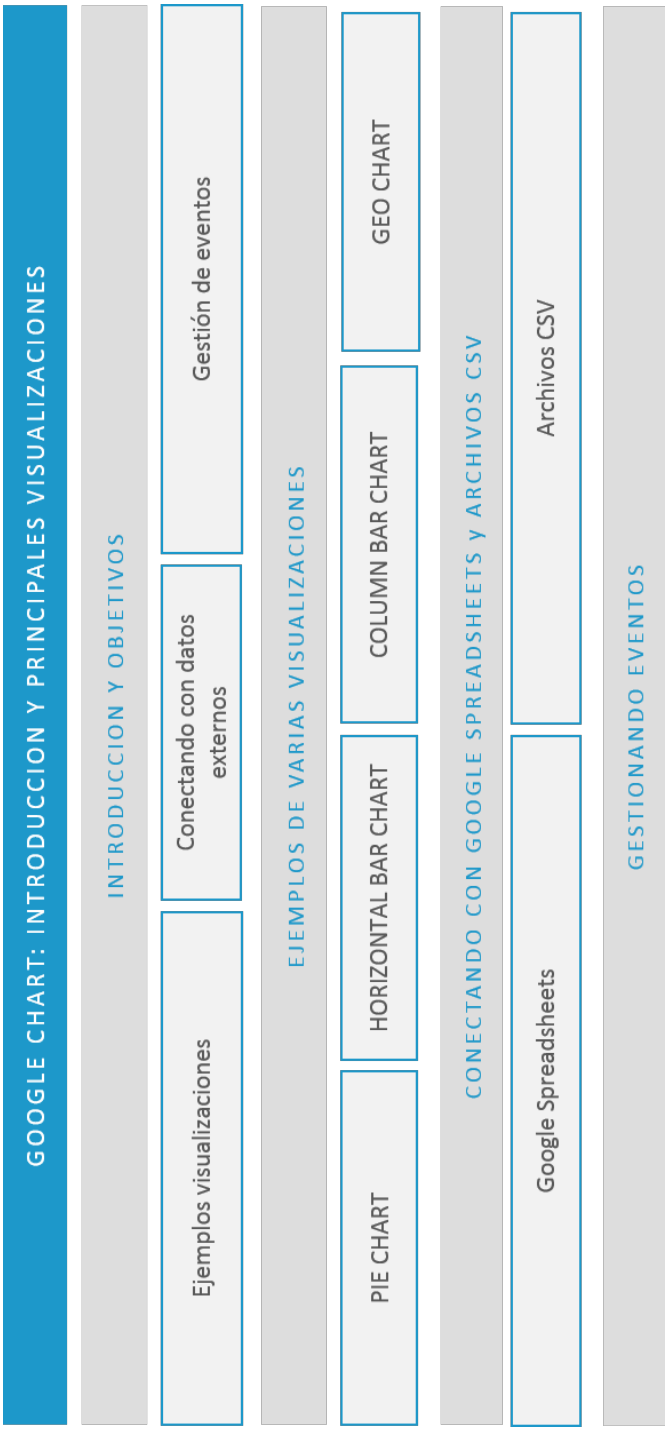
Herramientas de Visualización

Tema 2. Google Chart. Introducción y principales visualizaciones

Índice

Esquema



2.1. Introducción y objetivos

Google Chart Library es la librería de visualizaciones utilizada por Google Spreadsheets. Google utiliza esta funcionalidad en la hoja de cálculo que han implementado y esto nos permite infinitas posibilidades donde estas librerías son una pequeña funcionalidad.

Piensa en *real-time analytics*, potentes algoritmos procesando los datos en el *back-end*, visualizaciones de la actividad en Twitter con los contenidos de los *tweets* en sí..., un gran abanico de posibilidades donde, como expertos en *analytics*, pondréis el límite a las aplicaciones de la empresa con la que colabores. Al fin y al cabo, las visualizaciones son la representación alternativa de la información.

Empezaremos con una simple frase sobre Google y sus funcionalidades, entre las que incluiremos Google Chart Library:

> «Google has the functionality of a really complicated Swiss Army knife, but the home page is our way of approaching it closed. It's simple, it's elegant. You can slip it in your pocket, but it's got the great doodad when you need it». (Tischler, 2005).

A partir de aquí empiezan las *hand-on sessions,* practicar y programar será nuestra premisa. Si encuentras dificultades en seguir JavaScript, no dudes en volver al tema anterior y dedicar un poco de tiempo al enlace que incluye el tutorial de JavaScript. No nos asustemos, siguiendo y repitiendo los pasos que indicamos a lo largo del tema, se puede aprender fácilmente cómo trabajar con las visualizaciones.

Nos centraremos en los siguientes aspectos:

- Ejemplos de **visualizaciones**: aprenderemos a través de ejemplos existentes en Internet.
- Conectar con **datos externos**: para que no tengamos que modificar el código cada vez que queramos cambiar los valores.

- **Eventos**: para poder conectar nuestras visualizaciones con otras herramientas.

2.2. Ejemplos de varias visualizaciones

Pie chart

Empezaremos por un *pie chart* y con un ejemplo muy fácil que podemos encontrar en Quickstart de Google:

https://developers.google.com/chart/interactive/docs/quick_start

Ejemplo: código completo de un *pie chart*

```
<html>
<head>
<script type="text/javascript"
src="https://www.gstatic.com/charts/loader.js">
</script>
<script type="text/javascript">
// Load Charts and the corechart package.
google.charts.load('current', {'packages':['corechart']});
//Draw the pie chart for Sarah's pizza when Charts is
loaded.
google.charts.setOnLoadCallback(drawChart);
// Callback that draws the pie chart for Sarah's pizza.
function drawChart() {
// Create the data table for Sarah's pizza.
var data = new google.visualization.DataTable();
data.addColumn('string', 'Topping');
data.addColumn('number', 'Slices');
data.addRows([
['Mushrooms', 1],
['Onions', 1],
['Olives', 2],
['Zucchini', 2],
['Pepperoni', 1]
]);
// Set options for Sarah's pie chart.
var options = {title:'How Much Pizza Sarah Ate Last
Night',
width:400,
height:300};
```

```
// Instantiate and draw the chart for Sarah's pizza.
var chart = new
google.visualization.PieChart(document.getElementById
('Sarah_chart_div'));
chart.draw(data, options);
}
</script>
</head>
<body>
<!--Table and divs that hold the pie charts-->
<table class="columns">
<tr>
<td>
<div id="Sarah_chart_div" style="border: 1px solid #ccc">
</div></td>
</tr>
</table>
</body>
</html>
```

Este código lo puedes copiar en un fichero de texto, guardarlo con la terminación «.html» y abrirlo con un navegador o con el editor de código Brackets que vimos en el tema anterior. Vamos a separarlo por partes para entenderlo mejor.

Parte exclusiva de HTML

```
< html>
<head>
</head>
<body>
<!--Table and divs that hold the pie charts-->
<table class="columns">
<tr>
<td><div id="Sarah_chart_div" style="border: 1px solid #ccc">
</div></td>
</tr>
</table>
</body>
</html>
```

Esta es la parte exclusiva de HTML. Aquí te tienes que fijar en que hemos creado una sección dentro del HTML que tiene el identificador . Aquí es donde cargaremos nuestra gráfica.

¿Cómo cargamos la librería de Google Chart a nuestro HTML?

```
<script src="https://www.gstatic.com/charts/loader.js"></script>
<script>
google.charts.load('current', {packages: ['corechart']});
google.charts.setOnLoadCallback(drawChart);
...
</script>
```

La primera línea de este ejemplo se ocupa de cargar el propio «cargador» o de la librería. Esta sentencia solo debe ejecutarse una vez, independientemente de cuántas gráficas vayamos a pintar.

Lo que sí debemos hacer es cargar aquella librería asociada a las gráficas que queramos utilizar. En nuestro ejemplo cargamos la librería , que incluye los paquetes de gráficas tipo *bar, column, line, area, stepped area, bubble, pie, donut, combo, candlestick, histogram* y *scatter*. Los diferentes tipos de librerías y las gráficas que incluyen se encuentran en la sección Loading de cada tipo de gráfica.

Puedes acceder a la galería de gráficas a través del siguiente enlace:
https://developers.google.com/chart/interactive/docs/gallery

Podemos fijarnos en el detalle de que *packages* es plural y está definido con un *array*, por lo que podríamos cargar más de un paquete a la vez. Ejemplo:

```
google.charts.load('current', {packages:
['corechart', 'table', 'sankey']});
```

Por último, tenemos la siguiente línea que llama a la función solo cuando se termine de cargar los paquetes de la librería que hemos mencionado anteriormente.

JavaScript se ejecuta de forma secuencial y esta línea evita que se ejecute la función que pinta la gráfica sin estar cargada la librería, lo que daría un error. Además, en ese momento, el HTML está ya completamente cargado en el explorador.

```
// Draw the pie chart for Sarah's pizza when Charts is loaded.
google.charts.setOnLoadCallback(drawChart);
```

¿Qué es lo que hace nuestra función drawChart?

Primero hemos de crear nuestros datos.

```
// Create the data table for Sarah's pizza.
var data = new google.visualization.DataTable();
data.addColumn('string', 'Topping');
data.addColumn('number', 'Slices');
data.addRows([
['Mushrooms', 1],
['Onions', 1],
['Olives', 2],
['Zucchini', 2],
['Pepperoni', 1]
]);
```

es una estructura de datos con la que trabaja JavaScript, no muy diferente a lo que podría ser un *array* de *arrays* en Java o C.

Añadimos dos columnas asignándoles un nombre de cabecera que la librería puede utilizar en caso de que sea necesario. Además, indicamos el tipo de valores que contienen las celdas.

Finalmente, añadimos las filas con los valores que contengan. En este caso, añadimos todas las filas de una vez, pero también podríamos añadir las filas una por una. Esto puede ser útil si queremos hacer nuestra **visualización dinámica**, es decir, que cambie basándose en eventos.

```
data.addRow(['Pepperoni', 2]);
```

Después se definen las opciones del gráfico como **título y dimensiones**:

```
// Set options for Sarah's pie chart.
var options = {title:'How Much Pizza Sarah Ate Last Night',
width:400,
height:300};
```

Aquí debemos mirar las opciones que cada *chart* nos ofrece. Por ejemplo, los *pie charts* ofrecen más opciones. Probad estas dos variantes:

```
var options = {
width: 400,
height: 240,
title: 'Toppings I Like On My Pizza',
colors: ['#e0440e', '#e6693e', '#ec8f6e', '#f3b49f', '#f6c7b6']
};
```

Y el otro ejemplo:

```
var options = {
width: 400,
height: 240,
title: 'Toppings I Like On My Pizza',
colors: ['#e0440e', '#e6693e', '#ec8f6e', '#f3b49f', '#f6c7b6'],
is3D: true
};
```

Estos dos ejemplos modifican los colores y las dimensiones. Recuerda la teoría de visualizaciones antes de aplicar efectos y colores; siempre es más atractivo a la vista

aplicar efectos 3D y usar diferentes colores o gradientes de colores, sin embargo, puede jugar en contra de vuestras visualizaciones. No todo el mundo detecta los mismos gradientes ni percibe de la misma manera las dimensiones.

Finalmente, nos encontramos el código que dibuja nuestro gráfico.

```
// Instantiate and draw the chart for Sarah's pizza.
var chart = new
google.visualization.PieChart(document.getElementById
 ('Sarah_chart_div'));
chart.draw(data, options);
```

Instanciamos primero nuestro gráfico, en este caso un *pie chart*. Además, indicamos en la instancia dónde debe ser dibujado.

Cuando iniciamos la acción de dibujar, , le pasamos los valores data que queremos visualizar y las opciones que hemos definido. Con estos sencillos pasos, podemos generar el siguiente gráfico:

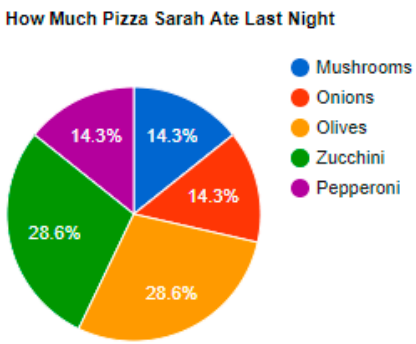


Figura 1. *Pie Chart*. Fuente: https://developers.google.com/chart/interactive/docs/quick_start

A continuación vamos a explicar las demás visualizaciones a partir de las diferencias respecto a la que hemos explicado. El cambio de gráfica es muy sencillo. Basta con especificar el tipo de gráfica deseada.

Horizontal bar chart

Especificamos en el código el tipo de gráfica :

```
var chart = new
google.visualization.BarChart(document.getElementById
('Sarah_chart_div'));
```

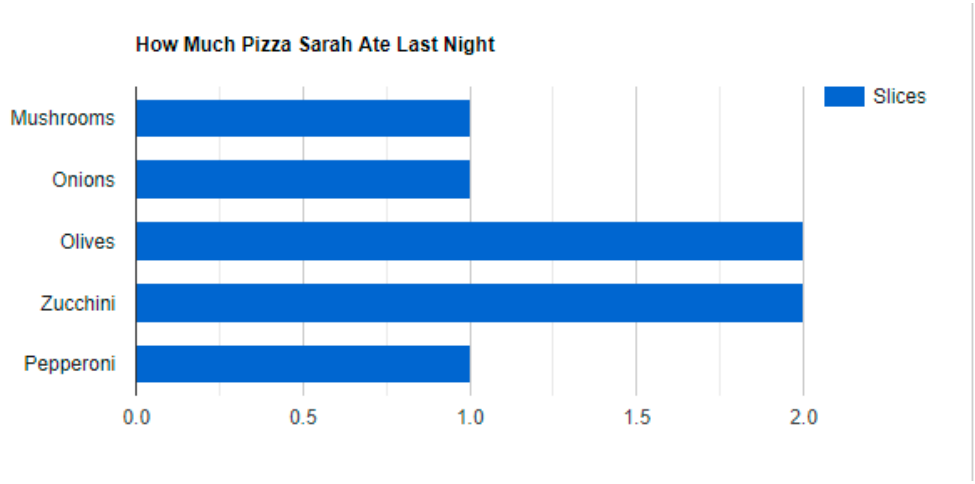


Figura 2. *Horizontal bar chart.*

Column bar chart

Especificamos en el código el tipo de gráfica :

```
var chart = new
google.visualization.ColumnChart(document.getElementById
('Sarah_chart_div'));
```

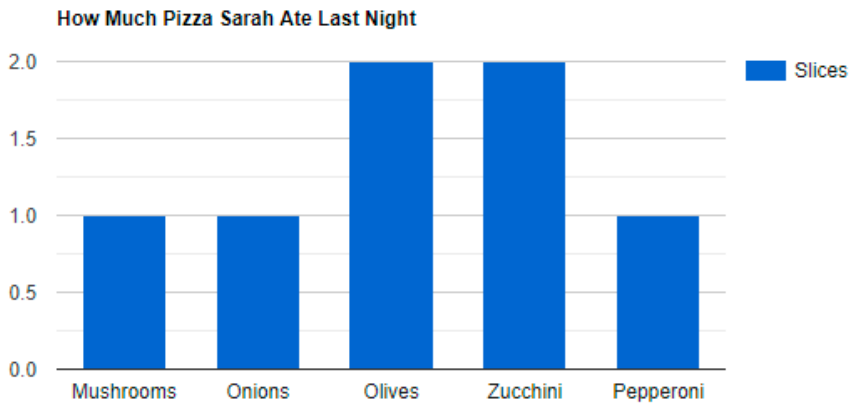


Figura 3. *Column bar chart.*

Geo chart

Cambiamos completamente de tipo de visualización, pero será más fácil si ya se domina las anteriores gráficas. La librería permite para mostrar un país, un continente o una región.

En lugar de cargar la librería , cargamos la librería .

```
google.charts.load('current', { 'packages': ['geochart'] });
```

Cambiamos nuestra estructura de datos para **definir localizaciones**:

```
var data = google.visualization.arrayToDataTable([
['Country', 'Popularity'],
['Germany', 200],
['United States', 300],
['Brazil', 400],
['Canada', 500],
['France', 600],
['RU', 700]
]);
```

Luego dibuja el mapa:

```
var chart = new google.visualization.GeoChart
(document.getElementById('regions_div'));
chart.draw(data, options);
```

Este es el resultado que deberías tener:

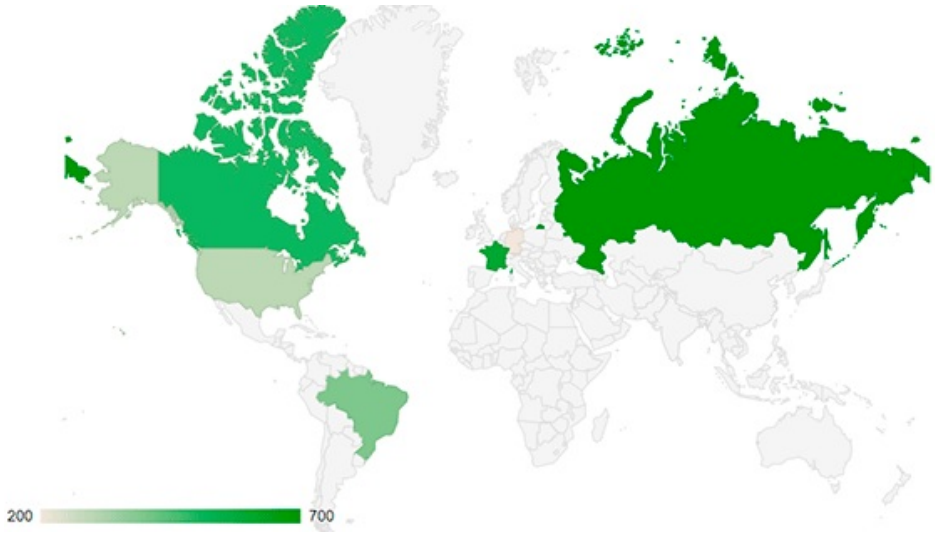


Figura 4. *Geo chart.*

Intenta modificar el código anterior.

Puedes acceder al código completo a través del siguiente enlace:

https://developers.google.com/chart/interactive/docs/gallery/geochart

2.3. Conectando con Google Spreadsheets y archivos CSV

Los datos de las visualizaciones no tienen por qué estar incrustados en el código fuente de nuestro fichero .html. Si fuera así, cada vez que quisiéramos mostrar información, deberíamos abrirlo y modificarlo, lo cual no es ni práctico ni realista. Para ello las visualizaciones recurren a leer los datos de **fuentes de datos externas**.

Google Spreadsheets

Si queremos evitar modificar el código JavaScript cada vez que modificamos los datos, podemos conectar los gráficos a hojas de cálculo existentes. El proceso se explica en el siguiente enlace.

Google Spreadsheets:
https://developers.google.com/chart/interactive/docs/spreadsheets

Pero intentaremos extender y explicar por partes el código. El código completo es el siguiente:

Ejemplo: código completo de Google Spreadsheets

```
<html>
<head>
<script type="text/javascript"
src="https://www.gstatic.com/charts/loader.js"></script>
<script type="text/javascript">
google.charts.load("current", {packages:["corechart"]});
google.setOnLoadCallback(drawChart);
function drawChart() {
var query = new google.visualization.Query(
'https://docs.google.com/spreadsheet/ccc?
key=0Atw2BTU52lOCdEZpUlVIdmxGOWZBR2tuLXhYN2dQTWc&usp
=drive_web#gid=0');
query.send(handleQueryResponse);
```

```
}
function handleQueryResponse(response) {
if (response.isError()) {
alert('Error in query: ' + response.getMessage() + ' ' +
response.getDetailedMessage());
return;
}
var data = response.getDataTable();
var chart = new google.visualization.
ColumnChart(document.getElementById('columnchart'));
chart.draw(data, { legend: { position: 'none' } });
}
</script>
<title>Data from a Spreadsheet</title>
</head>
<body>
<span id='columnchart'></span>
</body>
</html>
```

Te llamará la atención que la función ha cambiado:

```
function drawChart() {
var query = new google.visualization.Query(
'https://docs.google.com/spreadsheet/ccc?
key=0Atw2BTU52lOCdEZpUlVIdmxGOWZBR2tuLXhYN2dQTWc&usp=drive_web#gid=0');

query.send(handleQueryResponse);
```

Para utilizar Google Spreadsheet como fuente de datos se necesita:

- Asignar a la hoja de datos permisos de lectura Público en la web o Cualquiera con el enlace.
- Disponer de la URL completa del documento.
- Llamar a la función con la URL.
- Especificar los parámetros y .

- indica cuantas filas son cabecera y por lo tanto deben excluirse como datos.
- se refiere a la hoja a utilizar (en el caso de que el documento tuviera varias hojas).

Puedes copiar la URL y pegarla en tu explorador para ver el contenido.

	A	B
1	Name	Length
2	A	23
3	B	16
4	C	10
5	D	31
6		
7		

Figura 5. Ejemplo de Google Spreadsheet.

El resultado que se obtiene se envía a otra función llamada , cuya función es doble:

Gestionar el error en la petición del documento. ¿Qué errores consideramos en este caso? Por ejemplo, que la URL del documento no sea correcta. Probadlo en vuestro navegador o en Brackets, borrad uno de los caracteres de la URL y veremos cómo aparece una ventana reportando un error.

Dibujar la gráfica con elmétodo . Para ello se siguen los siguientes pasos:

- Asignamos los datos al gráfico a través de la variable response. El objeto response se pasa a la función cuando se llama a la función .
- Las opciones de la gráfica se definen al llamar a la función .

La **función** dibuja la visualización de la gráfica y tiene dos parámetros:

- : alberga los datos a usar para dibujar la gráfica.
- : mapa con duplas de valores en formato nombre y valor. Ejemplo:

```
{x:100, y:200, title:'An Example'}
function handleQueryResponse(response) {
if (response.isError()) {
alert('Error in query: ' + response.getMessage() + ' ' +
response.getDetailedMessage());
return;
}
var data = response.getDataTable();
var chart = new google.visualization.
ColumnChart(document.getElementById('columnchart'));
chart.draw(data, { legend: { position: 'none' } });
}
```

El resultado de este código es el siguiente:

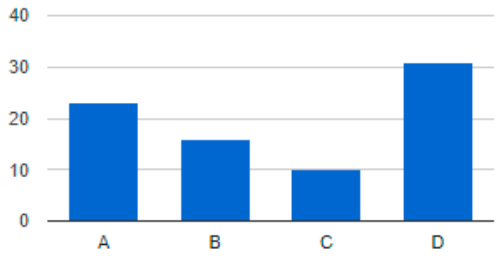


Figura 6. Resultado visualización datos Google spreadsheet.

Accede más información sobre la función y las opciones a utilizar a través del siguiente enlace:

https://developers.google.com/chart/interactive/docs/reference#visdraw

Archivos CSV

Los archivos CSV (*comma-separated values* o valores separados por comas) son un tipo de documento de texto ampliamente utilizado para **representar datos en forma de tabla**. Su característica principal es que las columnas se separan por comas y las filas por saltos de línea.

Los archivos CSV normalmente se utilizan para importar o exportar información de bases de datos.

En este ejercicio también utilizaremos jQuery, una librería de código abierto de JavaScript, para el tratamiento del fichero CSV. El código adaptado sería el siguiente:

Ejemplo: código adaptado de CSV

```
<html>
<head>
<title>Google Chart Example</title>
<script type="text/javascript"
src="//ajax.googleapis.com/ajax/libs/jquery/1.10.2/
jquery.min.js">
</script>
<script src="jquery.csv-0.71.js"></script>
<script type="text/javascript"
src="https://www.gstatic.com/charts/loader.js"></script>
<script type="text/javascript">
google.charts.load("current", {packages:["corechart"]});
google.setOnLoadCallback(drawChart);

function drawChart() {
// grab the CSV
$.get("https://docs.google.com/spreadsheet/pub?
key=0AhNdznUnSazTdElCVTRTYzF6Zm9DSmlUQkx1NE5DY1E&output
=csv",
function(csvString) {
// Transformar CSV a array utilizando jQuery
var arrayData = $.csv.toArrays(csvString, {onParseValue:
$.csv.hooks.castToScalar});
// Convertir array a DataTable
var data =
new google.visualization.arrayToDataTable(arrayData);
// Generar gráfica desde el DataTable
var chart = new
google.visualization.ColumnChart(document.getElementById
```

```
('columnchart'));
chart.draw(data, { legend: { position: 'none' } });
});
}
</script>
</head>
<body>
<div id="columnchart"></div>
</body>
</html>
```

Accede al fichero CSV que utilizaremos en el ejemplo a través del siguiente enlace:

https://docs.google.com/spreadsheet/pub?key=0AhNdznUnSazTdElCVTRTYzF6Zm9

Respecto al anterior ejemplo, resaltaremos el siguiente código:

```
$.get("https://docs.google.com/spreadsheet/pub?
key=0AhNdznUnSazTdElCVTRT
YzF6Zm9DSmlUQkx1NE5DY1E&output=csv", function(csvString) {
// transform the CSV string into a 2-dimensional array
var arrayData = $.csv.toArrays(csvString, {onParseValue:
$.csv.hooks.castToScalar});
...
});
```

Lo más importante a tener en cuenta es que el es un **formato** utilizado en jQuery. Esto crea una función y todo el código que va a continuación está contenido en dicha función:

- En cierta manera, cuando vemos el símbolo $ es equivalente al de JavaScript estándar. Quiere decir que vamos a hacer algo con el documento HTML.
- El realiza una petición http a la URL indicada.

A continuación es cuando la librería de jQuery para ficheros CSV nos ayuda, haciendo una llamada a , transformamos el fichero CSV a un formato que es

comprensible para la librería de Google Charts.

El resultado es el mismo que el obtenido desde la fuente de Google Spreadsheets, pero en este caso, tomando como referencia un fichero en formato CSV.

Figura 7. Resultado visualización datos del fichero CSV.

2.4. Gestionar eventos

Google Charts genera eventos que se pueden tratar y gestionar. Estos eventos se suelen disparar por acciones de usuario, como por ejemplo cuando se hace clic en una gráfica. Se puede registrar un método JavaScript que sea llamado cuando ocurran dichos eventos e incluso acceder a la información asociada a los mismos.

Cada gráfica y cada interacción definen sus propios eventos y la información asociada. En la documentación de Google Chart explican con detalle los posibles eventos:

Accede a más información sobre eventos a través del siguiente enlace: https://developers.google.com/chart/interactive/docs/events

En esta sección explicaremos de forma sencilla cómo podemos trabajar con los eventos para **interactuar entre dos visualizaciones**.

Ejemplo: código de dos visualizaciones

```
<html>
<head>
<script type="text/javascript"
src="https://www.google.com/jsapi"></script>
<script type="text/javascript">
google.load('visualization', '1.0',
{'packages':['corechart','table']});
// Set a callback to run when the Google
Visualization API is loaded.
google.setOnLoadCallback(drawSort);
function drawSort() {
var data = new google.visualization.DataTable();
data.addColumn('string', 'Name');
data.addColumn('number', 'Salary');
data.addColumn('boolean', 'Full Time');
data.addRows(5);
data.setCell(0, 0, 'John');
```

```
data.setCell(0, 1, 10000);
data.setCell(0, 2, true);
data.setCell(1, 0, 'Mary');
data.setCell(1, 1, 25000);
data.setCell(1, 2, true);
data.setCell(2, 0, 'Steve');
data.setCell(2, 1, 8000);
data.setCell(2, 2, false);
data.setCell(3, 0, 'Ellen');
data.setCell(3, 1, 20000);
data.setCell(3, 2, true);
data.setCell(4, 0, 'Mike');
data.setCell(4, 1, 12000);
data.setCell(4, 2, false);
var formatter = new google.visualization.NumberFormat
({prefix: '$'});
formatter.format(data, 1); //
 Apply formatter to second column
var view = new google.visualization.DataView(data);
view.setColumns([0, 1]);
var table = new
google.visualization.Table(document.getElementById
('table_sort_div'));
table.draw(view);
var chart = new
google.visualization.BarChart(document.getElementById
('chart_sort_div'));
chart.draw(view);
google.visualization.events.addListener(table, 'sort',
function(event) {
data.sort([{column: event.column, desc:
!event.ascending}]);
chart.draw(view);
}
);
}
</script>
</head>
<body>
<!--Div that will hold the pie chart-->
<div id="chart_sort_div"></div>
<div id="table_sort_div"></div>
```

```
</body>
</html>
```

En este último ejemplo de código prestad atención a dos detalles, ya que tenemos:

- Dos visualizaciones: una tabla y un *bar chart*.
- La función :

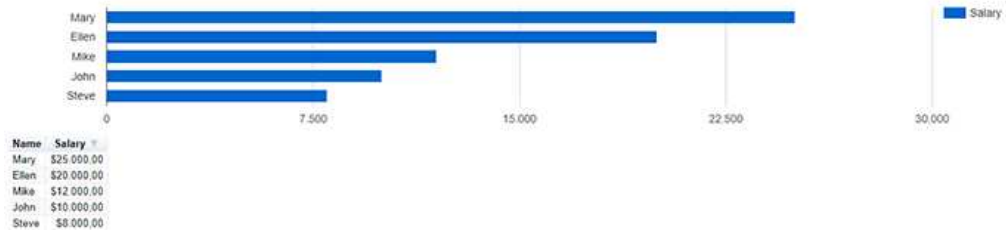
```
var table = new
google.visualization.Table(document.getElementById('table_sort_div'));

table.draw(view);
var chart = new
google.visualization.BarChart(document.getElementById('chart_sort_div'));

chart.draw(view);
google.visualization.events.addListener(table, 'sort',
function(event) {
data.sort([{column: event.column, desc: !event.ascending}]);
chart.draw(view);
});
```

La función es la que utilizaremos si queremos trabajar con eventos. Lo que creamos con esta función es una **entidad que «escuchará» los eventos** de las visualizaciones.

En este código creamos un que escucha los eventos de nuestro objeto table (primera variable de la función). Escucha solo un tipo de eventos: Es decir, cuando un usuario ordena la tabla que normalmente esa acción genera al pulsar en la cabecera de la tabla, el evento se lanza. Una vez que el escucha el evento, ordena nuestro objeto data con la función y redibuja el *bar chart* .

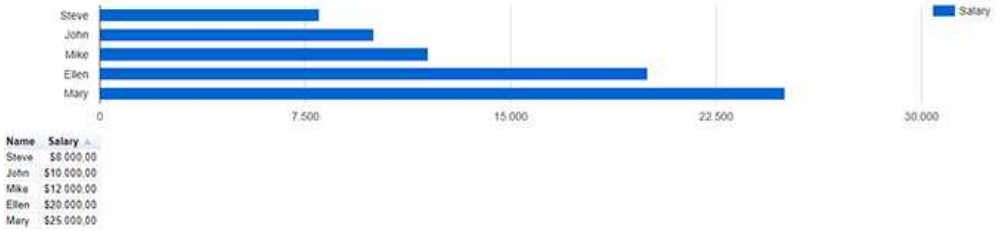
El resultado de este código es el siguiente: en la gráfica superior los datos están ordenados de mayor a menor salario. En la gráfica inferior, su visualización ha

cambiado automáticamente cuando hemos ordenado los datos de la tabla en sentido inverso, de menor a mayor.



Name	Salary
Mary	$25.000,00
Ellen	$20.000,00
Mike	$12.000,00
John	$10.000,00
Steve	$8.000,00

Figura 8. Resultado visualización datos ordenados de mayor a menor salario.



Name	Salary
Steve	$8.000,00
John	$10.000,00
Mike	$12.000,00
Ellen	$20.000,00
Mary	$25.000,00

Figura 9. Resultado visualización datos ordenados de menor a mayor salario.

2.5. Referencias bibliográficas

Tischler, L. (1 de noviembre de 2005). The Beauty of Simplicity. *Fast Company*. https://www.fastcompany.com/56804/beauty-simplicity

Controls and Dashboards

Google Developers. *Controls and Dashboards*. Google Charts. https://developers.google.com/chart/interactive/docs/gallery/controls

Esta guía muestra cómo construir un *dashboard* con Google Charts.

Ejemplos de Google Chart

Google Developers. *Chart Gallery*. Google Charts. https://developers.google.com/chart/interactive/docs/gallery

En Chart Gallery se pueden ver y probar más ejemplos de Google Charts.

Google Chart eventos

Google Developers. *Code Examples*. Google Charts. https://developers.google.com/chart/interactive/docs/examples

En Code Examples se puede ver y gestionar eventos de Google Charts.

Test

1. `necesita ser instanciada con un new para crear un objeto:`

   A. Sí.

   B. No, solo cuando le asignas valores directamente.

   C. No, solo cuando trabajas la primera vez con la gráfica.

   D. No.

2. ¿Qué función tiene ?

   A. Carga el propio cargador de librerías.

   B. Importa la librería *jQuery.*

   C. Carga los diferentes paquetes de la librería.

   D. Carga la última versión de Google Charts.

3. La primera variable que recibe la función `de JQuery es:`

   A. La URL a la que se dirige la petición.

   B. El evento que ha generado la visualización.

   C. El nombre de función destino donde se pasan los valores.

   D. El *array* de datos a visualizar.

4. `añade una columna a nuestra tabla de datos:`

   A. Sí, tantas veces como aparezca esta línea de código.

   B. Sí, aunque no admite valores iguales.

   C. No, porque la tabla ya viene con las columnas definidas.

   D. No.

5. Existen librerías en JQuery que nos facilitan la transformación de archivos CSV a datos comprensibles por la librería de Google Charts:

A. Sí.

B. No, porque jQuery no incorpora dichas funciones.

C. No, porque CSV no es un formato compatible con Google Chart.

D. Sí, aunque depende de que las versiones sean compatibles.

6. `nos dibujará *bar charts* verticales:`

A. Sí, admite tanto barras verticales como horizontales.

B. Sí, hace lo mismo que *colum chart*.

C. Sí, tanto barras independientes como apiladas.

D. No.

7. ¿Qué hace el código cuando contiene ?

A. Comprueba si el evento generado por otra visualización tiene el formato correcto.

B. Comprueba si ha habido algún error con la petición http.

C. Comprueba si ha habido algún problema en la generación de la visualización.

D. Comprueba si los datos del archivo son coherentes.

8. `se utiliza para escuchar eventos que generan las visualizaciones:`

A. Sí.

B. No, solo aplica los eventos del ratón.

C. No, se encarga de los eventos del sistema operativo.

D. No.

9. Todas las visualizaciones de Google Chart generan los mismos tipos de eventos:

A. Sí, porque todas funcionan de la misma forma.

B. Sí, porque de esta forma el código es escalable.

C. Sí, porque ello permite cambiar la gráfica sin tener que cambiar los eventos.

D. No.

10. ¿Incluiremos el paquete `para visualizar un mapa?`

A. Sí, para mostrar un país, un continente o una región.

B. Sí, para mostrar incluso direcciones de calles.

C. No, porque no existe dicha librería.

D. No.