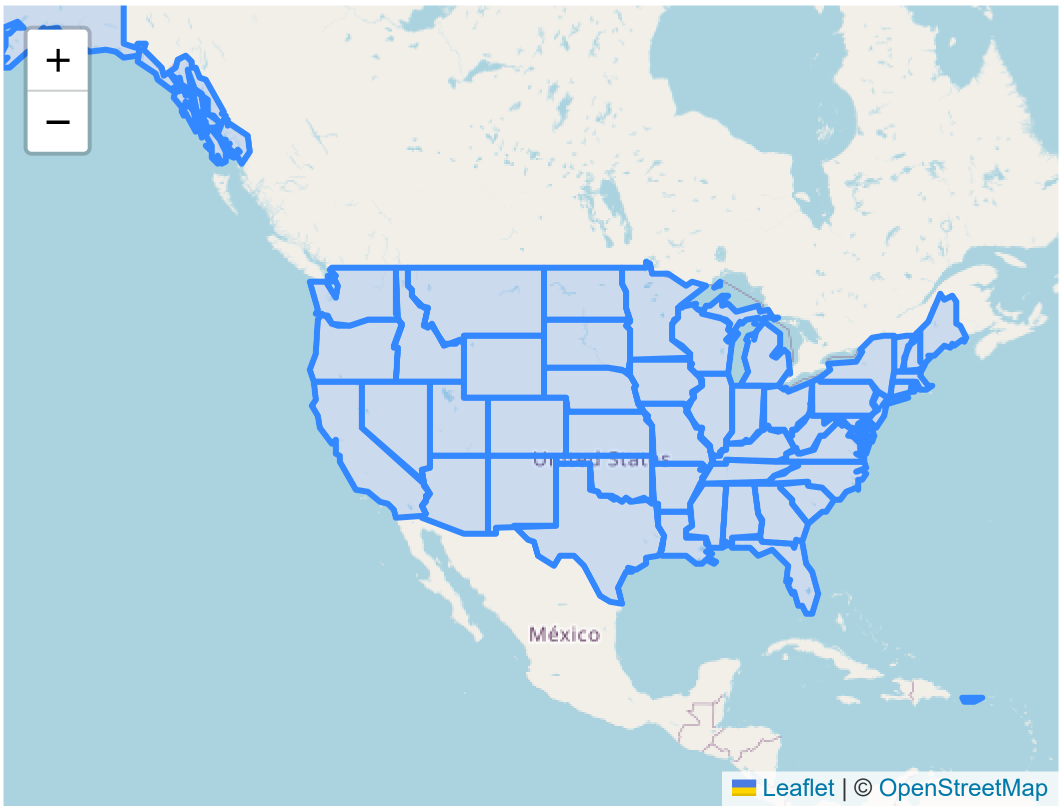
Basic States Map

Let's display our states data on the map:

```
var map = L.map('map').setView([37.8, -96], 4);

var tiles = L.tileLayer('https://tile.openstreetmap.org/{z}/{x}/{y}
    maxZoom: 19,
    attribution: '&copy; <a href="http://www.openstreetmap.org/copy
}).addTo(map);

L.geoJson(statesData).addTo(map);
```



See this example stand-alone.

Adding Some Color

Now we need to color the states according to their population density. Choosing nice colors for a map can be tricky, but there's a great tool that can help with it — ColorBrewer. Using the values we got from it, we create a function that returns a color based on population density:

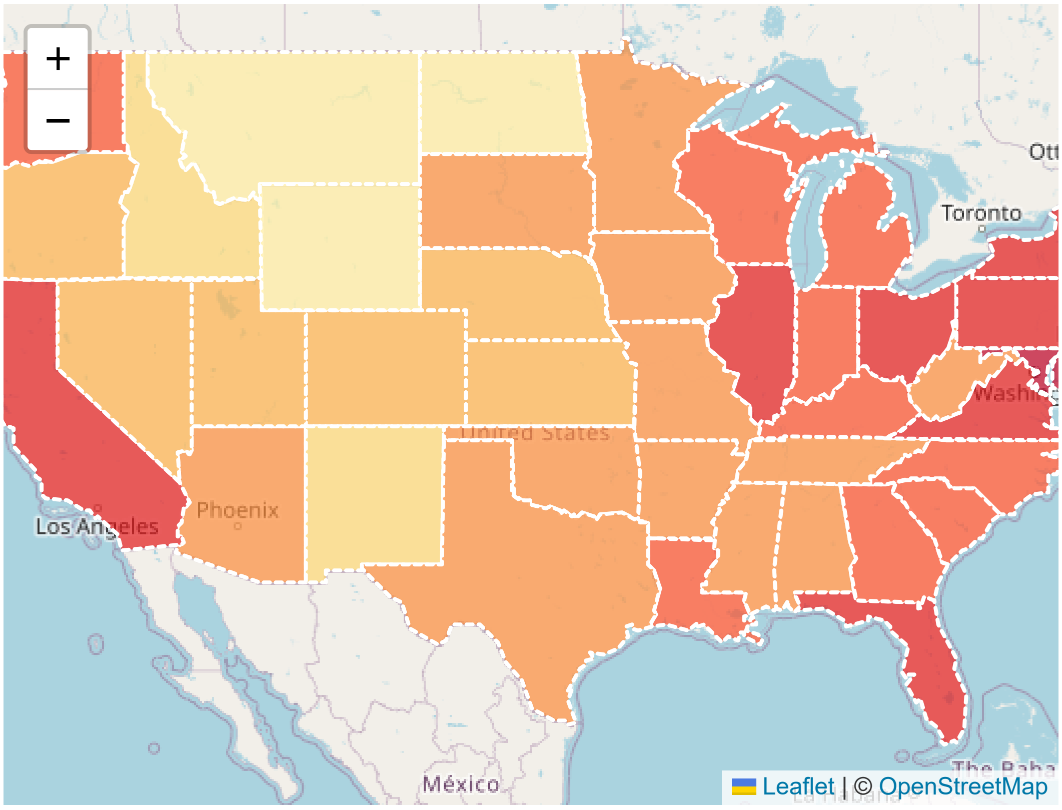
```
function getColor(d) {
    return d > 1000 ? '#800026' :
           d > 500  ? '#BD0026' :
           d > 200  ? '#E31A1C' :
           d > 100  ? '#FC4E2A' :
           d > 50   ? '#FD8D3C' :
           d > 20   ? '#FEB24C' :
           d > 10   ? '#FED976' :
                      '#FFEDA0';
}
```

Next we define a styling function for our GeoJSON layer so that its `fillColor` depends on `feature.properties.density` property, also adjusting the appearance a bit and adding a nice touch with dashed stroke.

```
function style(feature) {
    return {
        fillColor: getColor(feature.properties.density),
        weight: 2,
        opacity: 1,
        color: 'white',
        dashArray: '3',
        fillOpacity: 0.7
    };
}

L.geoJson(statesData, {style: style}).addTo(map);
```

Looks much better now!



See this example stand-alone.

Adding Interaction

Now let’s make the states highlighted visually in some way when they are hovered with a mouse. First we’ll define an event listener for layer `mouseover` event:

```
function highlightFeature(e) {
    var layer = e.target;

    layer.setStyle({
        weight: 5,
        color: '#666',
        dashArray: '',
        fillOpacity: 0.7
    });

    layer.bringToFront();
}
```

Here we get access to the layer that was hovered through `e.target`, set a thick grey border on the layer as our highlight effect, also bringing it to the front so that the border doesn’t clash with nearby states.

Next we'll define what happens on `mouseout`:

```
function resetHighlight(e) {
    geojson.resetStyle(e.target);
}
```

The handy `geojson.resetStyle` method will reset the layer style to its default state (defined by our `style` function). For this to work, make sure our GeoJSON layer is accessible through the `geojson` variable by defining it before our listeners and assigning the layer to it later:

```
var geojson;
// ... our listeners
geojson = L.geoJson(...);
```

As an additional touch, let's define a `click` listener that zooms to the state:

```
function zoomToFeature(e) {
    map.fitBounds(e.target.getBounds());
}
```

Now we'll use the `onEachFeature` option to add the listeners on our state layers:

```
function onEachFeature(feature, layer) {
    layer.on({
        mouseover: highlightFeature,
        mouseout: resetHighlight,
        click: zoomToFeature
    });
}

geojson = L.geoJson(statesData, {
    style: style,
    onEachFeature: onEachFeature
}).addTo(map);
```

This makes the states highlight nicely on hover and gives us the ability to add other interactions inside our listeners.

Custom Info Control

We could use the usual popups on click to show information about different states, but we'll choose a different route — showing it on state hover inside a custom control.

Here's the code for our control:

```
var info = L.control();

info.onAdd = function (map) {
    this._div = L.DomUtil.create('div', 'info'); // create a div wi
    this.update();
    return this._div;
};

// method that we will use to update the control based on feature p
info.update = function (props) {
    this._div.innerHTML = '<h4>US Population Density</h4>' +  (prop
        '<b>' + props.name + '</b><br />' + props.density + ' peopl
        : 'Hover over a state');
};

info.addTo(map);
```

We need to update the control when the user hovers over a state, so we'll also modify our listeners as follows:

```
function highlightFeature(e) {
    ...
    info.update(layer.feature.properties);
}

function resetHighlight(e) {
    ...
    info.update();
}
```

The control also needs some CSS styles to look nice:

```
.info {
    padding: 6px 8px;
    font: 14px/16px Arial, Helvetica, sans-serif;
    background: white;
    background: rgba(255,255,255,0.8);
    box-shadow: 0 0 15px rgba(0,0,0,0.2);
    border-radius: 5px;
}
.info h4 {
    margin: 0 0 5px;
    color: #777;
}
```

Custom Legend Control

Creating a control with a legend is easier, since it is static and doesn’t change on state hover. JavaScript code:

```
var legend = L.control({position: 'bottomright'});

legend.onAdd = function (map) {

    var div = L.DomUtil.create('div', 'info legend'),
        grades = [0, 10, 20, 50, 100, 200, 500, 1000],
        labels = [];

    // loop through our density intervals and generate a label with
    for (var i = 0; i < grades.length; i++) {
        div.innerHTML +=
            '<i style="background:' + getColor(grades[i] + 1) + '">
            grades[i] + (grades[i + 1] ? '&ndash;' + grades[i + 1]
    }

    return div;
};

legend.addTo(map);
```

CSS styles for the control (we also reuse the `info` class defined earlier):

```
.legend {
    line-height: 18px;
    color: #555;
}
.legend i {
    width: 18px;
    height: 18px;
```

```
        float: left;
        margin-right: 8px;
        opacity: 0.7;
    }
```

Enjoy the result on the top of this page, or on a separate page.