# Mejor opción para crear un mapa de calor por CCAA en España

Para representar un **mapa de calor (mapa coroplético)** de España segmentado por Comunidades Autónomas (CCAA) con un degradado de color según un valor numérico (por ejemplo, volumen de normativa), existen varias librerías gratuitas que podemos considerar. A continuación analizaremos las principales opciones compatibles con tu *stack* (Node.js, MongoDB, frontend en JavaScript/HTML/CSS sin framework) y elegiremos la más adecuada. Finalmente, se presenta una guía paso a paso de implementación.

## Opciones de librerías gratuitas para mapas coropléticos

**1. D3.js (con TopoJSON/GeoJSON):** D3 es una librería de visualización muy potente y flexible, que permite crear mapas vectoriales personalizados usando estándares web (HTML, SVG, etc.)[[1]](https://mappinggis.com/2025/04/las-mejores-apis-javascript-para-webmapping/#:~:text=D3JS%20es%20una%20librer%C3%ADa%20JavaScript,un%20mapa%20sencillo%20con%20D3). En combinación con datos geográficos (GeoJSON o TopoJSON), D3 puede generar mapas de cualquier región y aplicar escalas de color a los datos. *Pros:* máxima flexibilidad y personalización; permite integrar fácilmente datos en diversos formatos[[2]](https://mappinggis.com/2025/04/las-mejores-apis-javascript-para-webmapping/#:~:text=Image%3A%20D3%20librer%C3%ADa%20JavaScript%20para,crear%20mapas%20web). *Contras:* requiere mayor esfuerzo de codificación y conocimiento de SVG/proyecciones; no ofrece componentes de mapa listos, por lo que hay que implementar manualmente interacción (hover, tooltips, leyendas, etc.). D3 es excelente para visualizaciones a medida, pero **su curva de aprendizaje es alta** y podría ser más de lo necesario si buscas rapidez de implementación.

**2. Leaflet:** Leaflet es una de las librerías de mapas web más populares y ligeras (≈42 KB)[[3]](https://mappinggis.com/2025/04/las-mejores-apis-javascript-para-webmapping/#:~:text=Leaflet%20es%20tambi%C3%A9n%20una%20librer%C3%ADa,la%20simplicidad%2C%20rendimiento%20y%20usabilidad). Se centra en funciones básicas de mapas, priorizando la **simplicidad, rendimiento y usabilidad**[[3]](https://mappinggis.com/2025/04/las-mejores-apis-javascript-para-webmapping/#:~:text=Leaflet%20es%20tambi%C3%A9n%20una%20librer%C3%ADa,la%20simplicidad%2C%20rendimiento%20y%20usabilidad). Leaflet permite cargar datos GeoJSON de forma nativa y estilizar regiones según sus propiedades, lo que encaja perfectamente con un mapa coroplético. Es conocido por ser **fácil de aprender incluso para desarrolladores nóveles**, con documentación clara, muchos ejemplos, y una gran variedad de *plugins* para ampliar funcionalidades[[4]](https://mappinggis.com/2025/04/las-mejores-apis-javascript-para-webmapping/#:~:text=,Mejor%20soporte%20para%20m%C3%B3viles). *Pros:* código abierto (BSD), ampliamente usado en entornos enterprise; integración sencilla en páginas HTML con JavaScript puro; soporta interacción (eventos de mouse, popups) de forma intuitiva. *Contras:* a nivel de mapas vectoriales avanzados puede requerir plugins adicionales; dependerá de obtener los datos geográficos (no incluye mapas de España por defecto). En general, Leaflet ofrece un **equilibrio ideal entre buenas prácticas en mapas web y rapidez de implementación**, siendo adecuado para un SaaS enterprise que necesite mapas interactivos sin complejidad excesiva.

**3. Apache ECharts (opción de gráfico de mapa):** ECharts es una biblioteca de gráficos *open-source* (Apache 2.0) que soporta visualizaciones interactivas variadas, incluidos mapas coropléticos. Con ECharts se puede registrar un mapa de España (aportando el GeoJSON de las CCAA) y luego simplemente proporcionar los datos de cada región. Ofrece componentes integrados como visualMap para generar escalas de color continuas o discretas automáticamente y tooltip para mostrar valores al pasar el cursor. *Pros:* alto nivel de abstracción – la configuración define el mapa y los valores y la librería se encarga del renderizado, escalas de color y leyendas; incluye *tooltips* interactivos de serie[[5]](https://echarts.apache.org/en/download-map.html#:~:text=To%20create%20Map%20charts%2C%20it,Please); puede reutilizarse para gráficos futuros (barras, líneas, etc.) en la misma plataforma. *Contras:* curva de aprendizaje moderada en cuanto a estructura de opciones; necesitas conseguir el geojson de las CCAA y *registrarlo* en la librería. Aun así, es una opción muy potente y orientada a resultados rápidos. Muchos desarrolladores la usan para dashboards empresariales por su facilidad de configuración y resultados atractivos.

**4. Plotly.js:** Plotly es otra librería de visualización de datos interactiva (MIT) que soporta mapas coropléticos. Similar a ECharts, Plotly permite crear mapas coloreados proporcionando un GeoJSON y un conjunto de valores, manejando automáticamente la escala de colores y las etiquetas emergentes. *Pros:* alto nivel (poca codificación manual); gráficas interactivas con zoom/hover listos; integraciones con Python/R (irrelevante para tu stack, pero indica versatilidad). *Contras:* tamaño de la librería relativamente grande; documentación algo dispersa en el caso de uso JS puro; necesita incluir la definición del mapa (GeoJSON). En aplicaciones web puras se usa menos que Leaflet para mapas, pero sigue siendo viable.

**5. Otras opciones:** *OpenLayers* (muy completa pero más pesada y compleja que Leaflet), *Mapbox GL/MapLibre* (excelentes mapas con WebGL, pero requieren más configuración y un token si se usan servicios Mapbox), *Google Maps API* o *Google Geochart* (permiten mapas coropléticos de regiones, pero implican dependencias de Google y posibles costes[[6]](https://mappinggis.com/2025/04/las-mejores-apis-javascript-para-webmapping/#:~:text=Google%20ha%20conseguido%20democratizar%20el,servicio%20de%20pago%20por%20uso), por lo que no son la primera opción si buscamos solo soluciones libres). También existen librerías de mapas vectoriales simples como **JVectorMap/jQVMap** que traen mapas SVG predefinidos (a veces incluyen España por provincias); son fáciles de usar pero menos mantenidas y con menos flexibilidad que Leaflet o ECharts.

**Conclusión de opciones:** Dado tu criterio de *best practices* para un SaaS enterprise **y** rapidez en la implementación, la balanza se inclina hacia una solución que sea **madura, gratuita y mínima en código manual**. **Leaflet** destaca en este equilibrio: es open-source, ampliamente probado en producción (gobiernos, news media, empresas[[7]](https://leafletjs.com/examples/choropleth/#:~:text=This%20is%20a%20case%20study,yet%20to%20start%20doing%20so)) y muy rápido de implementar gracias a sus facilidades para choropleths. Alternativamente, **ECharts** también cumple con rapidez (poca codificación) y es usado en entornos enterprise para dashboards interactivos – sería una excelente elección si quisieras aprovechar su sistema de gráficos para otras visualizaciones en tu app. Sin embargo, dado que tu necesidad principal es un mapa geográfico interactivo, usaremos **Leaflet** en la guía a continuación (por su simplicidad de integración con vanilla JS). Mencionaremos en la guía cómo lograr todos los requisitos (degradado personalizable, tooltip al hover, actualización dinámica con filtros) usando Leaflet.

*(Nota:* Cualquiera de estas librerías requiere datos geoespaciales de las CCAA. Afortunadamente, existen fuentes abiertas: por ejemplo, OpendataSoft ofrece un dataset GeoJSON de las Comunidades Autónomas de España con geometrías simplificadas e identificadores ISO[[8]](https://public.opendatasoft.com/explore/dataset/georef-spain-comunidad-autonoma/?flg=en-us#:~:text=Autonomous%20communities%20)[[9]](https://public.opendatasoft.com/explore/dataset/georef-spain-comunidad-autonoma/?flg=en-us#:~:text=Enhancements). También el Instituto Geográfico Nacional y datos.gob.es publican shapefiles de límites administrativos. En esta guía asumiremos que dispones del GeoJSON de las CCAA o que puedes obtenerlo fácilmente de dichas fuentes y almacenarlo en tu app).\*

## Guía de implementación paso a paso con Leaflet

A continuación, se detalla cómo integrar Leaflet en tu aplicación Node/vanilla JS para mostrar un mapa coroplético de España por CCAA. Incluiremos los pasos para configurar el mapa, cargar datos dummy, aplicar un degradado de color según valores y agregar interacción de **hover** con un tooltip tipo *“caja negra”* mostrando el *score*. Finalmente, veremos cómo actualizar los datos dinámicamente al aplicar filtros.

**Paso 1: Instalar e incluir Leaflet**. Puedes instalar Leaflet via npm (npm install leaflet) o usar la CDN directamente en tu HTML. Si usas Vanilla JS sin *build tools*, la forma más rápida es añadir en la página HTML:<br/>html <link rel="stylesheet" href="https://unpkg.com/leaflet/dist/leaflet.css" /> <script src="https://unpkg.com/leaflet/dist/leaflet.js"></script><br/>Esto carga la librería Leaflet y sus estilos de mapa. Luego, en el cuerpo de tu HTML crea un contenedor para el mapa, por ejemplo: <div id="map" style="height: 600px;"></div> (asegurarse de darle altura/ancho vía CSS).

**Paso 2: Obtener el GeoJSON de Comunidades Autónomas**. Si ya dispones de los datos geoespaciales de las 17 CCAA, puedes incluirlos en tu app. Por ejemplo, podrías guardarlo en un archivo .js o .json e importarlo. Leaflet soporta cargar GeoJSON directamente con L.geoJson(geojsonData). Otra opción es hacer una petición AJAX para obtenerlo desde tu servidor (Node puede servir un .json). **Nota:** El GeoJSON debe incluir una propiedad identificadora de la comunidad (por nombre o código) para vincular con tus datos. Muchas fuentes usan properties.name con el nombre oficial de la comunidad. Por ejemplo, cada Feature del GeoJSON podría verse así: { "type": "Feature", "properties": { "name": "Andalucía" }, "geometry": { ... } }. Asegúrate de tener esa referencia (nombre/código) que también tendrás en tus datos dummy.

**Paso 3: Preparar datos de volumen (dummy)**. En tu frontend, define un objeto o arreglo que contenga el valor de “volumen normativo” para cada CCAA. Por ejemplo, algo así:

const datosCCAA = {  
 "Andalucía": 120, "Aragón": 80, "Asturias": 95, /\* ... resto de CCAA ... \*/ "Ceuta": 30, "Melilla": 28  
};

Esto asocia cada comunidad con un número (aquí inventado). Si tus datos reales vienen de MongoDB vía Node, la idea sería que el backend te envíe un JSON ya con esos valores por región. Para la visualización, usaremos este objeto para determinar los colores.

**Paso 4: Inicializar el mapa de Leaflet centrado en España**. En un script JS, una vez cargada la página, inicializa el mapa:

const map = L.map('map').setView([40, -3], 6); // Coordenadas aprox. centro de España, zoom 6

Por defecto el mapa Leaflet requerirá una capa base. Puedes usar una capa gratuita de OpenStreetMap:

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {  
 maxZoom: 19, attribution: '&copy; OpenStreetMap'  
}).addTo(map);

*(La capa base es opcional: si prefieres un fondo liso, Leaflet también puede mostrar solo las capas vectoriales. Con un fondo de OSM, el usuario podría hacer zoom/pan y ver el contexto geográfico.)*

**Paso 5: Agregar la capa GeoJSON de las CCAA con estilo de color**. Utilizaremos L.geoJson para añadir las regiones al mapa y aplicaremos un **estilo** que rellene cada comunidad con un color dependiendo de su valor. Podemos definir una función de ayuda getColor(valor) que devuelva un color en una escala. Por simplicidad, haremos una escala de colores de claro a oscuro (por ejemplo, amarillo claro para valores bajos hasta rojo oscuro para valores altos). Puedes definir rangos fijos o calcular dinámicamente el mínimo y máximo de tus datos para interpolar colores. En este ejemplo usaremos rangos fijos estilo **ColorBrewer** (similar a la documentación oficial[[10]](https://leafletjs.com/examples/choropleth/#:~:text=function%20getColor%28d%29%20,FFEDA0)):

function getColor(d) {  
 return d > 100 ? '#800026' :  
 d > 80 ? '#BD0026' :  
 d > 60 ? '#E31A1C' :  
 d > 40 ? '#FC4E2A' :  
 d > 20 ? '#FD8D3C' :  
 d > 10 ? '#FEB24C' :  
 '#FED976';  
}

*(En este ejemplo, >100 sería color más oscuro y <10 el más claro; deberás ajustar los intervalos según los posibles valores de tus datos para lograr un buen contraste. Alternativamente, podrías usar una escala continua.*)

Ahora definimos la función de estilo para cada feature GeoJSON usando getColor sobre el valor de esa CCAA en datosCCAA:

function style(feature) {  
 const nombre = feature.properties.name; // asumiendo 'name' tiene nombre CCAA  
 const valor = datosCCAA[nombre] || 0;  
 return {  
 fillColor: getColor(valor),  
 weight: 2, color: 'white', opacity: 1, // bordes blancos  
 fillOpacity: 0.7  
 };  
}  
}

Esta función asigna un fillColor según el valor de la comunidad, y también ajusta algunos estilos: bordes blancos de grosor 2, opacidad del relleno 0.7 para ver algo de mapa base debajo, etc. Aplicaremos este estilo al cargar la capa GeoJSON:

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

Con esto, el mapa mostrará cada región autonómica coloreada según su volumen (de amarillo claro a rojo oscuro en nuestro ejemplo). Ya tenemos un **mapa coroplético** básico. (En la referencia oficial de Leaflet se muestra cómo aplicar estilo basado en propiedades similares, por ejemplo utilizando feature.properties.density en un mapa de densidad de población[[11]](https://leafletjs.com/examples/choropleth/#:~:text=function%20style%28feature%29%20,7)).

**Paso 6: Añadir interacción *hover* con tooltip (caja de información)**. Para que al pasar el ratón por encima de cada comunidad aparezca un recuadro negro con el nombre y valor, tenemos un par de enfoques:

* *Usar un* *tooltip* *de Leaflet:* Desde Leaflet v1.3 hay tooltips integrados que se pueden asociar a capas vectoriales[[12]](https://gis.stackexchange.com/questions/31951/showing-popup-on-mouse-over-not-on-click-using-leaflet#:~:text=http%3A%2F%2Fleafletjs.com%2Freference). Podemos aprovecharlos para mostrar la info de cada CCAA. Por ejemplo, en la opción onEachFeature de L.geoJson, bindear un tooltip a cada *feature*:
* function onEachFeature(feature, layer) {  
   const nombre = feature.properties.name;  
   const valor = datosCCAA[nombre] || 0;  
   layer.bindTooltip(`<b>${nombre}</b>: ${valor}`, { sticky: true });  
  }  
  L.geoJson(geojsonData, {  
   style: style,  
   onEachFeature: onEachFeature  
  }).addTo(map);
* Con { sticky: true }, el tooltip seguirá al cursor mientras esté sobre la región. Podrías añadir opciones CSS para personalizar la apariencia (por defecto el tooltip de Leaflet es una caja ligera; para hacerla negra podrías sobreescribir la clase .leaflet-tooltip en CSS, por ejemplo: background-color: black; color: #fff; border:0; border-radius:4px; padding:5px;). Este método es sencillo y evita manejar manualmente eventos de mouse.
* *Usar eventos* mouseover *y un* *popup: Alternativamente, podríamos usar popups de Leaflet que se abren al hover. Esto era común antes de existir tooltips. La idea es similar: layer.bindPopup(contenido) y luego abrir/cerrar en eventos. Ejemplo:*[*[13]*](https://gis.stackexchange.com/questions/31951/showing-popup-on-mouse-over-not-on-click-using-leaflet#:~:text=Then%20you%20have%20more%20control,be%20bound%20to%20your%20marker)
* layer.bindPopup(`<b>${nombre}</b>: ${valor}`);  
  layer.on('mouseover', function() { this.openPopup(); });  
  layer.on('mouseout', function() { this.closePopup(); });
* Esto lograría un efecto parecido (asegurarse de ajustar la posición del popup si es necesario, para que no desaparezca al mover el ratón[[14]](https://gis.stackexchange.com/questions/31951/showing-popup-on-mouse-over-not-on-click-using-leaflet#:~:text=Note%3A%20You%20may%20run%20into,it%20doesn%27t%20disappear%20too%20easily)). En general, la solución con tooltip es más limpia para mostrar información al vuelo sin clic.

Cualquiera de las dos maneras mostrará una “caja” con la información de la comunidad al pasar el cursor. En nuestro caso, por ejemplo al hacer hover sobre *Andalucía* aparecería algo como "**Andalucía: 120**". Puedes estilizar esa caja para que luzca como desees (fondo negro semi-transparente, texto blanco, etc.).

Adicionalmente, se puede resaltar visualmente la región al hover. Con Leaflet, en el mismo onEachFeature, podemos definir eventos para cambiar el estilo: por ejemplo aumentar el grosor del borde y llevar la capa al frente cuando el mouse entra, y restaurar cuando sale[[15]](https://leafletjs.com/examples/choropleth/#:~:text=Now%20let%E2%80%99s%20make%20the%20states,event)[[16]](https://leafletjs.com/examples/choropleth/#:~:text=function%20resetHighlight%28e%29%20). Siguiendo el tutorial oficial, podrías implementar:

function highlightFeature(e) {  
 const layer = e.target;  
 layer.setStyle({  
 weight: 3,  
 color: '#666',  
 fillOpacity: 0.7  
 });  
 layer.bringToFront();  
}  
function resetHighlight(e) {  
 geojson.resetStyle(e.target);  
}

y ligar estos en onEachFeature:

layer.on({  
 mouseover: highlightFeature,  
 mouseout: resetHighlight  
});

donde geojson es la variable donde guardaste la capa L.geoJson (debes asignarla a una variable antes para poder usar resetStyle). Esto añadirá una retroalimentación visual al usuario al posicionar el cursor (un borde gris más marcado en la comunidad activa). Es opcional pero mejora la UX.

**Paso 7: (Opcional) Añadir una leyenda de colores.** Si deseas una pequeña leyenda que explique el gradiente (por ejemplo “0–20: claro, 20–40: naranja, … >100: rojo”), Leaflet permite crear controles personalizados fácilmente. En el tutorial de choropleth de Leaflet se demuestra cómo agregar un control tipo *legend* fijo[[17]](https://leafletjs.com/examples/choropleth/#:~:text=var%20legend%20%3D%20L.control%28)[[18]](https://leafletjs.com/examples/choropleth/#:~:text=for%20,br%3E%27%20%3A%20%27%2B%27%29%3B). En resumen, se haría:

const legend = L.control({position: 'bottomright'});  
legend.onAdd = function(map) {  
 let div = L.DomUtil.create('div', 'info legend'), grades = [0, 20, 40, 60, 80, 100], labels = [];  
 // generar HTML para cada tramo de colores  
 for (let i = 0; i < grades.length; i++) {  
 const from = grades[i];  
 const to = grades[i+1];  
 div.innerHTML +=  
 `<i style="background:${ getColor(from + 1) }"></i> ` +  
 from + (to ? `&ndash;${to}<br>` : '+');  
 }  
 return div;  
};  
legend.addTo(map);

Y definir estilos CSS para la clase .legend y .legend i acorde al diseño deseado[[19]](https://leafletjs.com/examples/choropleth/#:~:text=CSS%20styles%20for%20the%20control,class%20defined%20earlier). Esto mostrará recuadros de color con sus intervalos como referencia visual.

**Paso 8: Soportar filtros dinámicos (actualización de datos)**. Una vez implementado el mapa, querrás que al cambiar ciertos filtros en la UI (p. ej., seleccionar distintos tipos de normativa, años, etc.), los colores se actualicen según los **nuevos datos** correspondientes. En arquitectura, lo común es hacer una nueva petición al backend para obtener los valores filtrados por CCAA, y luego actualizar la visualización. Con Leaflet hay un par de formas de hacerlo:

* **Actualizar propiedades y re-estilizar:** Si la geometría de las regiones no cambia (siempre serán las 17 CCAA), podemos reutilizar la misma capa GeoJSON y simplemente actualizar el valor usado para colorear. Por ejemplo, tras recibir nuevos datos, actualiza el objeto datosCCAA con los nuevos valores y luego llama a geojson.resetStyle(). Al hacer esto, Leaflet re-ejecutará la función style(feature) para cada región[[16]](https://leafletjs.com/examples/choropleth/#:~:text=function%20resetHighlight%28e%29%20), aplicando los nuevos colores. Asegúrate de que en style(feature) estás obteniendo el valor desde la variable global datosCCAA para que recoja los cambios. Si también cambias el contenido del tooltip, podrías iterar por cada capa y actualizar su bindTooltip content o volver a llamar a bindTooltip en onEachFeature.
* **Remover y agregar capa de nuevo:** Una estrategia sencilla es remover la capa GeoJSON del mapa y crearla de nuevo con los nuevos datos. Por ejemplo:
* map.removeLayer(geojson);  
  geojson = L.geoJson(geojsonData, { style: style, onEachFeature: onEachFeature }).addTo(map);
* Esto garantiza que los colores y tooltips se recalculen con el nuevo dataset. Dado que el número de regiones es pequeño (17), el costo de recrear la capa es bajo.

En ambos casos, el mapa interactivo reflejará los datos filtrados. Si tus filtros cambian el *rango* de valores significativamente, tal vez debas ajustar también la escala de colores (ej.: podrías redefinir las clases en getColor o los límites de la visualización). Para una escala realmente dinámica, podrías implementar getColor para calcular colores gradualmente. Por ejemplo, usando una librería como **chroma.js** o simplemente interpolando entre dos colores según la proporción del valor dentro del mínimo-máximo actual. Leaflet no hace esto automáticamente, pero ECharts sí lo haría con visualMap. Aún así, es totalmente factible de implementar manualmente en Leaflet si se requiere.

**Paso 9: Integración con Node/MongoDB:** La guía anterior se centró en el front-end. Para conectarlo con tu stack: - El backend en Node.js debería exponer un endpoint (por ejemplo, /api/volumen-normativa?filtro=X) que devuelva un JSON con los valores por comunidad autónoma según los filtros seleccionados. Ese JSON podría tener la estructura { "Andalucía": 120, "Aragón": 80, ... } o una lista de objetos con código y valor. - En el cliente, cuando el usuario cambie un filtro, realizar una petición (fetch) a ese endpoint, obtener los datos y luego ejecutar la lógica de actualización de Leaflet (actualizar datosCCAA y recalcular estilos como vimos en el paso 8). - La capa GeoJSON de las CCAA (la geometría) puede estar almacenada estáticamente en el front (ya que las fronteras no cambian) o servirse una vez desde el backend. Incluso podrías guardar ese GeoJSON en MongoDB o en el propio código. Como buena práctica, mantenerlo en un archivo a ser cargado es suficiente, dada su rareza de cambio.

Siguiendo estos pasos, **Cursor** (tu asistente de código) debería ser capaz de implementar la funcionalidad leyendo esta guía e interactuando con tu base de código. **Resumiendo**, con Leaflet lograrás un mapa coroplético interactivo de España: - Coloreado por Comunidad Autónoma según tus datos (dummy o reales) con un degradado personalizable[[10]](https://leafletjs.com/examples/choropleth/#:~:text=function%20getColor%28d%29%20,FFEDA0)[[11]](https://leafletjs.com/examples/choropleth/#:~:text=function%20style%28feature%29%20,7). - Con tooltips al pasar el mouse mostrando el nombre y valor en una “caja” informativa. - Posibilidad de resaltar regiones al hover para mejor experiencia. - Capacidad de actualizar los colores cuando cambien los datos (filtros dinámicos), simplemente recalculando estilos o re-renderizando la capa.

Esta solución cumple con los requisitos usando solo tecnologías gratuitas y compatibles con tu stack. Leaflet, en particular, es **fácil de integrar** en tu front-end vanilla y está optimizado para mapas interactivos ligeros[[4]](https://mappinggis.com/2025/04/las-mejores-apis-javascript-para-webmapping/#:~:text=,Mejor%20soporte%20para%20m%C3%B3viles). Con fuentes abiertas para los mapas de España y unos pocos cientos de líneas de código, obtendrás el resultado deseado de forma rápida y manteniendo buenas prácticas de desarrollo web. ¡Ahora ya puedes proceder a implementar tu mapa de calor de normativa por CCAA!

**Referencias:**

* Leaflet – *Ejemplo de mapa coroplético interactivo* (documentación oficial)[[7]](https://leafletjs.com/examples/choropleth/#:~:text=This%20is%20a%20case%20study,yet%20to%20start%20doing%20so)[[11]](https://leafletjs.com/examples/choropleth/#:~:text=function%20style%28feature%29%20,7).
* MappingGIS – *Comparativa de librerías JS para mapas web (Leaflet vs otras)*[[3]](https://mappinggis.com/2025/04/las-mejores-apis-javascript-para-webmapping/#:~:text=Leaflet%20es%20tambi%C3%A9n%20una%20librer%C3%ADa,la%20simplicidad%2C%20rendimiento%20y%20usabilidad)[[4]](https://mappinggis.com/2025/04/las-mejores-apis-javascript-para-webmapping/#:~:text=,Mejor%20soporte%20para%20m%C3%B3viles).
* StackExchange GIS – *Mostrar popups/tooltips al hacer hover con Leaflet*[[13]](https://gis.stackexchange.com/questions/31951/showing-popup-on-mouse-over-not-on-click-using-leaflet#:~:text=Then%20you%20have%20more%20control,be%20bound%20to%20your%20marker)[[12]](https://gis.stackexchange.com/questions/31951/showing-popup-on-mouse-over-not-on-click-using-leaflet#:~:text=http%3A%2F%2Fleafletjs.com%2Freference).
* OpendataSoft – *Dataset geográfico de Comunidades Autónomas de España* (GeoJSON con límites administrativos)[[8]](https://public.opendatasoft.com/explore/dataset/georef-spain-comunidad-autonoma/?flg=en-us#:~:text=Autonomous%20communities%20)[[9]](https://public.opendatasoft.com/explore/dataset/georef-spain-comunidad-autonoma/?flg=en-us#:~:text=Enhancements).

[[1]](https://mappinggis.com/2025/04/las-mejores-apis-javascript-para-webmapping/#:~:text=D3JS%20es%20una%20librer%C3%ADa%20JavaScript,un%20mapa%20sencillo%20con%20D3) [[2]](https://mappinggis.com/2025/04/las-mejores-apis-javascript-para-webmapping/#:~:text=Image%3A%20D3%20librer%C3%ADa%20JavaScript%20para,crear%20mapas%20web) [[3]](https://mappinggis.com/2025/04/las-mejores-apis-javascript-para-webmapping/#:~:text=Leaflet%20es%20tambi%C3%A9n%20una%20librer%C3%ADa,la%20simplicidad%2C%20rendimiento%20y%20usabilidad) [[4]](https://mappinggis.com/2025/04/las-mejores-apis-javascript-para-webmapping/#:~:text=,Mejor%20soporte%20para%20m%C3%B3viles) [[6]](https://mappinggis.com/2025/04/las-mejores-apis-javascript-para-webmapping/#:~:text=Google%20ha%20conseguido%20democratizar%20el,servicio%20de%20pago%20por%20uso) Las 10 mejores librerías JavaScript para crear mapas web

<https://mappinggis.com/2025/04/las-mejores-apis-javascript-para-webmapping/>

[[5]](https://echarts.apache.org/en/download-map.html#:~:text=To%20create%20Map%20charts%2C%20it,Please) Download Maps - Apache ECharts

<https://echarts.apache.org/en/download-map.html>

[[7]](https://leafletjs.com/examples/choropleth/#:~:text=This%20is%20a%20case%20study,yet%20to%20start%20doing%20so) [[10]](https://leafletjs.com/examples/choropleth/#:~:text=function%20getColor%28d%29%20,FFEDA0) [[11]](https://leafletjs.com/examples/choropleth/#:~:text=function%20style%28feature%29%20,7) [[15]](https://leafletjs.com/examples/choropleth/#:~:text=Now%20let%E2%80%99s%20make%20the%20states,event) [[16]](https://leafletjs.com/examples/choropleth/#:~:text=function%20resetHighlight%28e%29%20) [[17]](https://leafletjs.com/examples/choropleth/#:~:text=var%20legend%20%3D%20L.control%28) [[18]](https://leafletjs.com/examples/choropleth/#:~:text=for%20,br%3E%27%20%3A%20%27%2B%27%29%3B) [[19]](https://leafletjs.com/examples/choropleth/#:~:text=CSS%20styles%20for%20the%20control,class%20defined%20earlier) Interactive Choropleth Map - Leaflet - a JavaScript library for interactive maps

<https://leafletjs.com/examples/choropleth/>

[[8]](https://public.opendatasoft.com/explore/dataset/georef-spain-comunidad-autonoma/?flg=en-us#:~:text=Autonomous%20communities%20) [[9]](https://public.opendatasoft.com/explore/dataset/georef-spain-comunidad-autonoma/?flg=en-us#:~:text=Enhancements) Autonomous communities - Spain — Opendatasoft

<https://public.opendatasoft.com/explore/dataset/georef-spain-comunidad-autonoma/?flg=en-us>

[[12]](https://gis.stackexchange.com/questions/31951/showing-popup-on-mouse-over-not-on-click-using-leaflet#:~:text=http%3A%2F%2Fleafletjs.com%2Freference) [[13]](https://gis.stackexchange.com/questions/31951/showing-popup-on-mouse-over-not-on-click-using-leaflet#:~:text=Then%20you%20have%20more%20control,be%20bound%20to%20your%20marker) [[14]](https://gis.stackexchange.com/questions/31951/showing-popup-on-mouse-over-not-on-click-using-leaflet#:~:text=Note%3A%20You%20may%20run%20into,it%20doesn%27t%20disappear%20too%20easily) Showing popup on mouse-over, not on click using Leaflet? - Geographic Information Systems Stack Exchange

<https://gis.stackexchange.com/questions/31951/showing-popup-on-mouse-over-not-on-click-using-leaflet>