Cápsula 4: *Idioms* para datos geográficos en D3.js

Hola, bienvenidxs a una cápsula del curso Visualización de Información. En esta realizaré un ejemplo de implementación de *idiom* para datos geográficos haciendo uso de D3.js.

Ya vimos cómo visualizar datos puramente geográficos, pero en la práctica rara vez trataremos con solo datos geográficos, algún tipo de datos tabular se acompaña probablemente.

Comenzaremos implementando un mapa coroplético similar al del ejemplo en la primera cápsula. Para ello, busqué la densidad poblacional a lo largo del mundo. Encontré en The World Bank un *dataset* que contiene esta información para la mayoría de países en múltiples años distintos. Por lo que extraje un archivo tabular simple con la densidad del año 2018 por país.

Con eso entonces tenemos de forma separada las geometrías de países en el GeoJSON antes utilizado, y los valores de densidades poblaciones en un archivo CSV. Si seguimos la idea del *data join* donde generamos una marca de área por elemento geométrico, entonces debemos también conectar de alguna forma nuestras dos fuentes de información.

Es posible realizar esto mediante nuestro programa de D3.js, haciendo cargados simultáneos de los *datasets*, pero opté por tiempo y simplicidad, generar un programa en Python que realiza esta unión.

Mediante un código identificador de países, pude agregar una propiedad a cada ”feature” del archivo GeoJSON original con la información de densidad correspondiente. Hay países con los cuales no se contaba información de densidad, por el *dataset* encontrado, así que a esos elementos no se les asignó esta propiedad simplemente.

Con eso tenemos que toda la información estará contenida en un solo archivo GeoJSON, más grande todavía. Así que en nuestro código cargaremos este nuevo archivo y definiremos una escala para los valores de densidad poblacional que hay en el *dataset*, y lo llevarán *colormap* continuo y secuencial.

Para generar esa escala debí tener varios cuidados. Primero, debí filtrar los datos usados para considerar como dominio: sólo deben considerarse regiones que tengan el atributo de densidad definido, y debí marcar un tope máximo de valores a considerar.

Esto último debido a que de forma impresionante, hay algunos valores de densidad que son valores aislados y son extremadamente altos en comparación al resto del mundo. El considerarlos generaría que prácticamente todo el mapa se vea igual, ya que los valores o son muy altos o muy bajos. Y se escogió arbitrariamente el valor de 500 de tope.

Con el esta escala secuencial, se genera una función de escala de color que entrega el valor que corresponde. Si el dato de densidad existe pero es muy alto, deja el valor máximo del *colormap* por defecto, si no pasa del tope, usa la escala secuencial recién creada, y si no existe el valor, deja un gris.

Con esa construcción, la adaptación del *data join* es bastante directa, solo basta con adaptar y usar las escalas en la definición de atributos de las regiones. Se usa la escala de color para el atributo “fill” y un gris claro para los bordes. Si probamos la implementación, ¡obtenemos el resultado! Un mapa coroplético muy similar a lo revisado en la primera cápsula.

Otra codificación usual que se realiza en estos contextos es mostrar puntos específicos en mapas generales, o agregar burbujas en ciertas coordenadas cuyo tamaño codifica algún atributo de interés. Ambas ideas podemos realizarlas con las herramientas que hemos visto hasta el momento.

La única diferencia que se apreciaría con nuestro código actual es cómo obtener las posiciones en nuestra representación bidimensional de un único punto geográfico compuesto por longitud y latitud. El generador de caminos utilizado transforma geometrías complejas a caminos para elementos “path”, por lo que usarlo sería mucho tal vez.

Y la respuesta es usar la función de proyección directamente. Recuerda que su función es exactamente transformar puntos del mundo geográfico al mundo bidimensional. Para verlo en práctica, agregaré dos elementos HTML de *input* numérico y un botón, que utilizaremos para agregar puntos en el mapa en vivo.

Con el código en pantalla, si se aprieta el botón, se usan los valores que hayan en los *inputs* en ese momento, y de haber valores numéricos con sentido, entonces se agrega un punto al SVG cuyas coordenadas se calculan usando la función de proyección definida anteriormente. La primera coordenada sería “x” y la segunda sería “y”.

Si lo probamos, vemos el mapa, y podemos usar los valores de longitud -70 y latitud -18, que son las coordenadas aproximadas de Arica, al norte de Chile. ¡Vemos que efectivamente deja un punto donde esperaríamos! Puedes jugar con este ejemplo, buscando coordenadas de lugares que te interesen, y ubicándolos en el mapa coroplético de este ejemplo.

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!