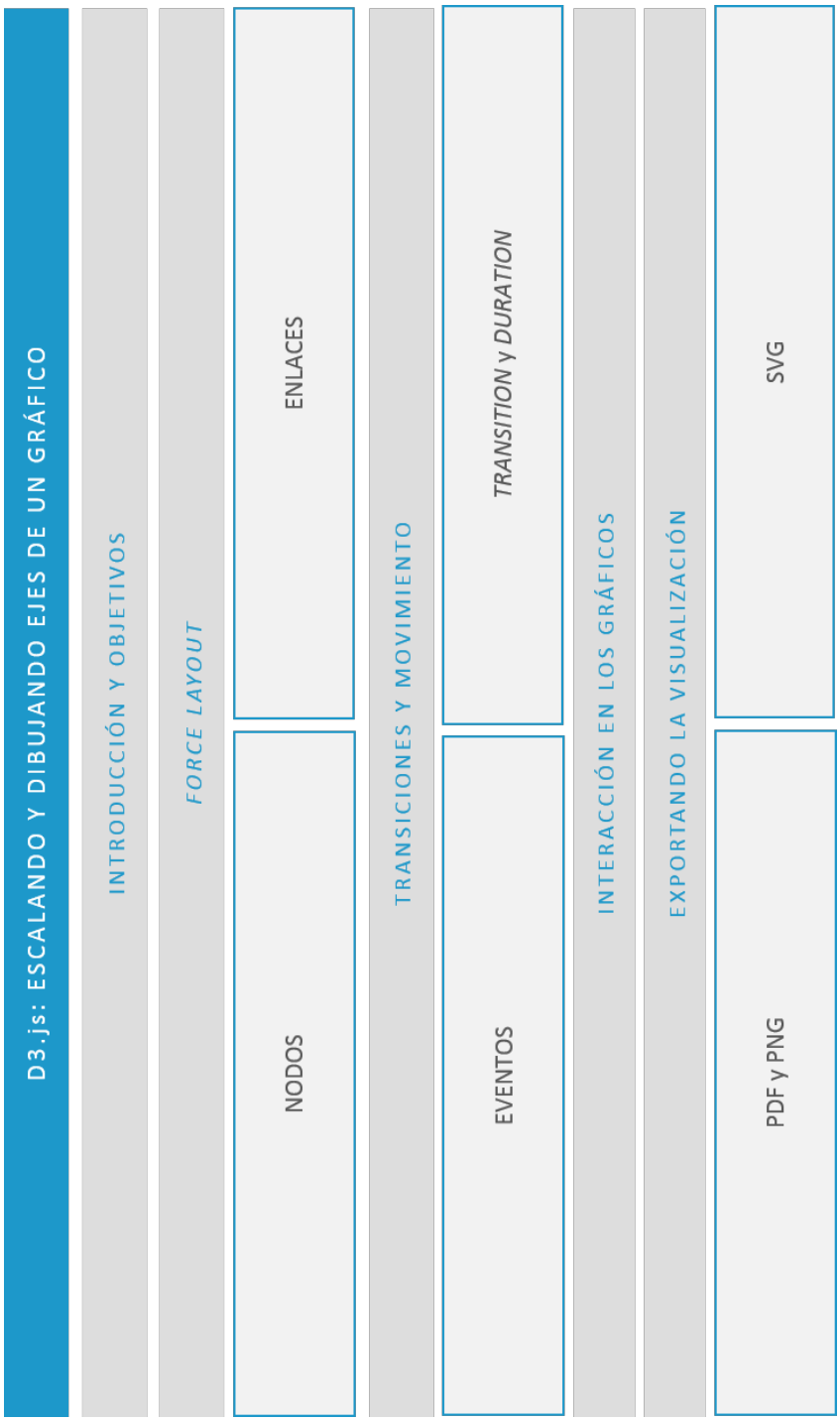
Herramientas de Visualización

Force Layout, transiciones, movimiento e interacción

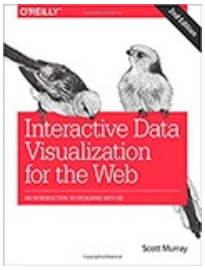
Índice

Esquema



Interactive Data Visualization for the Web

Murray, S. (2017). *Interactive Data Visualization for the Web.* [S. l.]: O'Reilly.

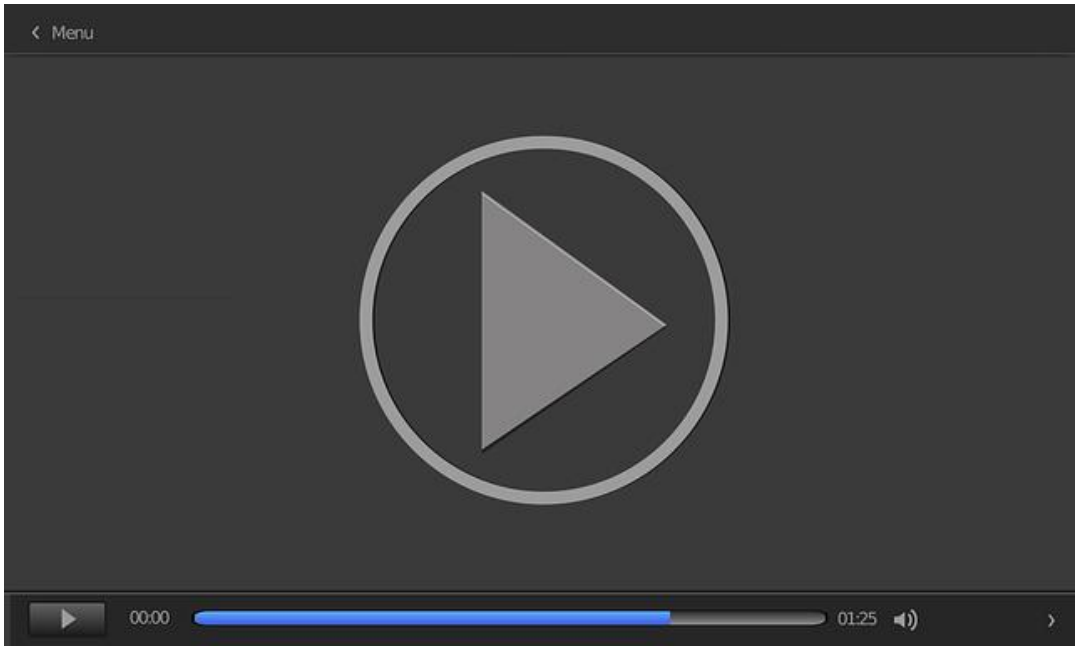


Murray, S. (2017). Dos son las secciones que nos interesan para este tema: Transiciones y actualización de las escalas.

Anand S - Beautiful visualizations with d3.js

HasGeek TV. (6 de diciembre de 2012). *Anand S - Beautiful visualizations with d3.js* [Vídeo]. Youtube. https://youtu.be/JAlZ0uJnqkA

Esta interesante conferencia-taller trata los conceptos de la creación de visualizaciones con D3. Podrás sumergirte en los ejemplos específicos que muestran.



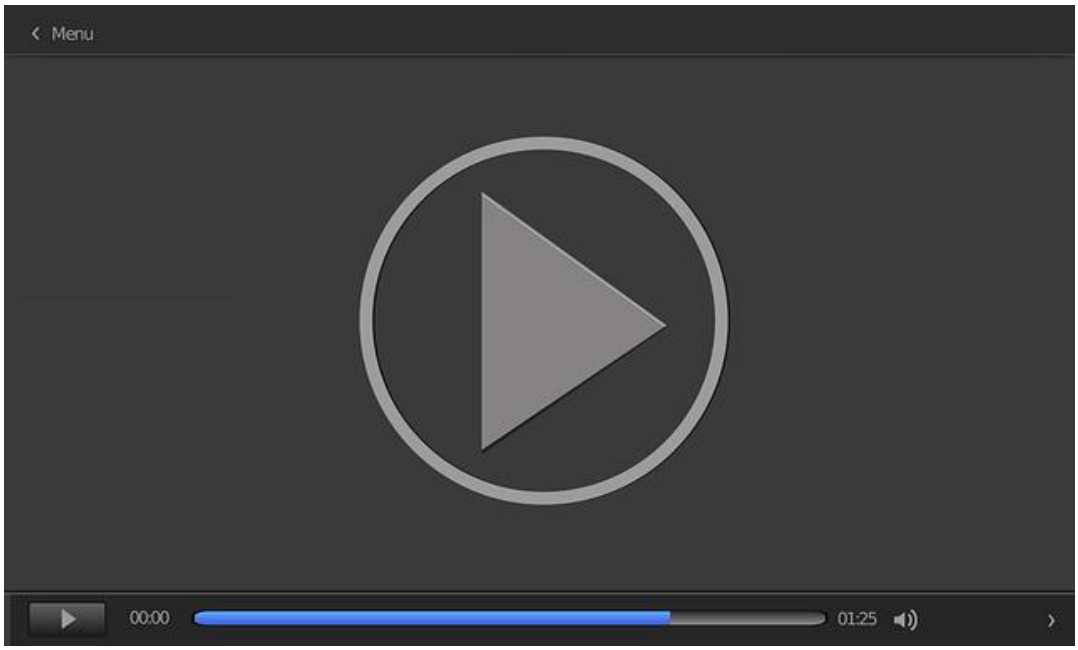
Accede al vídeo:

https://www.youtube.com/embed/JAlZ0uJnqkA

React y D3js trabajando juntos

Lemoncoders. (5 de julio de 2018). *React y D3js trabajando juntos* [Vídeo]. Youtube. https://youtu.be/8OcpV3tCBYU

Siempre es interesante el uso de múltiples librerías combinándose entre sí. En este caso os mostramos el uso de D3.js y React.



Accede al vídeo:

https://www.youtube.com/embed/8OcpV3tCBYU

Data Driven Delirious: An Intro to Data Visualisation Using D3.js

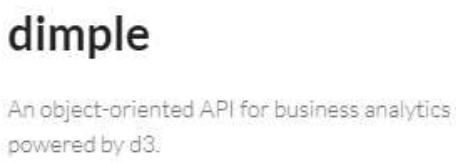
WDCNZ. (9 de julio de 2013). *Data Driven Delirious: An Intro to Data Visualisation Using D3.js* [Vídeo]. Vimeo. https://vimeo.com/72467564

Es bueno escuchar lo que nos pueden enseñar otros expertos, así que en este vídeo podéis encontrar una excelente charla de Alex Gibson en WDCNZ sobre visualización de datos con D3.js.

Dimple

Dimple. Página web oficial. http://dimplejs.org/

Mencionamos algunas librerías que, basadas en D3, intentan simplificar la toma de contacto con esta flexible librería. Con Dimple, convertiremos visualizaciones D3js en una librería bastante parecida en simplicidad a Google Charts.



Visual Sedimentation

Huron. S. (1 de abril de 2019). Visual Sedimentation. *Aviz*. https://www.aviz.fr/Research/VisualSedimentation

Otra librería basada en D3.js, JQuery y Box2DWeb, se especializa en visualizar datos en tiempo real.

NVD3

NVD3. Página web oficial. http://nvd3.org/

Librería que simplifica de manera similar a Dimple las visualizaciones más populares basadas en D3.js.

6.1. Introducción y objetivos

D3 incorpora multitud de gráficas diferentes y, aunque cada una de ellas necesita parámetros específicos adaptados a sus particularidades, lo interesante es que dominéis aquellos principios básicos de D3 que son comunes a todas ellas.

En este tema abordaremos un nuevo tipo de visualización denominada *Force-Directed Graph* o *Force Layout*, orientada a representar las **relaciones entre entidades** mediante un grafo. Es una gráfica muy interesante y especialmente utilizada, por ejemplo, en el ámbito de las redes sociales para reflejar las relaciones existentes entre diferentes usuarios.

Ya sabemos cómo dibujar un gráfico por completo, en este tema también empezaremos a darle movimiento. D3 es una de las librerías de visualización más potentes, por lo que hemos de exprimirla al máximo.

Trataremos sobre cómo actualizar datos reaccionando a la generación de eventos y desarrollaremos una parte importante relativa al dinamismo de las visualizaciones, como son las transiciones y el movimiento de los objetos, para darle una percepción de realidad o para apoyar nuestro mensaje. Por último, veremos cómo se pueden exportar las visualizaciones.

6.2. Force Layout

En esta ocasión nos apoyaremos en uno de los *datasets* referenciados en el libro *Getting started with D3* de M. Dewar (2012). En concreto, utilizaremos uno de los *datasets* que el autor tiene en Github.

Accede al *dataset* desde el aula virtual o a través del siguiente enlace: https://github.com/mikedewar/getting_started_with_d3.git

---

Para generar este gráfico habría que partir del archivo .json, cargándolo por ejemplo en Brackets. Se puede descargar o copiar desde la siguiente dirección:

Accede al archivo .json desde el aula virtual o a través del siguiente enlace: https://github.com/mikedewar/getting_started_with_d3/blob/master/visualisations/data/stations_graph.json

---

Código de Force Layout

```
<html>
   <head>
   <link rel="stylesheet" type="text/css" href="styles.css"/>
   <script   language="javascript" type="text/javascript"
      src="http://d3js.org/d3.v3.min.js"></script>
   <script type="text/javascript"> function read(){
      d3.json("stations_graph.json", function(json) {
          visualizeLayout(json);
      });
}
function visualizeLayout(data){
   var width = 1500
  var height = 1500
  var svg = d3.select("body")
  .append("svg")
```

```
  .attr("width", width)
  .attr("height", height);
  var node = svg.selectAll("circle.node")
  .data(data.nodes)
  .enter()
  .append("circle")
  .attr("class", "node")
  .attr("r", 12);
  var link = svg.selectAll("line.link")
  .data(data.links)
  .enter().append("line")
  .style("stroke","black");
  var force = d3.layout.force()
  .charge(-120)
  .linkDistance(30)
  .size([width, height])
  .nodes(data.nodes)
  .links(data.links)
  .start();
  force.on("tick", function() {
       link.attr("x1", function(d) { return d.source.x; })
       .attr("y1", function(d) { return d.source.y; })
       .attr("x2", function(d) { return d.target.x; })
       .attr("y2", function(d) { return d.target.y; });
       node.attr("cx", function(d) { return d.x; })
       .attr("cy", function(d) { return d.y; });
});
}
   </script>
   </head>
<body onload="read()">
</body>
</html>
```

Como siempre, dividiremos el código y lo explicaremos por partes. También explicaremos el archivo .json, que contiene los nodos (son paradas de metro) y las conexiones entre las paradas.

El gráfico que deberías obtener es el siguiente:

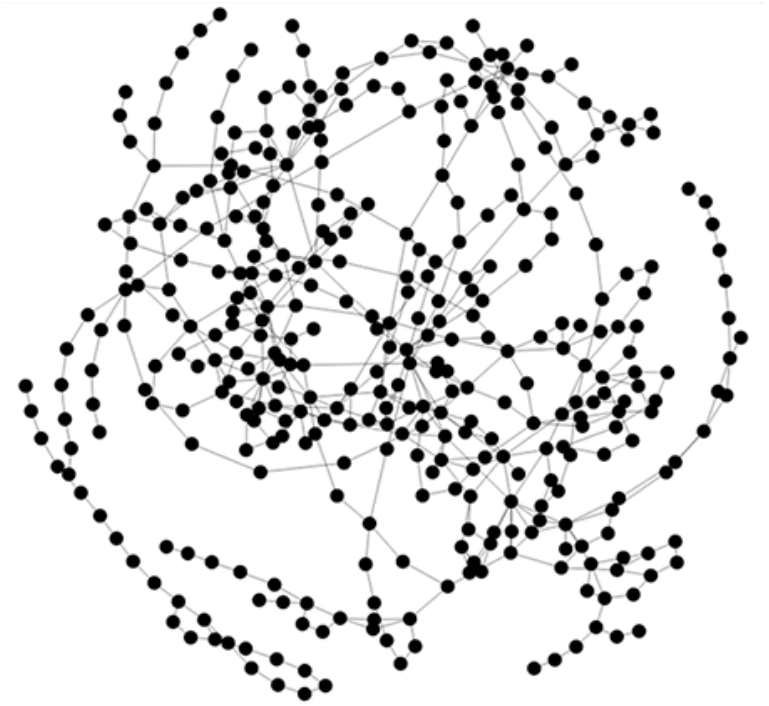


Figura 1. Gráfica con paradas de metro y las conexiones entre dichas paradas.

Fijémonos en las siguientes líneas de código que definen los **círculos o puntos** del gráfico. Ahora sí que hay una diferencia en la identificación de los círculos, aunque por ahora, sea similar a cuando dibujamos el histograma.

```
var node = svg.selectAll("circle.node")
   .data(data.nodes)
   .enter()
   .append("circle")
   .attr("class", "node")
   .attr("r", 12);
```

Y ahora definimos las **líneas** entre los puntos.

```
var link = svg.selectAll("line.link")
   .data(data.links)
   .enter()
   .append("line")
   .style("stroke","black");
```

A continuación, definimos el **algoritmo** que queremos utilizar. En este caso . Y para este algoritmo en concreto, se utilizan los atributos y .

```
var force = d3.layout.force()
   .charge(-120)
   .linkDistance(30)
   .size([width, height])
   .nodes(data.nodes)
   .links(data.links)
   .start();
```

Con la función adaptamos los valores de nuestro *dataset* a los requerimientos del algoritmo:

```
force.on("tick", function() {
   link.attr("x1", function(d) { return d.source.x; })
   .attr("y1", function(d) { return d.source.y; })
   .attr("x2", function(d) { return d.target.x; })
   .attr("y2", function(d) { return d.target.y; });
   node.attr("cx", function(d) { return d.x; })
   .attr("cy", function(d) { return d.y; });
```

Por último, algo muy típico de este tipo de gráficos es proporcionar la habilidad de arrastrar los nodos por la pantalla. Si añadimos línea de código, lograremos arrastrar cualquier nodo con solo hacer clic en él.

6.3. Actualización de gráficos con base en eventos

Como hemos explorado en otras librerías, todas las visualizaciones tienen sus eventos. ¿Cómo podemos generar esa interacción en nuestras propias visualizaciones? Con D3 veremos que todas estas acciones son muy fáciles. Hemos de entender cómo se producen, pero una vez que entendamos los conceptos básicos, no serán más que unas cuantas líneas de código.

> «*Time has been transformed, and we have changed; it has advanced and set us in motion; it has unveiled its face, inspiring us with bewilderment and exhilaration*» (Brainy Quote).

El movimiento es el secreto de la vida. No importa lo mal que estemos, debemos seguir hacia delante pase lo que pase. Las visualizaciones son reflejo de la vida. Dan mensajes, expresan ideas, sugieren caminos, por lo que darles movimiento parece ser una idea coherente a estos principios. Este tema trata de esto: dar interacción y movimiento a nuestras visualizaciones.

Tomemos como base para continuar el código de la gráfica *bar chart* con escala ordinal que desarrollamos en el anterior tema. Sobre dicho código vamos a explicar cómo trabajar con eventos.

Vamos a seguir cuatro pasos:

- Crear un botón que genere un evento para actualizar nuestro gráfico.
- Crear un nuevo *dataset* y actualizar la gráfica con él.
- Actualizar las dimensiones del gráfico.
- Hacer efectiva las actualizaciones.

Crear un botón

Todo elemento en D3.js puede ser un botón, siendo más estrictos, **todo elemento puede generar eventos**. Para comenzar, generaremos un texto y le asignaremos un *pop-up* como resultado a nuestro evento.

```
d3.select("body")
.append("p")
.text("Pulsa aquí")
.on("click", function() {
    alert("Conseguido! Ahora vamos a hacer algo útil...");
    });
```

Con este código antes de la definición del svg, al pinchar en el texto de «Pulsa aquí», se obtiene el siguiente resultado:

Figura 2. Evento en una gráfica.

Generar un nuevo *dataset*

Esto nos va a servir para actualizar en tiempo real la gráfica al pulsar el botón, que es nuestro objetivo real. El nuevo *dataset* sería el siguiente:

Actualizar las dimensiones del gráfico

El siguiente paso sería actualizar los valores de nuestro gráfico y adaptar altura y anchura al nuevo *dataset*:

```
     svg.selectAll("rect")
.data(dataset)
.attr("y", function(d) {
     return h - yScale(d);
})
.attr("height", function(d) {
     return yScale(d);
});
```

Introducimos todo esto en nuestro botón:

```
d3.select("body")
.append("p")
.text("Pulsa aquí")
.on("click", function() {
    dataset = [ 11, 12, 15, 20, 18, 17, 16, 18, 23, 25, 5, 10];
    svg.selectAll("rect")
    .data(dataset)
    .attr("y", function(d) {
          return h - yScale(d);
})
    .attr("height", function(d) {
        return yScale(d);
});
      });
```

Obtendremos el siguiente resultado:

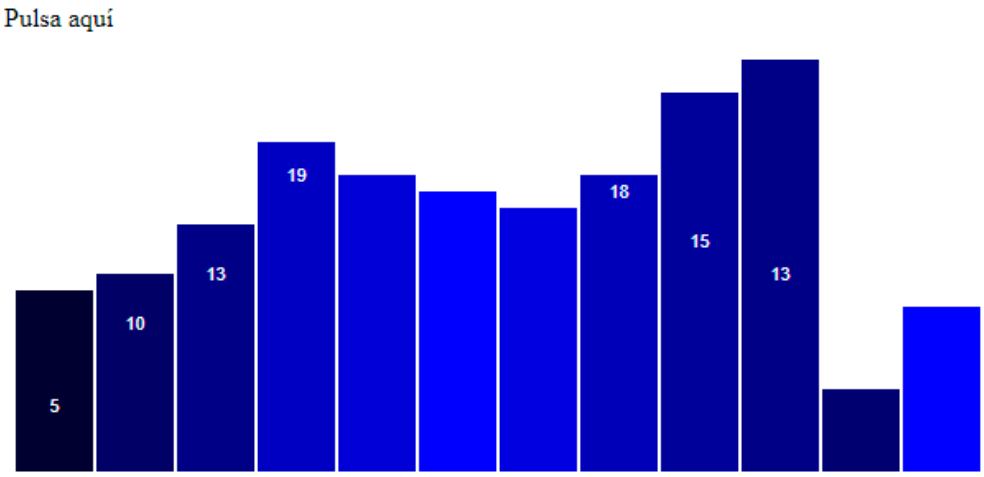


Figura 3. Gráfica actualizada con un nuevo *dataset*.

Hacer efectiva las actualizaciones

Si miramos el gráfico, vemos que no hemos actualizado el texto. Añadamos al evento el siguiente código:

```
svg.selectAll("text")
   .data(dataset)
   .text(function(d) {
   return d;
})
.attr("x", function(d, i) {
   return xScale(i) + xScale.rangeBand() / 2;
})
.attr("y", function(d) {
   return h - yScale(d) + 14;
});
```

Además, vamos a hacer que después de pulsar en el texto, este desaparezca borrando nuestro botón. Para ello añadimos lo siguiente:

Este es el resultado final:

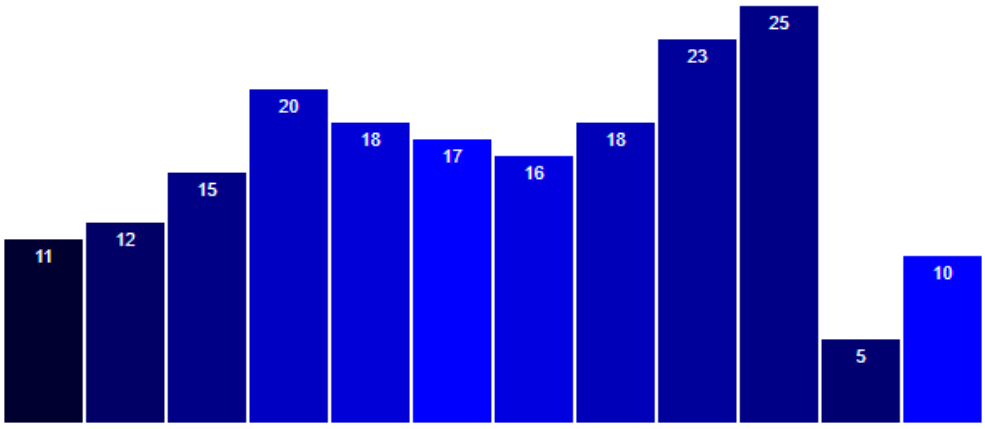


Figura 4. Gráfica final con el *dataset* actualizado.

6.4. Transiciones y movimiento

Las transiciones son el movimiento que dará vida a nuestro gráfico y en D3.js podemos hacerlo con una simple línea de código. Existe división de opiniones en cuanto a las transiciones: para algunos, estas atraen a usuarios más noveles y otros las consideran un adorno innecesario en visualizaciones.

¿Cómo se hacen las transiciones? Una simple línea de código: **transition()**. Se incluye en el código donde creábamos el botón para actualizar el *bar chart*. Al pulsar sobre tu evento, verás que el gráfico será animado.

```
svg.selectAll("rect")
   .data(dataset)
   .transition()
   .attr("y", function(d) {
      return h - yScale(d);
})
   .attr("height", function(d) {
     return yScale(d);
});
```

¿Qué podemos controlar en estas transiciones? Una de las cosas que podemos controlar es la **duración** utilizando duration(milisegundos). Probad este código:

```
svg.selectAll("rect")
   .data(dataset)
   .transition()
   .duration(3000)
   .attr("y", function(d) {
      return h - yScale(d);
})
   .attr("height", function(d) {
      return yScale(d);
});
```

hace que la animación dure tres segundos. Verás que las barras «crecen», sin

embargo, el texto de las barras está fijo. Por tanto, también deberíamos aplicar la **transición al texto**.

```
svg.selectAll("text")
   .data(dataset)
   .transition()
   .duration(3000)
   .text(function(d) {
      return d;
})
   .attr("x", function(d, i) {
     return xScale(i) + xScale.rangeBand() / 2;
})
   .attr("y", function(d) {
     return h - yScale(d) + 14;
});
```

Si nos fijamos en la transición, primero es lenta y luego acelera. Esto lo podemos modificar y regular con otra función () y hacer la **transición lineal**.

```
svg.selectAll("rect")
   .data(dataset)
   .transition()
   .duration(3000)
      .ease("linear")
   .attr("y", function(d) {
      return h - yScale(d);
})
   .attr("height", function(d) {
      return yScale(d);
});
```

Si solo modificas las barras pero no el texto, notarás la diferencia de cómo evolucionan los dos elementos. Prueba con otros métodos, no solo el lineal. Por ejemplo, o y observa cómo reacciona.

Otra de las particularidades que tenemos a nuestra disposición consiste en **retrasar la animación** con la función . Retrasaremos tres segundos nuestra animación:

```
svg.selectAll("rect")
   .data(dataset)
   .transition()
   .duration(3000)
      .ease("linear")
   .delay(3000)
   .attr("y", function(d) {
      return h - yScale(d);
})
   .attr("height", function(d) {
      return yScale(d);
});
```

6.5. Añadiendo interacción a los gráficos

Los modos de interaccionar con un gráfico son tan diversos como los estilos de los diseñadores. Por ejemplo, imaginemos que queremos que reaccione el valor al pulsar una barra. Al saber el valor, también podemos hacer otras acciones. Practiquemos cómo ver el valor, aunque esté escrito en la barra, en una ventana abierta.

```
svg.selectAll("rect")
   .data(dataset)
   .on("click", function(d) {
      alert(d);
})
   .transition()
     .ease("linear")
.attr("y", function(d) {
      return h - yScale(d);
})
.attr("height", function(d) {
      return yScale(d);
});
```

Haz clic en cualquiera de las barras y aparecerá una pequeña ventana con el valor de dicha barra.

Ahora bien, si queremos hacer algo más interesante, por ejemplo, hacer cambiar las barras cuando pasamos por encima, podemos probar el siguiente código:

```
svg.selectAll("rect")
   .data(dataset)
   .on("mouseover", function(d) {
d3.select(this)
   .style("fill", "red");
     })
       .on("mouseout", function(){
d3.select(this)
```

```
   .style("fill", function(d) {
          return "rgb(0, 0, " + (d * 10) + ")";
         });
    })
   .transition()
      .ease("linear")
   .attr("y", function(d) {
      return h - yScale(d);
})
   .attr("height", function(d) {
      return yScale(d);
});
```

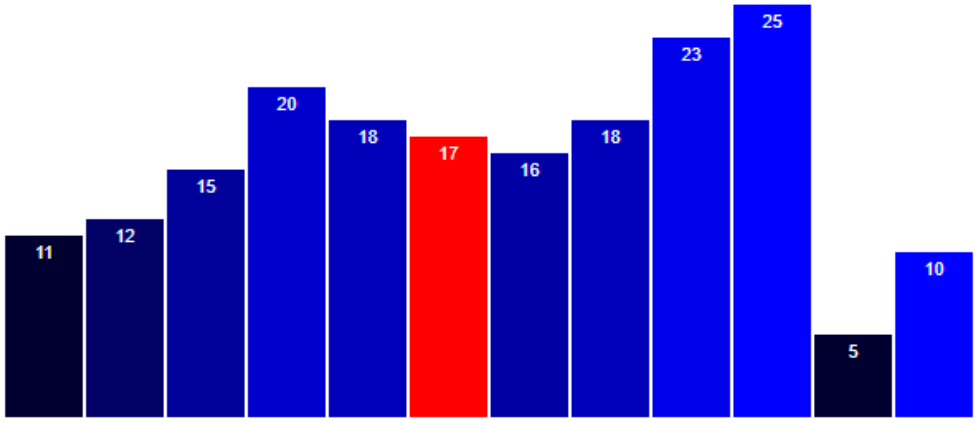


Figura 5. Gráfica con efecto de barra resaltada.

6.6. Exportando el resultado a PDF, Bitmaps y SVG

Esta sección no está muy relacionada con D3.js en sí, pero es evidente la utilidad de poder exportar nuestro gráfico desde la ventana del navegador a otros formatos.

- La primera opción es la captura de pantalla (Print Screen en PC y Command ⌘ + Shift + 3 en MAC), que nos generará un archivo de tipo Bitmap o PNG. Es la forma más tradicional y fácil para obtener resultados, aunque de baja calidad.
- También podemos imprimir en PDF, seleccionando dicho formato en la pantalla de opciones de impresión.
- La opción más interesante es trabajar con **SVG**, así podremos modificar la imagen en Photoshop, escalar la imagen a todos los tamaños, etc.

Para eso, en el caso anterior, seleccionaremos el gráfico, pulsaremos el botón derecho del ratón y seleccionaremos **Inspeccionar**, como indica la siguiente figura:

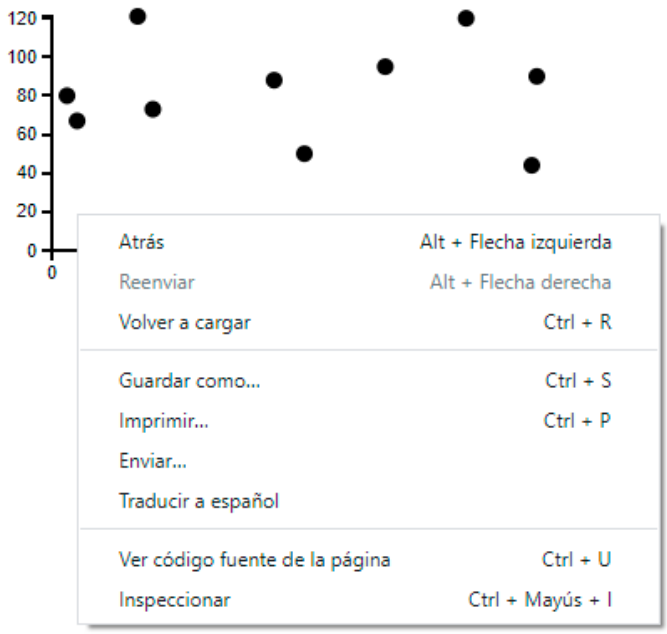


Figura 6. Inspeccionar elemento.

Seleccionaremos todo lo que hay entre las etiquetas <svg>…</svg>.

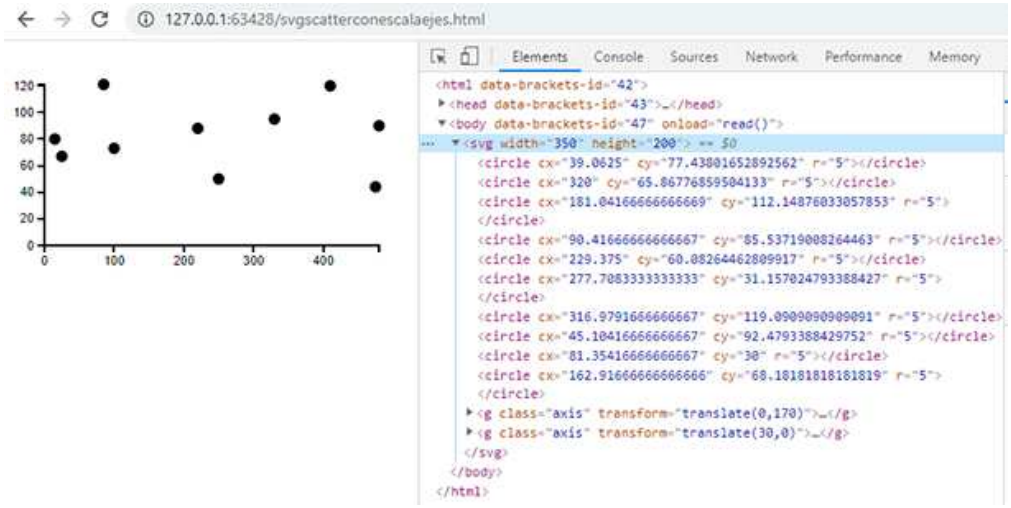


Figura 7. Detalle código SVG.

Lo copiaremos en un archivo nuevo que tenga la extensión .svg. Este archivo lo podrás abrir con Acrobat Reader, Photoshop, etc.

6.7. Referencias bibliográficas

Brainy Quote. *Khalil Gibran Quotes*. https://www.brainyquote.com/quotes/khalil_gibran_101832

Dewar, M. (2012). *Getting Started with D3*. (S. l.): O'Reilly Media.

1. ¿Cuánto tardará la transición en este código?

```
svg.selectAll("text")
.data(dataset)
.text(function(d) {
return d;
})
.attr("x", function(d, i) {
return xScale(i) + xScale.rangeBand() / 2;
})
.attr("y", function(d) {
return h - yScale(d) + 14;
});
```

A. 1 seg.

B. 2 seg.

C. Depende del número de barras del gráfico.

D. No hay transición.

2. ¿Cuánto tardará la transición en este código?

```
svg.selectAll("text")
.data(dataset)
.text(function(d) {
return d;
})
.transition()
.delay(1000)
.attr("x", function(d, i) {
return xScale(i) + xScale.rangeBand() / 2;
})
.attr("y", function(d) {
return h - yScale(d) + 14;
});
```

A. 1 seg.

B. 2 seg.

C. Depende del número de barras del gráfico.

D. El tiempo por defecto transición.

3. ¿Es la sintaxis correcta?

```
on("mouseover", function(d) {
d3.select(this)
.style("fill", "red");
})
```

A. Sí.

B. No, porque falta la selección del objeto.

C. No, porque el evento mouseover no existe.

D. No, porque la llamada a la función es incorrecta.

4. ¿Es la sintaxis correcta?

```
on("outmouse", function(d) {
d3.select(this)
.style("fill", "red");
})
```

A. Sí.

B. No, porque falta la selección del objeto.

C. No, porque el evento outmouse no existe.

D. No, porque la llamada a la función es incorrecta.

5. ¿Podemos escuchar eventos en los elementos p de D3?

A. Sí.

B. No, porque hacen falta objetos que puedan aceptar un clic de ratón sobre ellos.

C. No, porque los eventos aplican solo a los datos.

D. No, porque p no es un elemento de D3.

6. ¿Podemos escuchar/generar eventos en los elementos `de D3`?

A. Sí.

B. No, porque hacen falta objetos que puedan aceptar un clic de ratón sobre ellos.

C. No, porque los eventos aplican solo a los datos.

D. No, porque rect no es un elemento de D3.

7. ¿Qué conseguimos cuando utilizamos en la transición la función ?

A. Que la transición sea constante.

B. Que la transición dure lo mínimo posible.

C. Que la transición de los objetos sea vertical.

D. Que la transición de los objetos sea horizontal.

8. Si queremos exportar nuestra visualización a un archivo svg, tenemos que:

A. Exportar nuestro gráfico con la función D3.exportSVG.

B. Copiar el código HTML generado y copiarlo en un archivo svg.

C. No existe la posibilidad. Se pueden crear y visualizar gráficos SVG pero no exportarlos.

D. Ninguna de las anteriores.

9. Utilizando captura de pantalla para exportar nuestra visualización a otro formato:

A. Generamos un archivo Bitmaps.

B. Generamos un archivo SVG.

C. Generamos un archivo PDF.

D. No es posible capturar la pantalla.

10. ¿Cuál es el comportamiento al aplicar la función ?

A. Los nodos se pueden arrastrar y, cuando sueltas el nodo, este se queda en esa misma posición.

B. Los nodos se pueden arrastrar y, cuando sueltas el nodo, vuelven a su posición original.

C. Los nodos se mueven de forma independiente por sí mismos en un ciclo continuo.

D. Ninguna de las anteriores.