# Cápsula 2: Datos geográficos

Hola, bienvenidxs a una cápsula del curso Visualización de Información. En esta introduciré los datos georreferenciados, su procesamiento y uso con D3.js.

Para comenzar, debemos hablar de GeoJSON. Este es el formato estándar para guardar y codificar datos georreferenciados en la web. Les sonará por la parte final: JSON, un formato de datos que hemos ocupado antes.

GeoJSON es de hecho un caso específico de JSON, es un formato de archivos que usan la estructura JSON, con llaves y valores. Pero se usa de forma específica para guardar datos georreferenciados, y tiene una forma más específica de guardar dicha información.

En pantalla podrás ver el comienzo de un archivo en formato GeoJSON, llamado "paises.json" que extraje de un repositorio público y que se referencia en pantalla. Les compartiré este archivo con el código de las cápsulas, pero ten en cuenta que es algo pesado ya que contiene geometría de todo el mundo.

El archivo completo se contiene entre llaves, como un gran objeto, de tipo "FeatureCollection", y tiene un atributo "features" con una lista de más objetos. Así se organizan generalmente los archivos GeoJSON, como una colección de "features", o elementos, donde cada uno define de forma separada su geometría.

El elemento que alcanzamos a ver es de tipo "Feature" y específica algunas propiedades en el campo "properties", donde podemos ver que aparentemente se refiere al país de Aruba. También tiene un campo "geometry" que es un objeto que describe la geometría del elemento.

Este objeto específicamente dice ser de tipo "MultiPolygon", que es una de muchas opciones de geometría que permite GeoJSON. Otras variedades son puntos simples, líneas o polígonos simples. No iremos tan en detalle cómo se diferencia cada especificación de geometría, ya que de esa magia se encargará D3.

Lo que sí es común, es que en la propiedad "coordinates" se especifican las longitudes y latitudes de las coordenadas que definen cada geometría.

Nota que GeoJSON maneja las componentes geográficas en ese orden: longitud primero, y latitud segundo; a pesar de que el orden normal es el inverso. Esto es porque esta correspondencia va mejor con el usual orden de "x e y". Lo que pretendemos hacer es convertir una longitud en una posición horizontal, y la latitud en una posición vertical.

Ahora, cargaremos este *dataset* mediante D3.js y lo mostraremos en pantalla. Como cargaremos un archivo, debemos usar una función de cargado como "d3.json", como

muestra el código en pantalla. Nota que antes de esto, hay una sección de código que define el SVG donde mostraremos el mapa y sus medidas.

Al cargarse los datos, contaremos con un objeto por cada elemento geométrico, que en este caso corresponden a países del mundo. Lo que buscamos hacer es transformar estas geometrías a marcas gráficas dentro de SVG. De forma similar a lo que vimos con marcas varias, D3.js provee funciones que generan caminos de "path" SVG, y lo hace a partir de geometrías de GeoJSON.

La función "d3.geoPath", del paquete "d3-geo", hace justo eso, retorna un generador de caminos a partir de datos geográficos. Sobre esta función llamamos a "projection", que permite especificar la función de proyección que usará nuestro generador, y por ahora usaremos "d3.geoMercator". Iré en detalles de qué se trata esta proyección en la siguiente cápsula.

Con eso queda poco, solo queda generar un área por elemento en el SVG. Para eso podemos realizar un *data join* con el arreglo de "features" cargado, y por cada uno agregar un elemento "path", cuyo atributo "d" se definirá a partir de nuestro generador de caminos geográficos. Finalmente se les agrega un poco de color para verlo mejor.

Si probamos este código, tal vez se demore un poco por el tamaño del *dataset*. Eventualmente vemos que termina, ¡y muestra parte de un mapa! Pero algo anda raro, no muestra Australia, parece estar cortado. Esto es porque por defecto el generador de caminos usa ciertos tamaños y escala de reproducción.

Podemos cambiar dicha geometría mediante los métodos "translate" y "scale" sobre nuestra proyección, que especifican respectivamente el centro de la extensión visual para el mapa, y la escala de tamaño a utilizar. Usaremos los valores en pantalla para probar, y veremos que funciona y que veremos todo centrado, pero algo pequeño. Esto último es porque la escala es muy baja.

Podemos probar con varios valores de escala hasta encontrar uno que nos acomode. Pero alternativamente podemos usar otro método "fitSize", que recibe la extensión del espacio a usar, y un objeto de GeoJSON que hacer cupir en el espacio específicado. Con esto se traslada y ajusta la escala automáticamente. Si lo probamos, vemos que ahora sí podremos ver todo el mapa de forma correcta.

Mediante este método podemos fijar la escala incluso a elementos específicos dentro de la colección de "features". Por ejemplo, si le entregamos el dato en la posición 41, obtenemos que el mapa se ajusta a las posiciones de Chile, que justamente es el elemento en esa posición (del arreglo).

Con eso termina el contenido de esta cápsula. Recuerda que si tienes preguntas, puedes dejarlas en los comentarios del video para responderlas en la sesión en vivo de esta temática. ¡Chao!