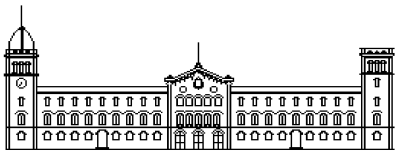


Treball fi de carrera

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques
Universitat de Barcelona

Adaptación de grafos reutilizables D3 para NVD3

Adrián González Martín

Directora: Mireia Ribera Turró
Realitzat a: Departament de Matemàtica e Informàtica. UB

Barcelona, 26 de Enero de 2017

Universidad de Barcelona

ABSTRACT

La siguiente memoria contiene un estudio teórico-práctico de cómo adaptar gráficos existentes en D3 para NVD3. La memoria se centra en la explicación de las tecnologías necesarias para realizar esta adaptación, también explica con ejemplos como se lleva a cabo esta adaptación y los pasos necesarios tanto para que un usuario normal pueda usar estos gráficos como los pasos necesarios para que un programador pueda hacer una adaptación él mismo de otro gráfico.

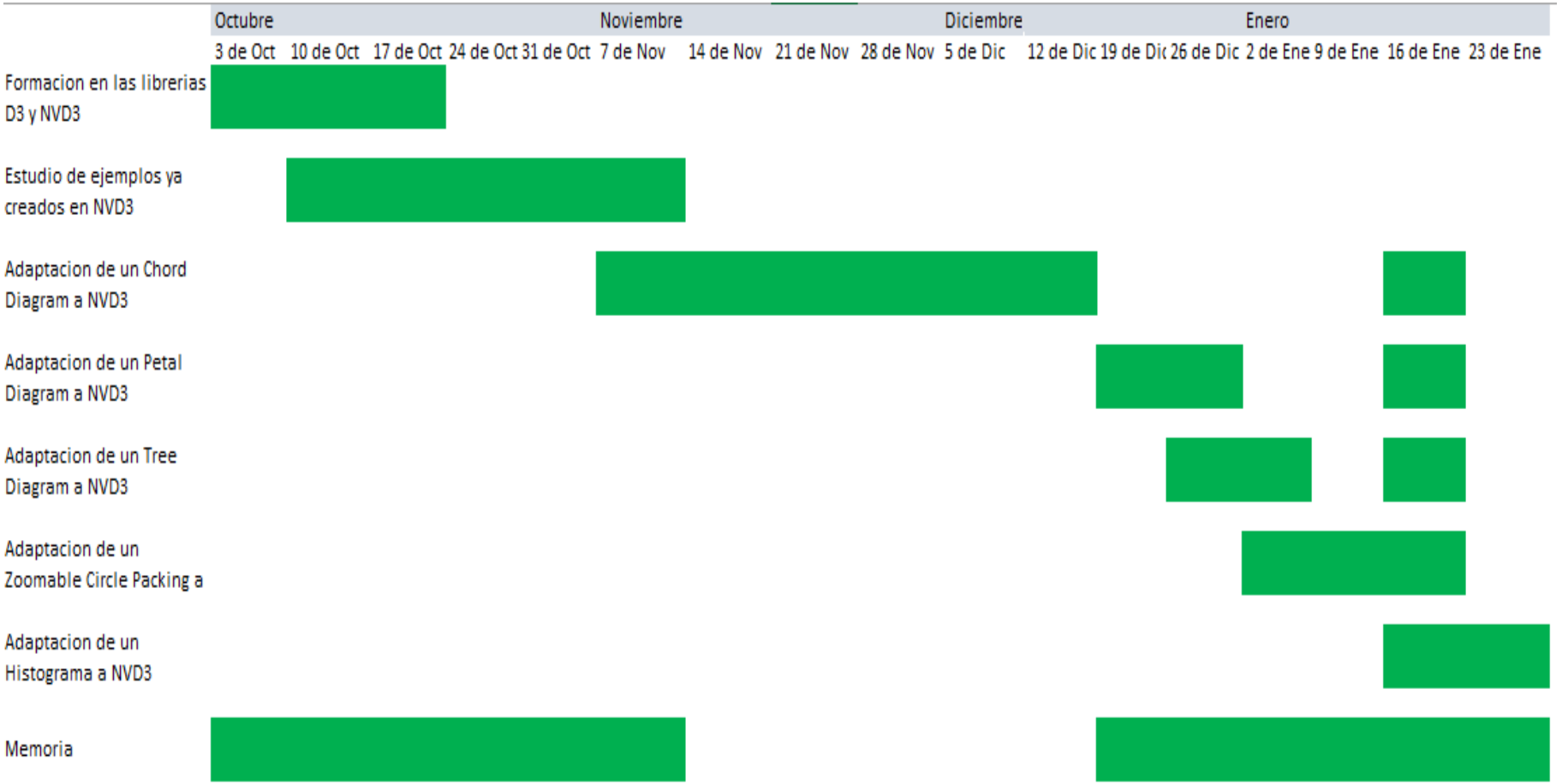
En el mundo de la visualización de datos hay muchas maneras de representar datos, normalmente se suelen representar por gráficos o diagramas para su mejor comprensión. Hay algunos de estos gráficos que ya están implementados tanto en D3 como en NVD3 pero otros muchos aun no lo están. La intención con este TFG es la de enriquecer con más gráficos la librería NVD3 que es muy reciente pero a su vez muy útil para el usuario no experimentado por su sencillez de crear gráficos.

ABSTRACT

The following report contains a theoretical-practical study of how to adapt it to D3 for NVD3. The memory focuses on explaining the technologies needed for the realization of this adaptation, also explained with examples how this adaptation is carried out and the necessary steps both for a normal user can use these graphics as the steps necessary to a programmer can make an adaptation himself of another graph.

In the world of data visualization there are many ways of representing data, usually they are represented by graphs or diagrams for their better understanding. There are some of these graphics that are already implemented in both D3 and NVD3 but many others are not yet. The intention with this TFG is to enrich with more graphics the NVD3 library which is very recent but in turn very useful for the user not experienced for its simplicity of creating graphs.

PLANIFICACIÓN



ÍNDICE TFG

1. Introducción

El trabajo de final de grado es una asignatura de cuarto curso que nos permite profundizar en un tema en concreto.

El TFG elegido se centra en la representación de grafos reutilizables usando D3. Cuando queremos comunicar una idea, en ocasiones podemos utilizar una imagen. Las representaciones visuales nos ayudan a ilustrar los conceptos que, si se expresaran con palabras, nos resultaría difícil de explicar claramente. Cuando tenemos datos que necesitamos para ilustrar conceptos, ideas y propiedades de los datos, el uso de la representación visual nos ofrece un instrumento de comunicación muy válido. La parte más difícil es la definición de estas representaciones.

En el momento de realizar el TFG ya hemos cursado toda la carrera, lo que nos permite aplicar todos los conocimientos a los diferentes apartados de este trabajo. El TFG guarda relación con las siguientes asignaturas:

- Factores Humanos: Utilización del lenguaje HTML para la edición de documentos web. CSS para añadir estilo a los documentos Web y JavaScript
- Software Distribuido: Utilización del lenguaje HTML para la edición de documentos web. Manipulación de ficheros JSON. CSS para añadir estilo a los documentos Web y JavaScript
- Taller de Nuevos Usos de la Informática: Utilización de Big Data, conjunto de datos tan grandes y complejos que resulta difícil de procesar utilizando herramientas de gestión de base de datos o aplicaciones tradicionales de procesamiento de datos.

Para realizar el proyecto se han seguido una serie de procesos y técnicas como programar con JavaScript, la utilización de las librerías D3 y NVD3, la manipulación de documentos CSV y documentos JSON y el uso de gráficos expresados en SVG.

La motivación para realizar este proyecto reside en ayudar a enriquecer la librería NVD3 para que usuarios no muy experimentados en D3 puedan crear gráficos sin tener que programarlos desde cero.

La cantidad de gráficos que existen para ilustrar datos son incontables (gráfico de barras, gráfico de líneas, *scatterplot*, etc…). Muchos de estos gráficos ya están creados en D3 y programadores experimentados pueden adaptarlos a las necesidades de NVD3 y crear gráficos reutilizables. Estos gráficos reutilizables serán más sencillos de utilizar para el usuario final ya que la parte de programación será mínima porque el código ya está creado en la librería NVD3.

Lo interesante del proyecto es la utilización de la librería NVD3 que permite crear un gráfico en D3 haciendo una llamada a la librería NVD3, pasándole los datos y algunos parámetros como por ejemplo el color del gráfico o el tamaño que va a ocupar.

2. Objetivos

El objetivo principal del TFG es explicar los pasos para instanciar gráficos D3 de uso común utilizando la librería NVD3. Para su mejor comprensión, se explicarán los pasos con varios ejemplos. La explicación de los gráficos creados irá en dos direcciones. En primer lugar, dirigida a aquellos programadores que quieran aportar a la librería sus ejemplos, indicándoles los pasos que tienen que seguir para conseguirlo. En segundo lugar, a aquellos usuarios que quieran usar algún ejemplo de la librería, mostrándoles los pasos que han que seguir.

La base del proyecto será el entendimiento y la realización de gráficos reutilizables ya creados en NVD3, que nos ayudarán a desarrollar nuestros ejemplos.

Una vez creados los ejemplos, contribuir con ellos a enriquecer la librería NVD3.

3. Antecedentes

3.1 Conceptos relacionados

Los conceptos, técnicas y tecnologías que he tenido que aprender antes de la realización del TFG son:

Document Object Model o (DOM), es esencialmente una interfaz de plataforma que proporciona un conjunto estándar de objetos para representar documentos HTML, XHTML y XML.

Scalable Vector Graphics o (SVG) es un formato de gráficos vectoriales bidimensionales, estáticos o animados, en formato XML.

Comma-Separated Values o (CSV) son un tipo de documento en formato abierto, sencillo para representar datos en forma de tabla, en el que las columnas se separan por comas y las filas por saltos de línea. En algunos casos, los campos dentro del archivo CSV deben de ir encerrados entre comillas dobles.

JavaScript Object Notation o (JSON) es un formato de texto para el intercambio de datos. JSON describe los datos con una sintaxis dedicada a identificar y gestionar datos. Es una alternativa a XML, por su fácil uso en JavaScript.

D3 es una librería JavaScript que nos permite enlazar datos a un Document Object Model (DOM), y luego aplicar transformaciones en el documento basadas en estos datos. Por ejemplo, se puede utilizar D3 para generar una tabla HTML a partir de una matriz de números. O bien, utilizar los mismos datos para crear un gráfico interactivo SVG con transiciones suaves y con interacción.

Los primeros navegadores Web representaban páginas estáticas, la única función de éstas era pasar de una a otra a través de un enlace. En 1996, *Netscape* introdujo el primer navegador con JavaScript, un nuevo lenguaje de programación que podía ser interpretado por el navegador mientras que la página se está visualizando.

Esto permitió a los navegadores de Internet evolucionar, pasando de ser navegadores estáticos (sin animación) a navegadores dinámicos (con animación).

En 2005, Jeffrey Heer, Stuart Card, y James Landay introdujeron *prefuse*, una librería con un conjunto de herramientas para llevar la visualización de datos a la Web.

Dos años más tarde, Jeff Heer introdujo *Llamarada*, una librería con un conjunto de herramientas similares, pero escrito en ActionScript, por lo que sus visualizaciones podrían ser vistas en la web a través del reproductor de Flash. *Llamarada*, como *prefuse*, se basó en un navegador con un plug-in adicional.

En el año 2009, Jeff Heer se había trasladado a Stanford, donde estaba aconsejando a un estudiante graduado llamado Michael Bostock. Juntos crearon *Protovis*, una librería con un conjunto de herramientas de visualización basado en JavaScript.

Protovis permitía generar visualizaciones sencillas, incluso para usuarios sin experiencia previa en programación. Sin embargo, para lograr este objetivo, se creó una capa de representación abstracta. El diseñador podría entrar en esta capa utilizando sintaxis *Protovis*, pero no era accesible a través de métodos estándar, por lo que era difícil depurar.

En 2011, Mike Bostock, Vadim Ogievetsky, y Jeff Heer anunciaron oficialmente D3. A diferencia de *Protovis*, D3 opera directamente en el documento web en sí. Esto significa que la depuración y la experimentación eran más sencillas. La única desventaja es una curva de aprendizaje más pronunciada que con las otras herramientas.

NVD3 es una librería que ofrece componentes **Charts y Charts reutilizables** construidos sobre d3.js, NVD3 nos ofrece toda la potencia de D3 simplificando su uso.

Charts son una representación gráfica de datos. Para crearlos debemos de especificar los datos y las funciones que hacen que el gráfico se cree. En los **Charts reutilizables,** al estar las funciones creadas en la librería NVD3 únicamente hay que especificar los datos y algún parámetro más para su creación.

3.2 Chord Diagram

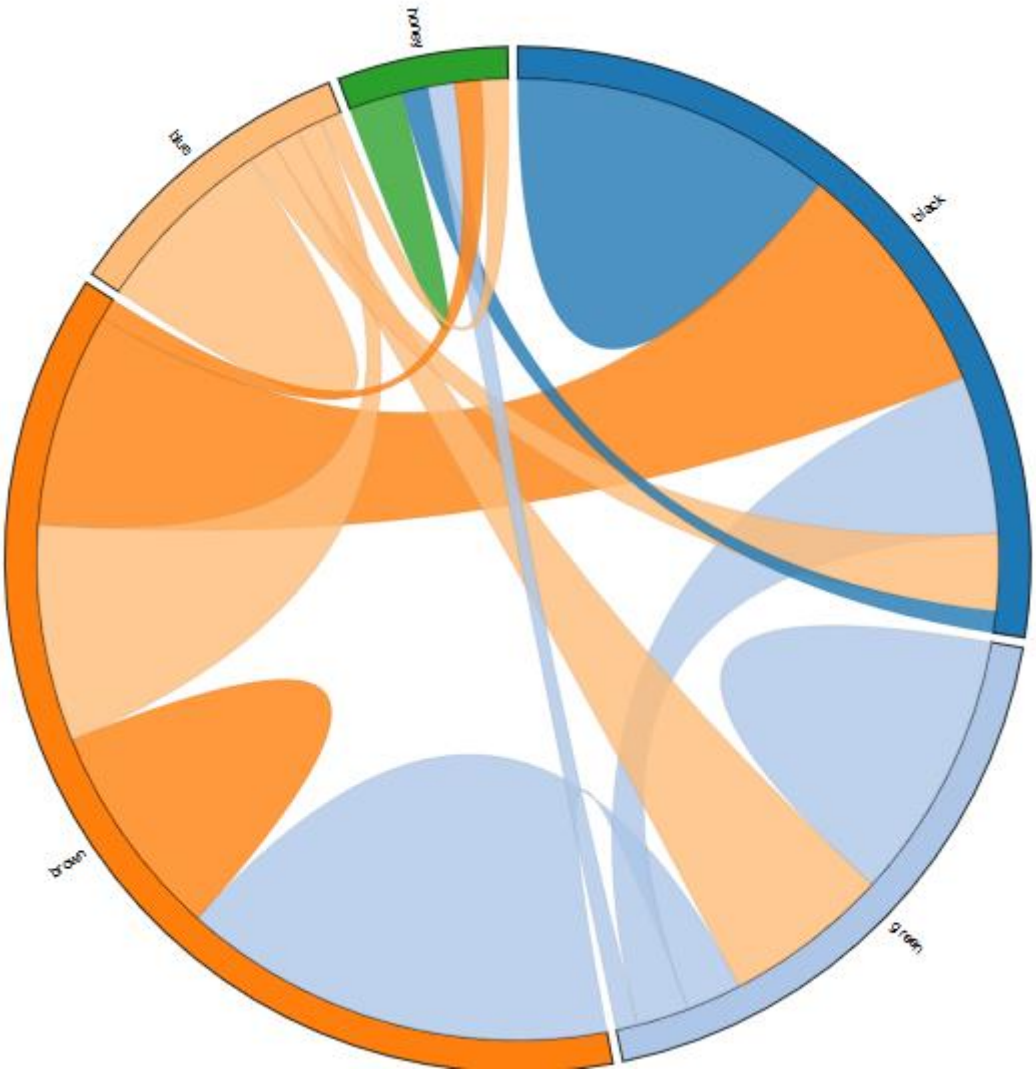
Chord diagram es un gráfico que representa las interrelaciones entre datos de una matriz similar a la que se muestra a continuación:

From/To	Black	Brown	Blue	Green	Honey
Black	11975	8916	2868	5871	790
Brown	8010	8090	8045	16145	790
Blue	1013	940	6907	990	790
Green	1951	2060	6171	10048	790
Honey	990	990	990	990	1990

Como podemos ver, la matriz es cuadrada, y en la primera fila y columna se añade el nombre del dato que se va a representar. En nuestro caso lo datos son: Black, Brown, Blue, Green y Honey. Los valores numéricos que hay dentro de la matriz indican la relación de un dato con otro.

En la representación visual los datos se organizan radialmente alrededor de un círculo en forma de puntos. Las relaciones entre los puntos del círculo se representan como un arco que une dos puntos. Dependiendo del valor numérico que pongamos en la matriz el arco de unión será de mayor o menor tamaño.

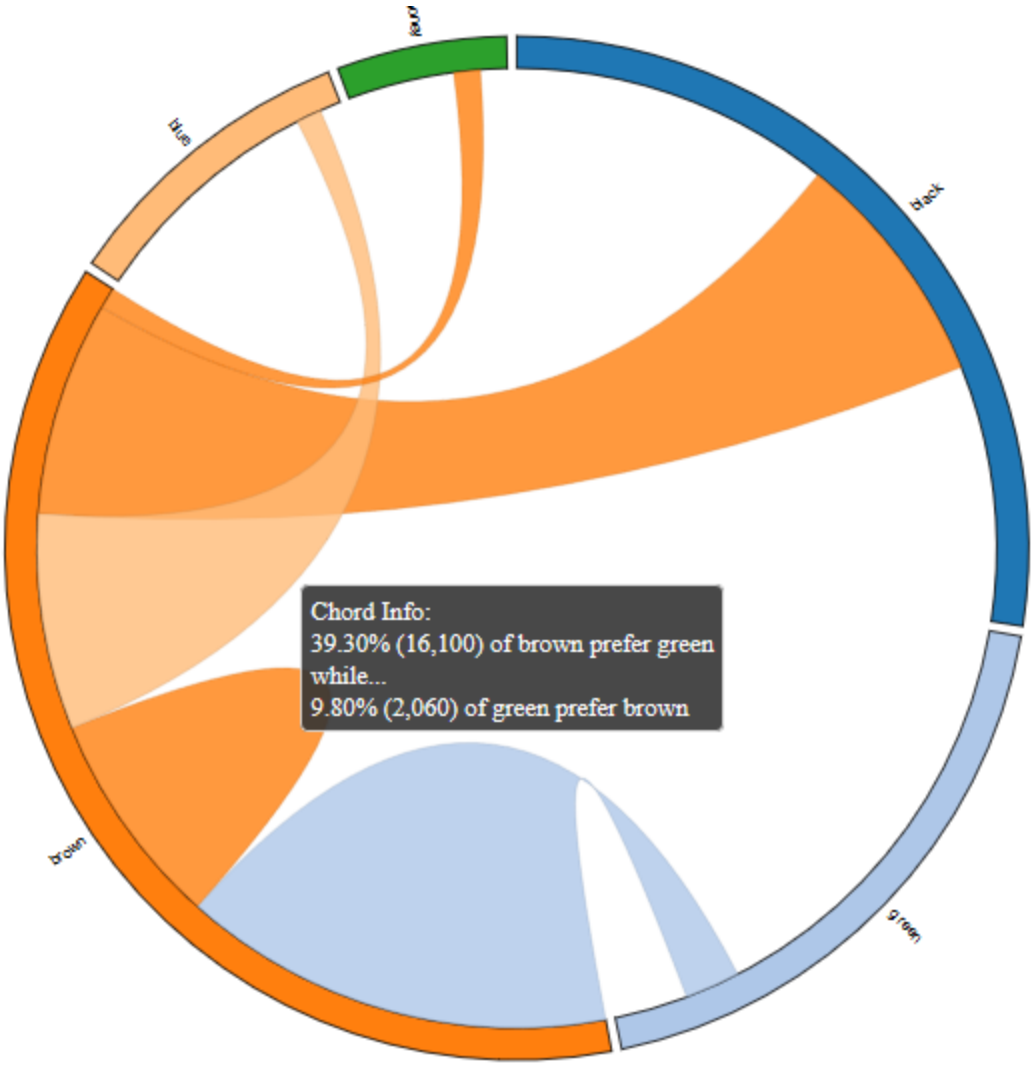
A continuación podemos ver un ejemplo de Chord diagram:



Como hemos mencionado anteriormente, los datos guardados en la matriz se organizan alrededor del círculo en forma de secciones. De esta forma, el dato Black está representado en el gráfico por el color azul marino y el dato Green está representado por el azul celeste.

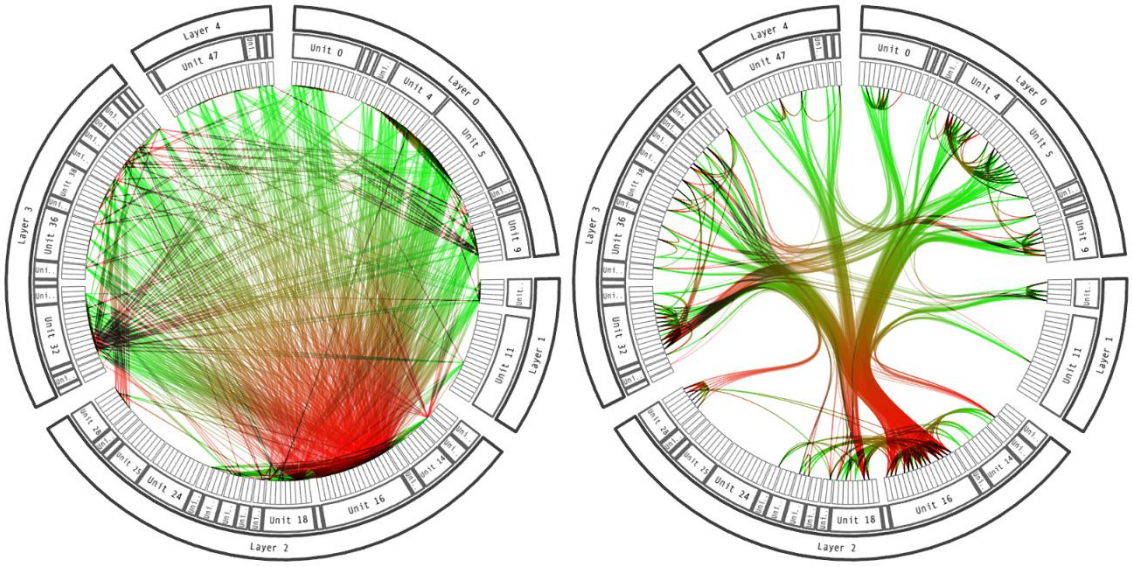
Dependiendo del valor que relacione el dato a con el dato b, el arco será de mayor o menor tamaño.

En la siguiente imagen podemos ver todos los arcos de relación del dato Brown. Si nos fijamos en el arco que relaciona Brown con Green vemos que se crea con los valores introducidos en la matriz ya mencionada. Brown tiene un valor de relación con Green de 16100 y Green tiene un valor de relación con Brown de 2060. Como el primer valor es mucho mayor, el arco tiene que ser de una dimensión mayor en la parte de Brown.



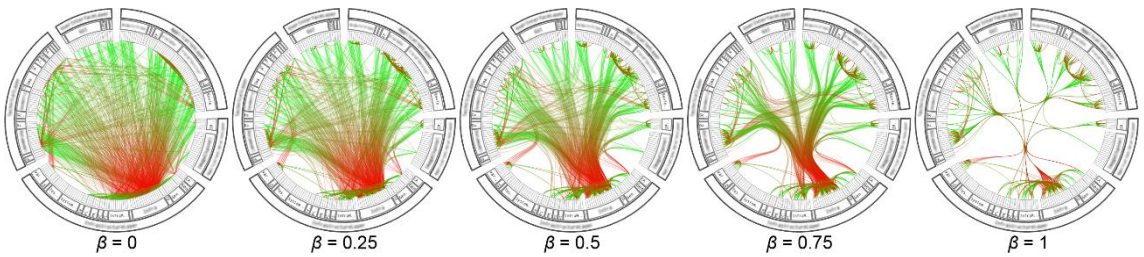
Este gráfico debe su nombre a una terminología utilizada en geometría. La cuerda (chord) de un círculo es un segmento de línea geométrica que une dos puntos situados en el círculo. Los diagramas de cuerdas son también conocidos como diagramas de red radiales.

Los diagramas de cuerdas utilizan una técnica llamada **Hierarchical edge bundles.** Una pequeña cantidad de datos puede ser representada en un diagrama circular usando líneas rectas para mostrar las interconexiones. Esta técnica reduce la complejidad visual del diagrama



Hierarchical edge bundles es un método de agrupación de los bordes de adyacencia. Se hace agrupación de las cuerdas con el fin de reducir el desorden. Esta técnica fue propuesta por Danny Holten, un estudiante de post-doctorado en la Universidad Tecnológica de Eindhoven para presentar los gráficos de una forma más simple.

Se basa en el uso de la fuerza agrupación β para proporcionar un equilibrio entre vistas de bajo nivel y de alto nivel de las relaciones de adyacencia. El valor de β aumenta de izquierda a derecha; valores bajos proporcionan principalmente información de conectividad de nodo a nodo de bajo nivel, mientras que los valores altos proporcionan información de alto nivel.



3.3 Petal Diagram

Petal diagram es un tipo de gráfico circular que se utiliza para representar datos en forma de porcentajes y proporciones. Los datos que son necesarios para crear este diagrama son: el nombre del ítem que se quiere representar y el valor numérico del ítem. Estos datos se pueden organizar en una tabla de 2 columnas, una indicando el nombre del ítem y otra, el valor del ítem.

A continuación vemos un ejemplo:

Item	Value
Telecinco	14.4
Antena 3	13.3
La 1	10.5
Autonómicas	7.4
La Sexta	7.2
Televisión de pago	6.8
Cuatro	6.1

Los datos se organizan radialmente en forma de círculo. Cada pétalo es un ítem distinto y el tamaño del pétalo depende del valor del item.

A continuación podemos ver un ejemplo de petal diagram:

3.4 Tree Diagram

Tree Diagram es un tipo de gráfico que se utiliza generalmente para representar todos los posibles resultados de un experimento. Estos experimentos normalmente van asociados al cálculo de probabilidades. Otro uso muy importante de esta estructura es la representación de esquemas.

Los elementos principales de este diagrama son:

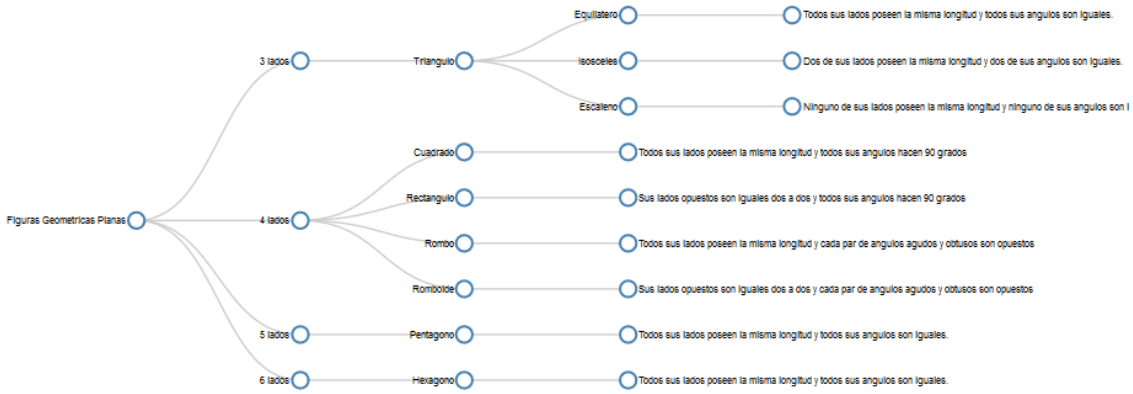
- Nodo: es un elemento de la estructura de un árbol.
- Hijo: es un nodo descendente de otro nodo del árbol.
- Padre: es un nodo con hijos.
- Hojas: son nodos que no tienen hijos.
- Raíz: es el nodo superior del árbol y es el único que no tiene padre.
- Rama: son nodos que son padres y además tienen hijos. Forman una estructura de unión entre una hoja y la raíz.
- Nivel: el nivel de un nodo está definido por el número de conexiones entre el nodo y la raíz.

Para crear un diagrama de árbol necesitamos el nombre del nodo y el nombre de su padre. Estos datos se pueden organizar en una tabla de 2 columnas, una indicando el nombre del nodo y otra, el nombre del padre.

A continuación vemos un ejemplo:

Nodo	Padre
Figuras Geométricas Planas	null
3 lados	Figuras Geométricas Planas
Triángulo	3 lados
Equilátero	Triángulo
4 lados	Figuras Geométricas Planas
Cuadrado	4 lados
Rectángulo	4 lados
5 lados	Figuras Geométricas Planas
Pentágono	5 lados

A continuación podemos ver un ejemplo de tree diagram.



Como podemos ver en el ejemplo, la raíz del árbol se llama Figuras Geométricas Planas. Este nodo raíz tiene 4 hijos que son: 3 lados, 4 lados, 5 lados y 6 lados. Estos 4 hijos son nodos ramas, ya que son hijos y padres a la vez. Por último, el árbol tiene 9 hojas, que es el final de cada una de las ramas que se han ido abriendo en el árbol.

3.5 Circle Packing Diagram

Circle Packing Diagram es un método para visualizar grandes cantidades de datos estructurados jerárquicamente. Este diagrama guarda relación con el tree diagram en lo que a estructura se refiere. Los círculos se encapsulan por niveles, dos círculos tangentes representan nodos hermanos al mismo nivel. El tamaño de un nodo-hoja (aquel que no contiene ningún círculo dentro) se suele corresponder con un valor asociado a ese nodo.

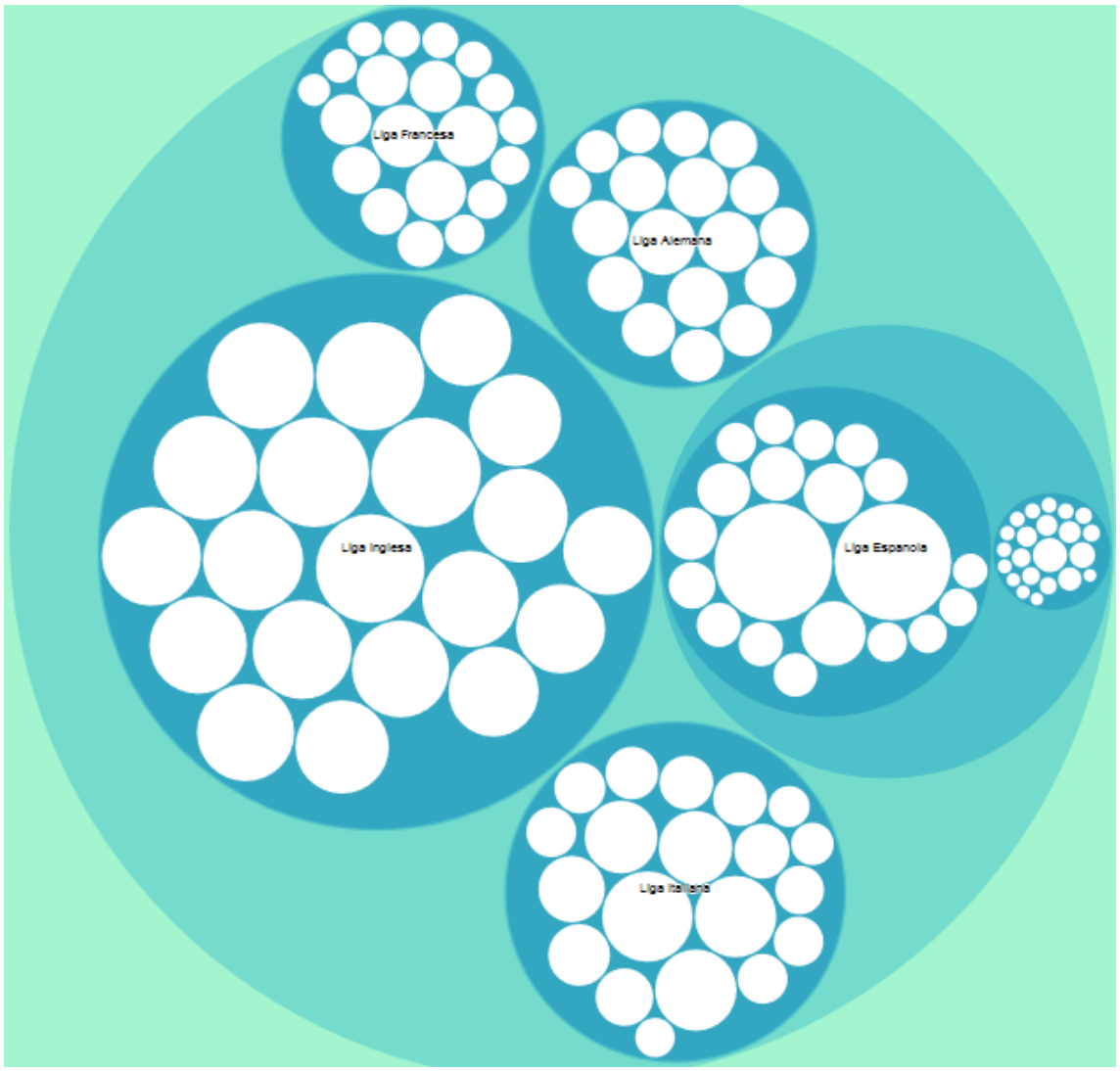
Una ventaja de este gráfico es la buena visión general de grandes conjuntos de datos y la representación clara de las agrupaciones y las relaciones estructurales.

Para crear este diagrama necesitamos el nombre del nodo, el nombre de su padre y el tamaño. Estos datos se pueden organizar en una tabla de 3 columnas: una indicando el nombre del nodo, otra el nombre del padre y otra con el tamaño del nodo. El tamaño del nodo padre va en función de los nodos hijos.

A continuación vemos un ejemplo:

Nodo	Padre	Tamaño
Ligas Europeas	null	
Liga Española	Ligas Europeas	
Primera División	Liga Española	
F.C Barcelona	Primera División	160
Real Madrid C.F	Primera División	156.6
S.D Eibar	Primera División	13.8
Liga Inglesa	Ligas Europeas	
Premier League	Liga Inglesa	
Chelsea	Premier League	138.6
Manchester City	Premier League	137.9
Manchester United	Premier League	135.5
Burnley	Premier League	91.5

A continuación podemos ver un ejemplo de circle packing diagram.



Como podemos ver dentro en el gráfico, hay un círculo grande que se corresponde con la raíz y dentro hay 5 más pequeños que se corresponden con los nodos hijos de la raíz. Los círculos de color blanco son los nodos-hojas. El tamaño de estos nodos depende del valor tamaño asociado al nodo.

3.6 Histograma

Histograma es una representación gráfica de una variable en forma de barras, donde la superficie de cada barra es proporcional a la frecuencia de los valores representados. Comúnmente, sirven para obtener una visión de la distribución de la población respecto a una variable cuantitativa continua.

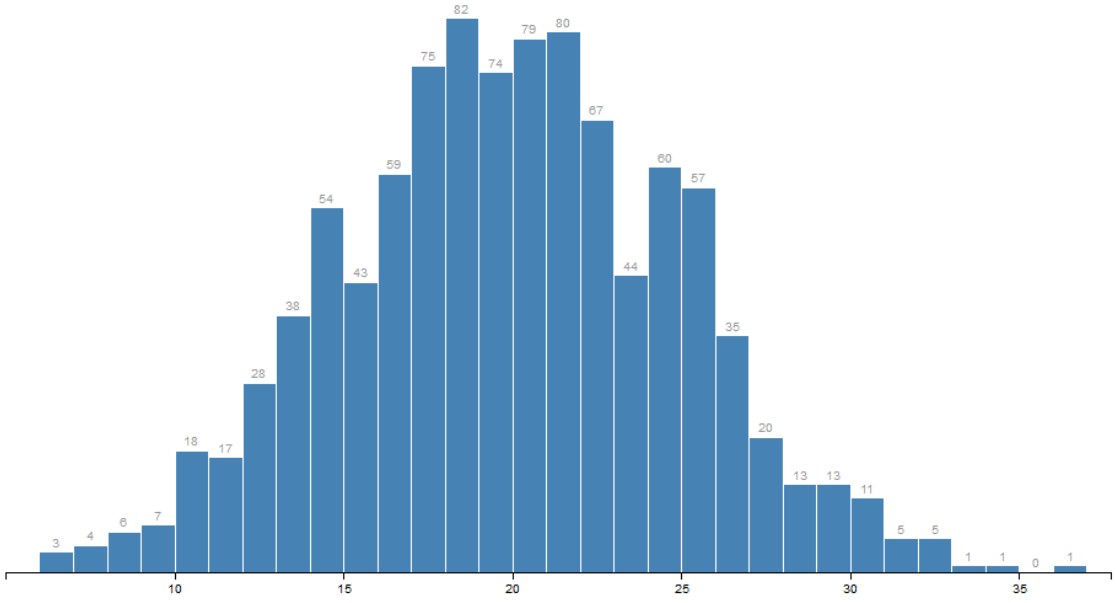
Para crear un histograma necesitamos los valores de la variable continua. Estos datos se pueden organizar en una tabla de 1 columna indicando en cada fila el valor.

A continuación vemos un ejemplo:

Datos
15
20
17
10
20
19
25
23
20

Estos datos introducidos se agruparan por tramos formando las barras del histograma. El número de tramos es el que se encarga de establecer si un dato está en un tramo o en otro. El tamaño de las barras indica el número total de datos asociados al tramo.

A continuación podemos ver un ejemplo de histograma.



4. Implementación

4.1 Conceptos comunes

Si otros programadores quieren incluir nuevos gráficos en NVD3 el primer paso sería bajarse las librerías D3 y NVD3 (ver Apéndice de usuario).

En esta sección se explican los pasos más importantes para la adaptación de gráficos en D3 a la librería NVD3.

Para adaptar cualquier gráfico a la librería NVD3, necesitamos el código del gráfico en D3. Este código ha de ser lo más genérico posible para que se pueda extrapolar a todos los posibles casos.

Todos los gráficos que contiene la librería NVD3 tienen partes comunes. La primera es la llamada a la función *chart,* que contiene la extracción de los datos y la creación del gráfico. La llamada de la función ha de ser parecida a la que vemos a continuación.

```
function chart(selection) {
    renderWatch.reset();
    selection.each(function(data) {
```

La variable *selection* contiene los datos que hemos introducido en el *select* del archivo HTML, como podemos ver a continuación.

```
var chart = nv.models.histogram()
        .color(myColor)
        .width(960)
        .height(500)
        .duration(30);

d3.select('#svgContent svg')
    .datum(data)
    .call(chart);
```

A parte de los datos se establecen otros parámetros como la anchura y altura del gráfico, sus colores, sus datos, el margen que queremos que tenga la página web, etc…

La variable data guarda los datos que cogemos del *selection*.

Por último, todos los gráficos acaban retornando el valor de *chart* que será el gráfico en sí.

4.2 Adaptación de un Chord Diagram de D3 a NVD3

Para la adaptación de un chord diagram a la librería NVD3, he utilizado el código en D3 de la siguiente página.

http://www.delimited.io/blog/2013/12/8/chord-diagrams-in-d3

Dentro de la función *chart* que hemos mencionado anteriormente, procesamos los datos que vamos a graficar con el chord diagram. Recordemos que estos datos estaban guardados en la variable *data* y están organizados en tres columnas: *from, to* y *count*. Los añadimos a la variable *mpr*. Esta variable será la que contenga la matriz de relaciones que es la base del chord diagram.

```
var mpr = chordMpr(data);
    mpr
      .addValuesToMap('from')
      .setFilter(function (row, a, b) {
        return (row.from === a.name && row.to === b.name)
      })
      .setAccessor(function (recs, a, b) {
        if (!recs[0]) return 0;
        return +recs[0].count;
      });
```

Se establecen las medidas del gráfico: alto, ancho y radio. También se establecen de qué color serán las cuerdas del gráfico.

```
var w = width, h = height, r1 = h / 2, r0 = r1 - 100;

var fill = color
```

Estos parámetros se los pasamos desde el archivo HTML cuando hacemos la llamada a la función.

```
var chart = nv.models.drawChords()
    .color(d3.scale.myColors().range())
    .width(960)
    .height(800);

d3.select('#chart1 svg')
    .datum(data)
    .call(chart);

return chart;
```

Añadimos un objeto SVG al cuerpo de la página y agregamos un elemento *circle* con el alto y ancho indicados, y situamos su centro en el medio. Alrededor de este círculo se distribuyen los grupos ordenados según la disposición en la matriz de relaciones.

```
var chord = d3.layout.chord()
    .padding(.02)
    .sortSubgroups(d3.descending)
    .sortChords(d3.descending);

var arc = d3.svg.arc()
    .innerRadius(r0)
    .outerRadius(r0 + 20)

var svg = d3.select("body").append("svg:svg")
    .attr("width", w)
    .attr("height", h)
  .append("svg:g")
    .attr("id", "circle")
    .attr("transform", "translate(" + w / 2 + "," + h / 2 + ")");

    svg.append("circle")
        .attr("r", r0 + 20);
```

Creamos un *tooltip* que nos mostrará información sobre el gráfico cuando situemos el cursor sobre un grupo o sobre una cuerda del gráfico. Añadimos color tanto a los grupos como a las cuerdas. Creamos un texto que se alinea con el arco del grupo e indicará su nombre.

```
var g = svg.selectAll("g.group")
    .data(chord.groups())
  .enter().append("svg:g")
    .attr("class", "group")
    .on("mouseover", mouseover)
    .on("mouseout", function (d) { d3.select("#tooltip").style("visibility", "hidden") });

g.append("svg:path")
    .style("stroke", "black")
    .style("fill", function(d) { return fill(d.index); })
    .attr("d", arc);

g.append("svg:text")
    .each(function(d) { d.angle = (d.startAngle + d.endAngle) / 2; })
    .attr("dy", ".35em")
    .style("font-family", "helvetica, arial, sans-serif")
    .style("font-size", "10px")
    .attr("text-anchor", function(d) { return d.angle > Math.PI ? "end" : null; })
    .attr("transform", function(d) {
      return "rotate(" + (d.angle * 180 / Math.PI - 90) + ")"
          + "translate(" + (r0 + 26) + ")"
          + (d.angle > Math.PI ? "rotate(180)" : "");
    })
    .text(function(d) { return rdr(d).gname; });
```

Las siguientes 3 funciones nos sirven para controlar lo que queremos que el *tooltip* muestre. Si estamos sobre una cuerda, se mostrará el mensaje que contiene chordTip. Mientras que, si estamos sobre un grupo, se mostrará el mensaje que contiene gropupTip. La última función nos sirve para saber dónde está situado el cursor.

```
function chordTip (d) {
    var p = d3.format(".2%"), q = d3.format(",.3r")
    return "Chord Info:<br/>"
      + p(d.svalue/d.stotal) + " (" + q(d.svalue) + ") of "
      + d.sname + " prefer " + d.tname
      + (d.sname === d.tname ? "": ("<br/>while...<br/>"
      + p(d.tvalue/d.ttotal) + " (" + q(d.tvalue) + ") of "
      + d.tname + " prefer " + d.sname))
  }

  function groupTip (d) {
    var p = d3.format(".1%"), q = d3.format(",.3r")
    return "Group Info:<br/>"
        + d.gname + " : " + q(d.gvalue) + "<br/>"
        + p(d.gvalue/d.mtotal) + " of Matrix Total (" + q(d.mtotal) + ")"
  }

  function mouseover(d, i) {
    d3.select("#tooltip")
      .style("visibility", "visible")
      .html(groupTip(rdr(d)))
      .style("top", function () { return (d3.event.pageY - 80)+"px"})
      .style("left", function () { return (d3.event.pageX - 130)+"px";})

    chordPaths.classed("fade", function(p) {
      return p.source.index != i
          && p.target.index != i;
    });
  }
```

4.3 Adaptación de un Petal Diagram de D3 a NVD3

Para la adaptación de un petal diagram a la librería NVD3, he utilizado el código en D3 de la siguiente página.

http://bl.ocks.org/lokesh005/1b23c84b68f5be134ff0

Dentro de la función *chart* que hemos mencionado anteriormente, se establecen los parámetros del gráfico.

```
var radius = Math.min(width-100,height-100)/2;

var arc = d3.svg.arc()
          .outerRadius(radius -230)
          .innerRadius(radius - 50)
          .cornerRadius(20);
var arcOver = d3.svg.arc()
.outerRadius(radius +50)
.innerRadius(0);

var a=width/2 - 20;
var b=height/2 - 90;
```

Creamos un objeto *pie chart* con el alto y ancho indicados y situamos su centro en el medio. También le añadimos un atributo *tooltip* que mostrará información del gráfico.

```
var svg = d3.select("body").append("svg")
          .attr("viewBox", "0 0 " + width + " " + height/2)
          .attr("preserveAspectRatio", "xMidYMid meet")
          .append("g")
          .attr("transform","translate("+a+","+b+")");

          div = d3.select("body")
.append("div")
.attr("class", "tooltip");

var pie = d3.layout.pie()
          .sort(null)
          .value(function(d){return d.value;})
          .padAngle(.02);
```

Por último, le pasamos los datos al *pie* para que genere el gráfico y le damos funcionalidad al *tooltip*. Si el cursor está sobre un pétalo mostrará información. Para finalizar,añadimos color a cada pétalo del gráfico.

```
var g = svg.selectAll(".arc")
        .data(pie(data))
        .enter()
        .append("g")
        .attr("class","arc")
         .on("mousemove",function(d){
            var mouseVal = d3.mouse(this);
            div.style("display","none");
            div
            .html("Item:"+d.data.item+"</br>"+"Valor:"+d.data.value)
            .style("left", (d3.event.pageX+12) + "px")
            .style("top", (d3.event.pageY-10) + "px")
            .style("opacity", 1)
            .style("display","block");
        })
        .on("mouseout",function(){div.html(" ").style("display","none");});



        g.append("path")
        .attr("d",arc)
        .style("fill",function(d){return color(d.data.item);})
         .attr("d", arc);;
```

4.4 Adaptación de un Tree Diagram de D3 a NVD3

Para la adaptación de un tree diagram a la librería NVD3, he utilizado el código en D3 de la siguiente página.

https://bl.ocks.org/d3noob/d316a488cae262ae24e6ca897b209f9e

Dentro de la función *chart* que hemos mencionado anteriormente, creamos un espacio para un árbol y le asignamos un tamaño. También establecemos el nodo raíz.

```
var treemap = d3.tree()
    .size([height, width/2]);

 data.forEach(function(d) {
     if (d.parent == "null") { d.parent = null};
    });
```

Establecemos una jerarquía de nodos a los datos, añadimos nombre a los nodos y asignamos los datos del nodo al diseño del árbol.

```
 treeData.each(function(d) {
     d.name = d.id;
   });

 var nodes = d3.hierarchy(treeData, function(d) {
   return d.children;
   });

 nodes = treemap(nodes);
```

Añadimos un objeto SVG al cuerpo de la página y agregamos un elemento *group* al SVG.

```
var svg = d3.select("body").append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom),
  g = svg.append("g")
    .attr("transform",
      "translate(" + margin.left + "," + margin.top + ")");
```

Añadimos las conexiones entre los nodos.

```
var link = g.selectAll(".link")
  .data( nodes.descendants().slice(1))
  .enter().append("path")
  .attr("class", "link")
  .attr("d", function(d) {
    return "M" + d.y + "," + d.x
    + "C" + (d.y + d.parent.y) / 2 + "," + d.x
    + " " + (d.y + d.parent.y) / 2 + "," + d.parent.x
    + " " + d.parent.y + "," + d.parent.x;
    });
```

Añadimos cada nodo como un grupo, le agregamos un círculo al nodo y por último le ponemos texto al nodo.

```
var node = g.selectAll(".node")
  .data(nodes.descendants())
  .enter().append("g")
  .attr("class", function(d) {
    return "node" +
    (d.children ? " node--internal" : " node--leaf"); })
  .attr("transform", function(d) {
    return "translate(" + d.y + "," + d.x + ")"; });

node.append("circle")
  .attr("r", 10);

node.append("text")
  .attr("dy", ".35em")
  .attr("x", function(d) { return d.children ? -13 : 13; })
  .style("text-anchor", function(d) {
  return d.children ? "end" : "start"; })
  .text(function(d) { return d.data.name; });
```

4.5 Adaptación de un Circle Packing Diagram de D3 a NVD3

Para la adaptación de un circle packing diagram a la librería NVD3, he utilizado el código en D3 de la siguiente página.

http://bl.ocks.org/mbostock/7607535

Dentro de la función *chart* que hemos mencionado anteriormente, añadimos un objeto SVG al cuerpo de la página y situamos su centro en el medio.

```
var svg = d3.select("svg"),
    margin = 20,
    diameter = width,
    g = svg.append("g").attr("transform", "translate(" + diameter / 2 + "," + diameter / 2 + ")");
```

Creamos una escala de colores para que cada nivel de nuestro árbol tenga un color distinto.

```
var color = d3.scaleLinear()
    .domain([-1, 5])
    .range(["hsl(152,80%,80%)", "hsl(228,30%,40%)"])
    .interpolate(d3.interpolateHcl);
```

Guardamos en *root* los datos que habíamos pasado desde nuestro HTML y creamos la jerarquía de nuestro árbol.

```
var root = data;

root = d3.hierarchy(root)
    .sum(function(d) { return d.size; })
    .sort(function(a, b) { return b.value - a.value; });

var focus = root,
    nodes = pack(root).descendants(),
    view;
```

Creamos los círculos de cada nivel por clases, donde diferenciaremos entre nodo raíz, nodo rama y nodo hoja. A cada clase le pertenece un color distinto. Le damos funcionalidad a los círculos del gráfico haciendo zoom cuando hacemos un click sobre ellos. Incorporamos texto a los círculos y solo será visible el texto de los hijos del nodo en el que estemos.

```
var circle = g.selectAll("circle")
  .data(nodes)
  .enter().append("circle")
    .attr("class", function(d) { return d.parent ? d.children ? "node" : "node node--leaf" : "node node--root"; })
    .style("fill", function(d) { return d.children ? color(d.depth) : null; })
    .on("click", function(d) { if (focus !== d) zoom(d), d3.event.stopPropagation(); });

var text = g.selectAll("text")
  .data(nodes)
  .enter().append("text")
    .attr("class", "label")
    .style("fill-opacity", function(d) { return d.parent === root ? 1 : 0; })
    .style("display", function(d) { return d.parent === root ? "inline" : "none"; })
    .text(function(d) { return d.data.name; });
```

La función zoom nos permite cambiar de un nivel a otro dentro del árbol.

```
function zoom(d) {
  var focus0 = focus; focus = d;

  var transition = d3.transition()
      .duration(d3.event.altKey ? 7500 : 750)
      .tween("zoom", function(d) {
        var i = d3.interpolateZoom(view, [focus.x, focus.y, focus.r * 2 + margin]);
        return function(t) { zoomTo(i(t)); };
      });

  transition.selectAll("text")
    .filter(function(d) { return d.parent === focus || this.style.display === "inline"; })
      .style("fill-opacity", function(d) { return d.parent === focus ? 1 : 0; })
      .on("start", function(d) { if (d.parent === focus) this.style.display = "inline"; })
      .on("end", function(d) { if (d.parent !== focus) this.style.display = "none"; });
}
```

La función zoomTo establece la visión del gráfico inicial.

```
function zoomTo(v) {
  var k = diameter / v[2]; view = v;
  node.attr("transform", function(d) { return "translate(" + (d.x - v[0]) * k + "," + (d.y - v[1]) * k + ")"; });
  circle.attr("r", function(d) { return d.r * k; });
}
```

4.6 Adaptación de un Histograma de D3 a NVD3

Para la adaptación de un histograma a la librería NVD3, he utilizado el código en D3 de la siguiente página.

https://gist.github.com/nnattawat/8916402

Dentro de la función *chart* que hemos mencionado anteriormente, se establecen los parámetros del gráfico. Dentro de *values* guardamos los datos que habíamos pasado desde el documento HTML. En *number* guardamos el número de divisiones del histograma, establecemos las medidas de ancho y largo, y por último creamos un dominio con el valor mínimo y máximo.

```
var values = data;

var number = duration;

width = width - margin.left - margin.right;
height = height - margin.top - margin.bottom;

var max = d3.max(values);
var min = d3.min(values);
var x = d3.scale.linear()
      .domain([min, max])
      .range([0, width]);
```

Creamos un formato para los números del eje y establecemos particiones equivalentes para cada barra del histograma. Establecemos un máximo y mínimo en el eje vertical y creamos un dominio con el valor mínimo y máximo. Por último creamos un eje numérico con el rango de x.

```
var formatCount = d3.format(",.0f");

var dat = d3.layout.histogram()
    .bins(x.ticks(number))
    (values);

var yMax = d3.max(dat, function(d){return d.length});
var yMin = d3.min(dat, function(d){return d.length});

var y = d3.scale.linear()
    .domain([0, yMax])
    .range([height, 0]);

var xAxis = d3.svg.axis()
    .scale(x);
```

Añadimos un objeto SVG al cuerpo de la página y dentro de él añadimos las barras del histograma y añadimos un campo texto a cada barra indicando el valor. Por último añadimos el eje horizontal al gráfico.

```
var svg = d3.select("body").append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
  .append("g")
    .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

var bar = svg.selectAll(".bar")
    .data(dat)
  .enter().append("g")
    .attr("class", "bar")
    .attr("transform", function(d) { return "translate(" + x(d.x) + "," + y(d.y) + ")"; });

bar.append("rect")
    .attr("x", 1)
    .attr("width", (x(dat[0].dx) - x(0)) - 1)
    .attr("height", function(d) { return height - y(d.y); })
    .attr("fill", color);

bar.append("text")
    .attr("dy", ".75em")
    .attr("y", -12)
    .attr("x", (x(dat[0].dx) - x(0)) / 2)
    .attr("text-anchor", "middle")
    .text(function(d) { return formatCount(d.y); });

svg.append("g")
    .attr("class", "axis axis--x")
    .attr("transform", "translate(0," + height + ")")
    .call(d3.axisBottom(x));
```

5. Conclusiones

Para concluir, queremos destacar algunos puntos del trabajo que nos han llamado la atención.

En primer lugar, a pesar de haber realizado una amplia búsqueda bibliográfica, hemos podido observar que la documentación que existe en la actualidad sobre NVD3 es más bien escasa. En este caso no ha supuesto un problema a destacar, aunque ha retrasado la adaptación del código del primer ejemplo. En los siguientes ejemplos, la adaptación ha sido más sencilla ya que partimos del primer ejemplo como guía para no cometer algunos errores que han surgido

En segundo lugar, a nivel personal ha supuesto la suma de unos conocimientos en concreto que podemos valorar de forma positiva. Entre ellos, la utilización de dos librerías que eran desconocidas para mí hasta ahora y nuevos tipos de gráficos como pueden ser el chord diagram o el circle packaging diagram.

Por último queremos destacar que el trabajo realizado podrá servir de guía tanto para los usuarios que quieran utilizar los ejemplos realizados, como para los programadores que quieran crear sus ejemplos propios.

6. Bibliografía

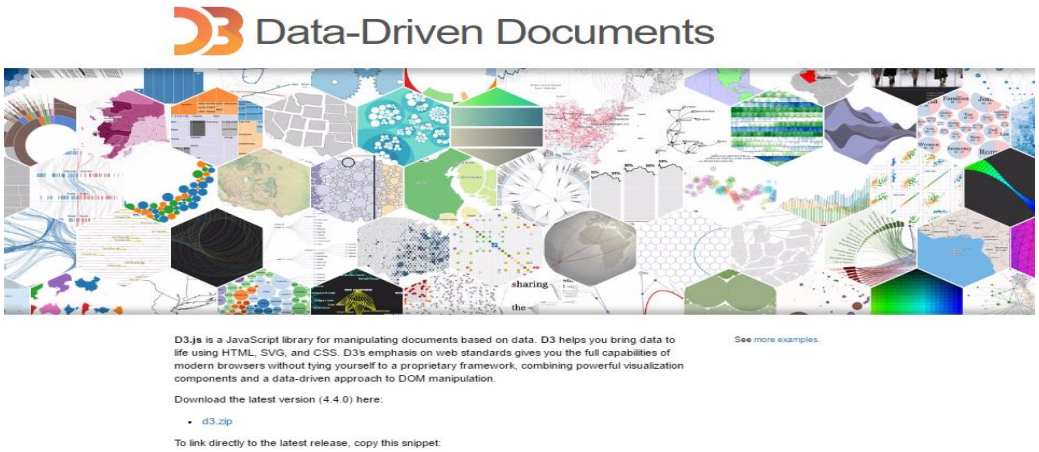
- Murray, S. (2012). *Interactive Data Visualization for the Web.* CA: O'REILLY.
- Delimited Technologies. (2014). "Chrord Diagrams in D3". Recuperado de http://www.delimited.io/blog/2013/12/8/chord-diagrams-in-d3
- Mike Boston. (2015). "D3*".* Recuperado de https://d3js.org/
- Mazza, R. (2009). *Introduction to Information Visualization.* Londres: Springer.
- Novus Partners. (2014). "NVD3 Re-usable charts for d3.js". Recuperado de http://nvd3.org/

7. Apéndice A: Manual de usuario

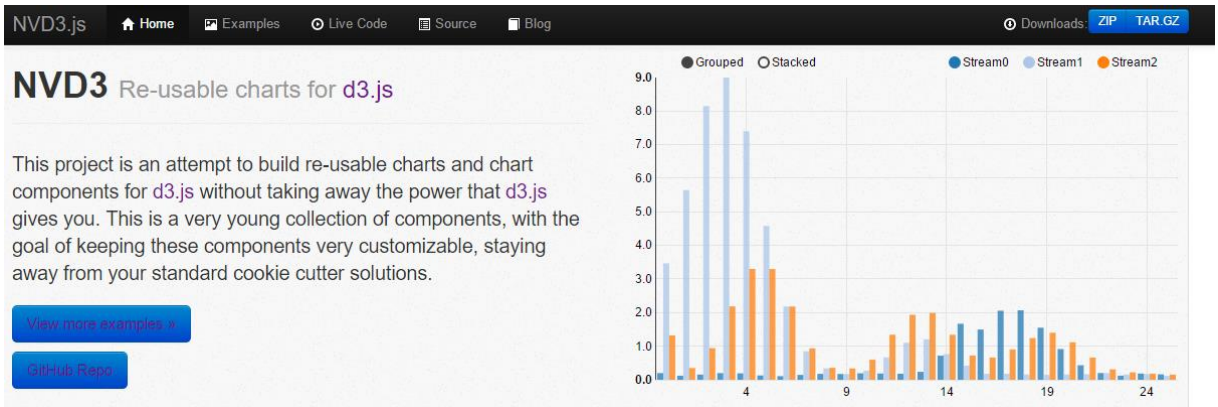
7.1 Instalación de las librerías D3 y NVD3

El primer paso para poder generar un gráfico utilizando la librería NVD3 es bajarse las librerías D3 y NVD3. Contienen las funciones necesarias para el uso de grafos en D3 y las funciones de creación de los gráficos reutilizables.

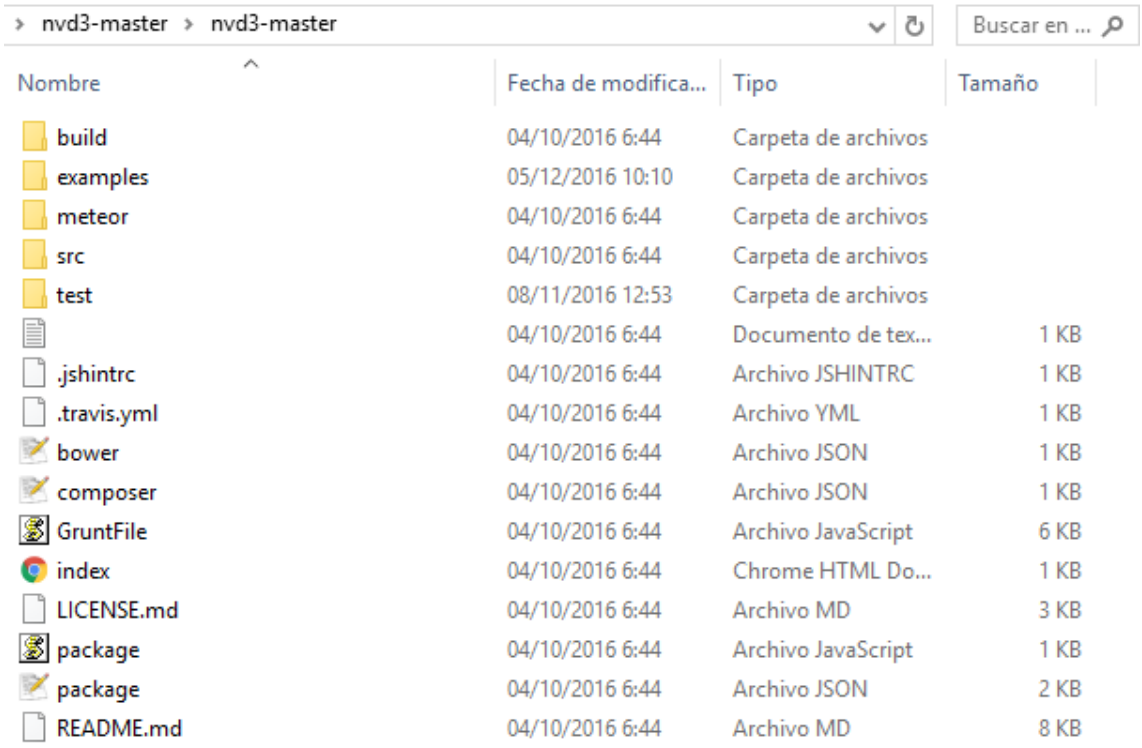
Para bajarnos la librería D3 vamos a su página inicial (https://d3js.org/) y nos descargamos la última versión de D3 (ver imagen).



A continuación, nos descargamos NVD3. Para ello, vamos a su página inicial (http://nvd3.org/) y nos descargamos su última versión (ver imagen).

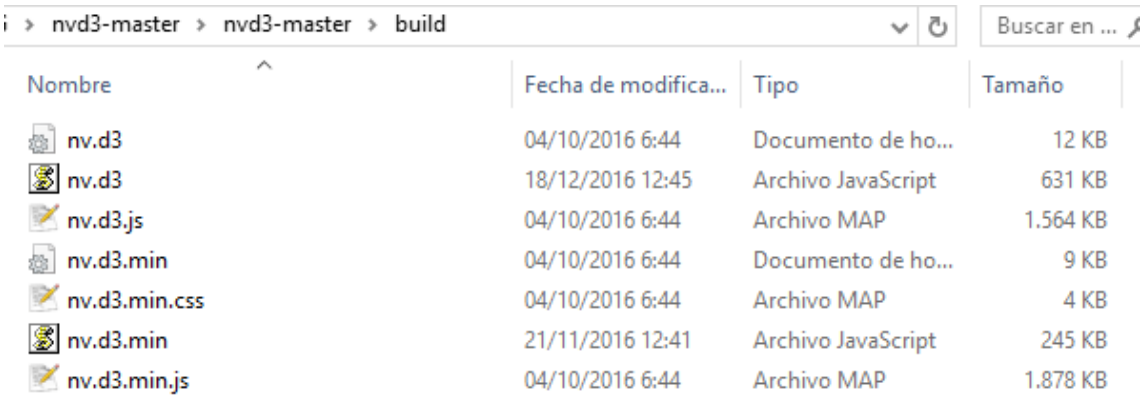


Una vez descargadas las librerías, nuestro directorio quedará de esta manera:



Dentro de la carpeta nvd3-master tenemos 5 carpetas y varios documentos. De todos ellos nos centraremos en 2 carpetas: la carpeta build y la carpeta examples.

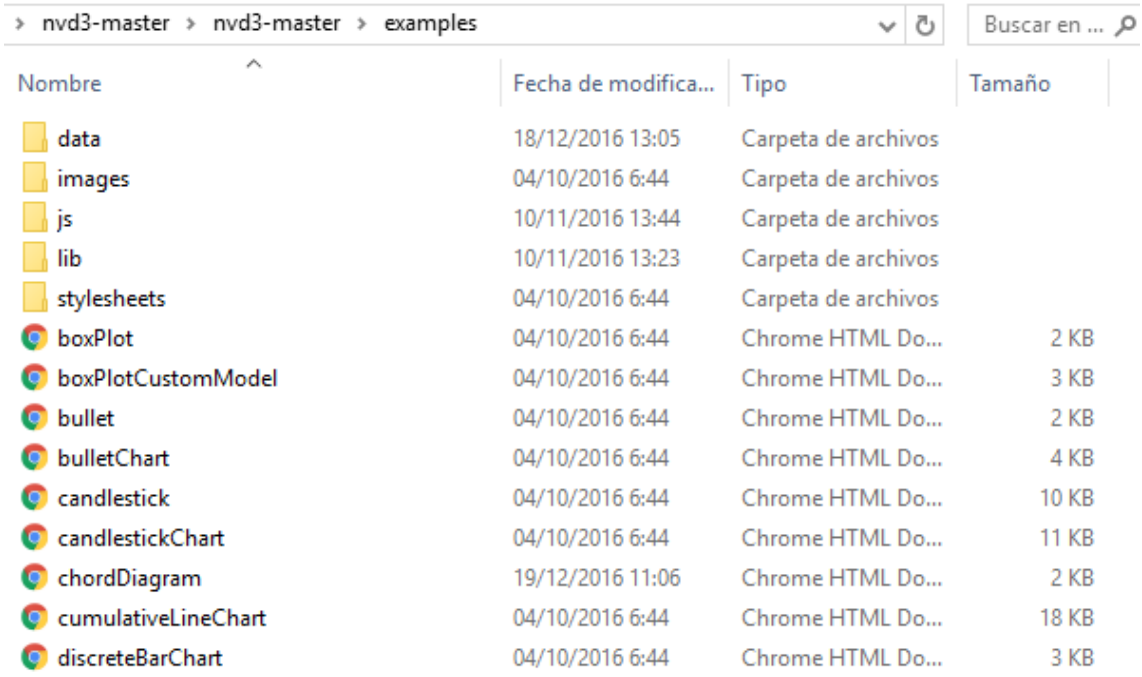
Dentro de la carpeta build tenemos una serie de documentos, de todos ellos el más importante es nv.d3.js. Es un documento escrito en JavaScript que contiene todas las funciones de gráficos reutilizables.



Para llamar a estas funciones necesitamos crear un archivo HTML que lo haga.

Este paso lo veremos a continuación con varios ejemplos.

Dentro de la otra carpeta tenemos todos los ejemplos creados en NVD3 para poder visualizarlos a través de un navegador o simplemente inspeccionar su código. Además encontramos 5 carpetas que contienen entre otros archivos: datos, librerías auxiliares, imágenes, etc…



7.2 Creación de un Chord Diagram

Una vez descargadas las librerías necesitamos crear un archivo CSV que contenga los datos que nos permita generar el gráfico.

El archivo CSV contiene 3 columnas de 1 elemento separados por comas. La primera fila de la columna tiene que contener estos 3 elementos: **from,** de donde salen las cuerdas del chord diagram, **to,** a donde llegan las cuerdas y **count,** el número total de cuerdas que van de un lado al otro.

Por ejemplo:

```
"from","to","count"
"black","blue",2868
"green","brown",2060
"brown","green",16145
"blue","black",1013
"black","brown",8916
"green","green",10048
"brown","black",8010
"blue","blue",6907
"black","green",5871
"green","black",1951
"brown","blue",8045
"blue","brown",940
"black","black",11975
"green","blue",6171
"brown","brown",8090
"blue","green",990
"honey","green",990
"honey","blue",990
"honey","black",990
"honey","brown",990
"honey","honey",1990
"green","honey",790
"blue","honey",790
"black","honey",790
"brown","honey",790
```

En este ejemplo, el gráfico tendrá 5 arcos del círculo denominados Honey, Black, Brown, Green y Blue. Las cuerdas que unen los arcos del círculo serán los elementos numéricos de la tercera columna.

El último paso para crear nuestro gráfico será el de crear un archivo HTML que hará la llamada a la librería NVD3.

Un ejemplo de archivo HTML podría ser:

```
<body>

<script src="lib/underscore.js"></script>

<div id="tooltip"></div>

<div id="chart1">
  <svg></svg>
</div>

  <script>

  var myColors = ["#000000", "#37EA17", "#957244", "#228DF1", "#F7B125"];
  d3.scale.myColors = function() {
      return d3.scale.ordinal().range(myColors);
  };

  d3.csv('data/ojos.csv', function (error, data) {
        nv.addGraph(function() {
          var chart = nv.models.drawChords()
              .color(d3.scale.myColors().range())
              .width(960)
              .height(800);

          d3.select('#chart1 svg')
              .datum(data)
              .call(chart);

          return chart;
      });
    });
  </script>
</body>
```

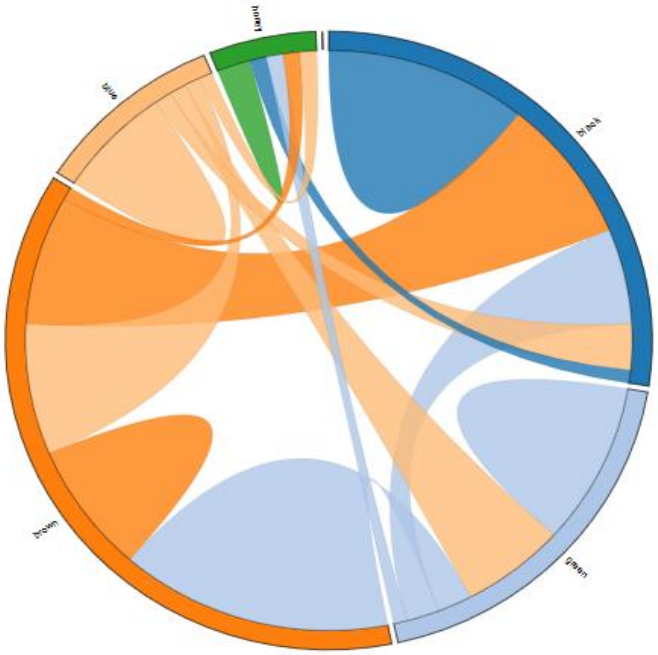
En él, creamos un espacio (un elemento div en nuestra página HTML) para un gráfico SVG y a ese espacio lo llamamos chart1. También creamos otro espacio para un *tooltip,* una ventana emergente que nos mostrará información de la relación entre los elementos del gráfico.

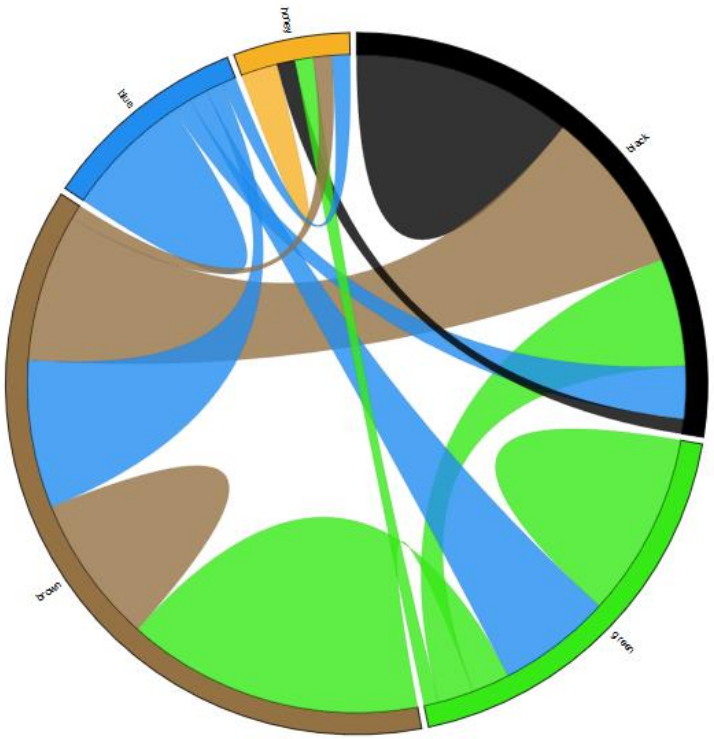
De manera opcional se puede crear un listado de colores para poder personalizar nuestro diagrama, sino, tendrá unos colores predeterminados. También podemos establecer las dimensiones de nuestro gráfico.

A continuación cargamos los datos que habíamos guardado en el documento CSV y por último, llamamos a la función de la librería NVD3 drawChords que es la que creará el gráfico y lo guardará en la variable chart1 que habíamos creado.

El último paso es abrir nuestro documento HTML desde el navegador para ver que todo ha ido bien.

A continuación vemos 2 ejemplos de chrods diagrams: el primero con los colores predeterminados y el segundo haciendo una selección de los mismos.





7.3 Creación de un Petal Diagram

Una vez descargadas las librerías necesitamos crear un archivo CSV que contenga los datos que nos permita generar el gráfico.

El archivo CSV contiene 2 columnas de 1 elemento separados por comas. La primera fila de la columna ha que contener estos 2 elementos: **ítem,** el nombre del pétalo y **value,** el valor asociado al pétalo.

Por ejemplo:

```
"item","value"
"Telecinco",14.4
"Antena 3",13.3
"La 1",10.5
"Autonomicas",7.4
"La Sexta",7.2
"Television de pago",6.8
"Cuatro",6.1
"FDF",3.3
"Neox",2.5
"La 2",2.5
"Nova",2.4
"Divinity",2.2
"13 TV",2.1
"Clan",2
"Energy",1.9
"D-Max",1.8
"Mega",1.7
"Paramount Channel",1.7
"Boing",1.4
"Disney Channel",1.1
"Atreseries",0.9
"TVE 24H",0.8
"DKISS",0.8
"Autonomicas Privadas",0.7
"BeMad tv",0.6
"Gol tv",0.5
"Ten",0.5,
"Teledeporte",0.4
```

En este ejemplo, el gráfico tendrá 28 pétalos que se corresponde con los nombres de la primera columna del archivo. El tamaño de cada pétalo depende del valor asociado al nombre.

El último paso para crear nuestro gráfico será el de crear un archivo HTML que hará la llamada a la librería NVD3.

Un ejemplo de archivo HTML podría ser:

```
<body>

  <div id="svgContent">
    <svg></svg>
  </div>

<script>

    var myColors = ["#3344ff", " #ffc533", " #0708c3", " #afafaf", "  #01a228", "  #000000",
    d3.scale.myColors = function() {
        return d3.scale.ordinal().range(myColors);
    };

    d3.csv('data/barometro.csv', function (error, data) {
        nv.addGraph(function() {
            var chart = nv.models.platelets()
            .color(d3.scale.myColors().range())
            .width(300)
            .height(300);

            d3.select('#svgContent svg')
                .datum(data)
                .call(chart);

            return chart;
        });
    });

</script>
</body>
```

En él, creamos un espacio para un gráfico SVG y lo llamamos svgContent.

De manera opcional, se puede crear un listado de colores para poder personalizar nuestro diagrama, sino, tendrá unos colores predeterminados. Además es posible personalizar el tamaño del gráfico.

Cargamos los datos que hemos guardado en el documento CSV y por último, llamamos a la función de la librería NVD3 petal que es la que creará el gráfico y lo guardará en la variable svgContent que habíamos creado antes.

El último paso es abrir nuestro documento HTML desde el navegador para ver que todo ha ido bien.

A continuación vemos 2 ejemplos de petal diagrams: el primero con los colores predeterminados y el segundo haciendo una selección de los mismos.

7.4 Creación de un Tree Diagram

Una vez descargadas las librerías necesitamos un archivo CSV que contenga los datos que nos permita generar el gráfico.

El archivo CSV contiene 2 columnas de 1 elemento separados por comas. La primera fila de la columna tiene que contener estos 2 elementos: **name,** el nombre del nodo y **parent,** el nombre de su padre.

Por ejemplo:

```
"name","parent"
"Figuras Geometricas Planas","null"
"3 lados","Figuras Geometricas Planas"
"Triangulo","3 lados"
"Equilatero","Triangulo"
"Todos sus lados poseen la misma longitud y todos sus angulos
son iguales.","Equilatero"
"Isosceles","Triangulo"
"Dos de sus lados poseen la misma longitud y dos de sus
angulos son iguales.","Isosceles"
"Escaleno","Triangulo"
"Ninguno de sus lados poseen la misma longitud y ninguno de
sus angulos son iguales.","Escaleno"
"4 lados","Figuras Geometricas Planas"
"Cuadrado","4 lados"
"Todos sus lados poseen la misma longitud y todos sus angulos
hacen 90 grados","Cuadrado"
"Rectangulo","4 lados"
"Sus lados opuestos son iguales dos a dos y todos sus angulos
hacen 90 grados","Rectangulo"
"Rombo","4 lados"
"Todos sus lados poseen la misma longitud y cada par de
angulos agudos y obtusos son opuestos","Rombo"
"Romboide","4 lados"
"Sus lados opuestos son iguales dos a dos y cada par de
angulos agudos y obtusos son opuestos","Romboide"
"5 lados","Figuras Geometricas Planas"
"Pentagono","5 lados"
"Todos sus lados poseen la misma longitud y todos sus angulos
son iguales.","Pentagono"
"6 lados","Figuras Geometricas Planas"
"Hexagono","6 lados"
"Todos sus lados poseen la misma longitud y todos sus angulos
son iguales.","Hexagono"
```

El último paso para crear nuestro gráfico será el de crear un archivo HTML que hará la llamada a la librería NVD3.

Un ejemplo de archivo HTML podría ser:

```
<body>

<div id="svgContent">
  <svg></svg>
</div>

  <!-- load the d3.js library -->
  <script src="http://d3js.org/d3.v4.min.js"></script>

  <script>

  d3.csv('data/figuras.csv', function (error, data) {
      nv.addGraph(function() {
              var chart = nv.models.treeDiagram()
                  .width(1600)
                  .height(500);

              d3.select('#svgContent svg')
                  .datum(data)
                  .call(chart);

              return chart;
      });
  });

  </script>

</body>
```

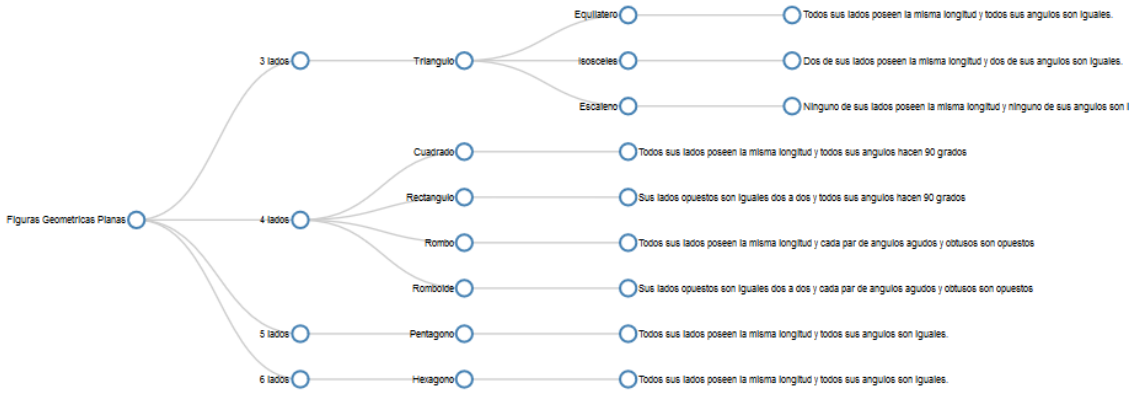
En él, creamos un espacio para un gráfico SVG y lo llamamos svgContent.

Podemos establecer de manera opcional el tamaño de nuestro gráfico, sino lo hacemos, tendrá un tamaño prestablecido.

Después cargamos los datos que hemos guardado en el documento CSV y por último, llamamos a la función de la librería NVD3 treeDiagram que es la que creará el gráfico y lo guardará en la variable svgContent que habíamos creado anteriormente.

El último paso es abrir nuestro documento HTML desde el navegador para ver que todo ha ido bien.

A continuación podemos ver un ejemplo de tree diagram.



7.5 Creación de un Circle Packing Diagram

Una vez descargadas las librerías necesitamos crear un archivo JSON que contenga los datos que nos permita generar el gráfico. Estos datos serán los encargados de crear los nodos. Necesitamos indicar el nombre del nodo y los hijos del mismo. Si un nodo no tiene hijo, habrá que indicar es su tamaño.

Por ejemplo:

```
{
 "name": "Ligas Europeas",
 "children": [
  {
   "name": "Liga Espanola",
   "children": [
    {
     "name": "Primera Division",
     "children": [
      {"name": "F.C Barcelona", "size": 160},
      {"name": "Real Madrid C.F", "size": 156.6},
      {"name": "Valencia C.F", "size": 48},
      {"name": "Atletico de Madrid", "size": 41.6},
       {"name": "Villarreal C.F", "size": 34},
      {"name": "Sevilla C.F", "size": 32.5},
      {"name": "Athletic Club", "size": 32},
      {"name": "Real Sociedad", "size": 25.1},
       {"name": "Levante U.D", "size": 22.5},
      {"name": "R.C.D. Espanyol", "size": 22.4},
      {"name": "Malaga C.F", "size": 22},
      {"name": "Celta de Vigo", "size": 21.7},
       {"name": "Getafe C.F", "size": 21},
      {"name": "Elche C.F", "size": 18.6},
      {"name": "Granada C.F", "size": 18.5},
       {"name": "Rayo Vallecano", "size": 18.1},
      {"name": "R.C.D. de la Coruna", "size": 18},
      {"name": "Cordoba C.F", "size": 17.1},
      {"name": "U.D Almeria", "size": 16.5},
       {"name": "S.D Eibar", "size": 13.8}
     ]
    }
   ]
  },
```

El último paso para crear nuestro gráfico será el de crear un archivo HTML que hará la llamada a la librería NVD3.

Un ejemplo de archivo HTML podría ser:

```
<body>

<div id="svgContent">
    <svg width="960" height="960"></svg>
  </div>

  <script src="https://d3js.org/d3.v4.min.js"></script>
<script>

    d3.json('data/flare.json', function(error, data) {
        nv.addGraph(function() {
                var chart = nv.models.zoomableCirclePacking()
                    .width(960)
                    .height(960);

                d3.select('#svgContent svg')
                    .datum(data)
                    .call(chart);

                return chart;
        });
    });

</script>

</body>
```

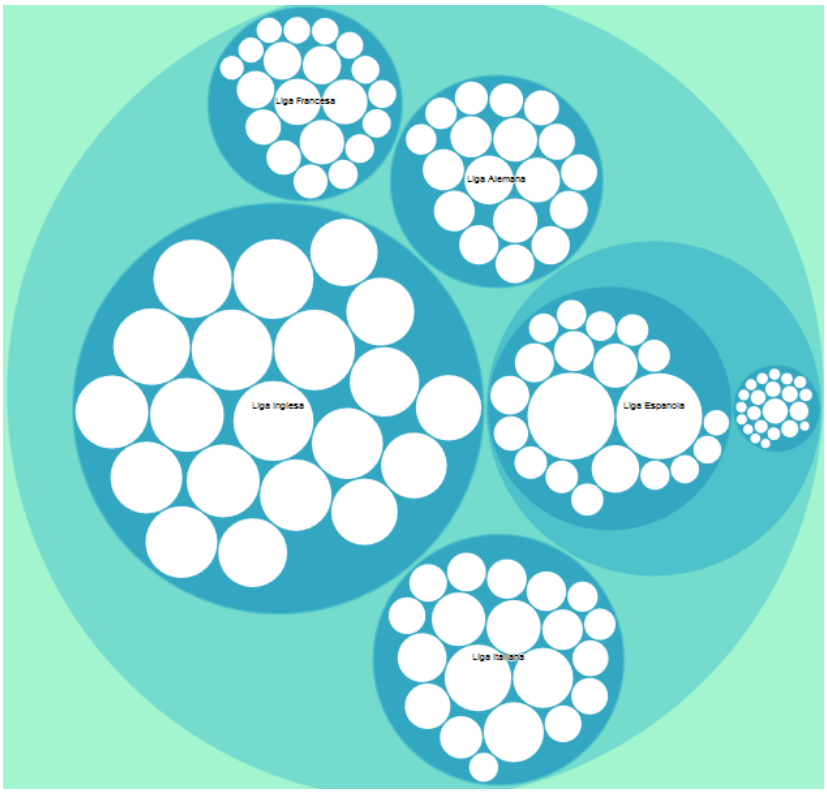
En él, creamos un espacio para un gráfico SVG y lo llamamos svgContent.

Podemos establecer de manera opcional el tamaño de nuestro gráfico, sino lo hacemos tendrá un tamaña prestablecido. Para ello, lo indicaremos en la función y en el SVG.

Después cargamos los datos que habíamos guardado en el documento JSON y por último, llamamos a la función de la librería NVD3 zoomableCirclePacking que es la que creará el gráfico y lo guardará en la variable svgContent que habíamos creado antes.

El último paso es abrir nuestro documento HTML desde el navegador para ver que todo ha ido bien.

A continuación podemos ver un ejemplo de circle packing diagram: en la primera imagen podemos vemos el gráfico completo y en la segunda, vemos una ampliación de la anterior.



7.6 Creación de un Histograma

Una vez descargadas las librerías nos faltaría crear un archivo HTML que hará la llamada a la librería NVD3 para crear nuestro gráfico.

Un ejemplo de archivo HTML podría ser:

```
<body>

<div id="svgContent">
  <svg></svg>
</div>
  <!-- load the d3.js library -->
  <script src="http://d3js.org/d3.v4.min.js"></script>

  <script>

  var data = d3.range(1000).map(d3.random.normal(20, 5));

  var myColor = "steelblue";

  nv.addGraph(function() {
      var chart = nv.models.histogram()
              .color(myColor)
              .width(960)
              .height(500)
              .duration(30);

      d3.select('#svgContent svg')
          .datum(data)
          .call(chart);

      return chart;
  });
  </script>
</body>
```

En él, creamos un espacio para un gráfico SVG y lo llamamos svgContent.

Después generamos los datos esta vez de forma aleatoria.

De manera opcional se puede crear un color que nos permitirá personalizar nuestro diagrama, sino se crea, nuestro diagrama tendrá un color predeterminado. También podemos personalizar las medidas de nuestro gráfico y el número de barras que queremos que tenga.

Por último, llamamos a la función de la librería NVD3 histogram que es la que creará el gráfico y lo guardará en la variable svgContent que habíamos creado anteriormente.

El último paso es abrir nuestro documento HTML desde el navegador para ver que todo ha ido bien.

A continuación podemos ver dos ejemplos de histograma: en el primero, podemos ver el histograma sin personalizar y en el segundo, el histograma personalizado con el color, el tamaño y el número de barras que hemos elegido.