**Tutorial: DC.JS e Geo Choropleth Chart**

Baseado em <https://steemit.com/utopian-io/@faad/tutorial-11-dive-into-dc-js-a-javascript-library-geo-choropleth-chart> (05/03/2018)

O que vou aprender?

* Desenvolver gráfico Geo Choropleth
* Analisar dados CSV em d3.js
* Analisar dados JSON em d3.js

Requisitos

* Noções básicas de HTML e CSS
* Noções básicas de Crossfilter.js
* Fundamentos do JavaScript

Dificuldade

* Intermediário

Conteúdo do tutorial

* O que é o gráfico Geo Choropleth?
* Criar o ambiente de trabalho
* Desenvolver o gráfico de Geo Choropleth

# O que é o gráfico Geo Choropleth?

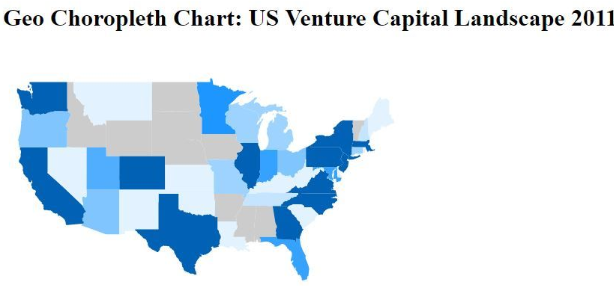
Um mapa choropleth (coroplético) é uma apresentação do mapa em que as áreas são coloridas ou padronizadas em proporção à medição da variável estatística exibida no mapa, como renda per capita ou a população por área.

Os mapas coropléticos fornecem uma maneira fácil de comparar as diferentes áreas com base em dados estatísticos. Com ele é possível observar como uma medição varia em uma área geográfica ou como mostrar o nível de variabilidade dentro de uma região.

Hoje, vamos desenvolver um gráfico de coropleto de "US Venture Capital Landscape 2011", regiões diferentes podem ser sombreadas de forma diferente com base em cálculos diferentes. Aqui, para nosso tutorial, trabalharemos em dois formatos de dados, CSV e JSON. O arquivo CSV fornece os dados estatísticos e o arquivo JSON nos fornece as informações que nos ajudarão a desenvolver nosso gráfico. Primeiro baixe esses dois arquivos:

CSV (<https://raw.githubusercontent.com/dc-js/dc.js/develop/web/vc/vc.csv>)

JSON (<https://raw.githubusercontent.com/dc-js/dc.js/develop/web/vc/vc.csv>)



# Criar o ambiente de trabalho

Antes de passarmos para nosso gráfico, teremos que criar um ambiente de trabalho que nos permita desenvolver nosso gráfico. Primeiro de tudo abra seu editor de texto, crie um arquivo com o nome **geoChoroplethChart.html** e salve-o em seu diretório de trabalho.

Em seguida, escreva todo o código HTML necessário para trabalhar com arquivos HTML.



Agora escreva o código para o cabeçalho da página e para a div na qual você deseja mostrar seu gráfico “geoChoroplethChart”. Escreva esse código dentro da tag body.

<h1>Geo Choropleth Chart: US Venture Capital Landscape 2011</h1>

<div id="geoChoroplethChart"></div>

Agora, vincule seu arquivo à biblioteca **dc.js**. Aqui, você deve vincular dois arquivos dc.js, um para o arquivo js e outro para o arquivo css. Adicione este código na tag <head>

<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/dc/2.1.9/dc.min.js"></script>

<link rel="stylesheet" type="text/css" href="https://cdnjs.cloudflare.com/ajax/libs/dc/2.1.9/dc.min.css" />

Mas o dc.js não funciona sozinho, por isso você tem que vincular outras duas bibliotecas das quais o dc.j depende: d3.js e crossfilter.js. Também ligue estes dois.

<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/d3/3.5.1/d3.min.js"></script>

<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/crossfilter/1.3.12/crossfilter.min.js"></script>

Agora você está apto a trabalhar com dc.js.

# Inicie a desenvolver o gráfico de Geo Choropleth

Agora vamos passar para o nosso gráfico. Primeiro vamos trabalhar no nosso arquivo CSV, analisar os dados csv que podemos usar esses dados para desenvolver nosso gráfico. D3.JS nos permite analisar os dados csv, nos fornecer um método csv ([]). Use este método.

d3.csv("vc.csv", function(error, csvData){

// The rest of the code will be written withing this body....

});

Você pode ver que nós passamos dois argumentos para o método csv ([]), primeiro é o nome do arquivo csv e o outro é um método (função). O método em si contém dois argumentos: “error” que contém os erros se existirem, e “csvData” com os dados analisados do arquivo csv. Você pode verificar esses dados no console, escrevendo este código.

Console View:



Se você verificou, os dados analisados retornam para nós em forma de “string”; isso cria um problema para nós quando estamos trabalhando com dados numéricos. Você pode ver a propriedade “Raised” (é a última coluna no arquivo CSV), que possui dados em formato numérico. Então, vamos ter que realizar a conversão. Existem algumas maneiras de fazer isso, mas achamos essa a melhor forma:

csvData.forEach(function(d){

d.Raised = +d.Raised

});

Now, pass this our parsed data to the crossfilter.js that'll allow us to create dimension and grouped our data.

var facts = crossfilter(csvData);

Its time to create dimension, we'll create the dimension by states.

var dimensionByState = facts.dimension(function (d) { return d.State });

Now grouped the data, we will sum up the Raised fund by states.

var groupByRaised = dimensionByState.group().reduceSum(function (d) { return d.Raised });

Here above we completed our word with csv data, now we will work with JSON data that allow us to create US map.

First we parse the json data with the help of d3.js library, the library gives us a method json([]). That allow us to parse the json data. This method works same like the csv([]) method, we've explained above how it works. Write this code.

d3.json("us-states.json", function(error, jsonData){

// dc.js code will be written here.

});

We've done our data related stuff , its time to develop our chart, create the object of geoChoroplethChart class and pass it the div's ID we created above. Also define the width and height of the chart.

var geoChoroplethChart = dc.geoChoroplethChart("#geoChoroplethChart")

.width(1024)

.height(600)

Now pass the dimension and group of the data that we create above from crossfilter.js library.

var geoChoroplethChart = dc.geoChoroplethChart("#geoChoroplethChart")

.width(1024)

.height(600)

.dimension(dimensionByState)

.group(groupByRaised)

Here, we are one step away from our chart, geoChoroplethChart give s us a special method  
overlayGeoJson(json, name, keyAccessor), this method is mandatory to create our chart. The method allow us to insert a new GeoJson map layer, the method executed number times, depends upon the how much GeoJson data layers you have. If you overlay multiple layers with the same name the new overlay will override the existing one.

It takes three argumenst, geojsonData, name of the data and keyAccessor.

d3.json("us-states.json", function(error, jsonData){

var geoChoroplethChart = dc.geoChoroplethChart("#geoChoroplethChart")

.width(1024)

.height(600)

.dimension(dimensionByState)

.group(groupByRaised)

.overlayGeoJson(jsonData.features, "State",

function(d){

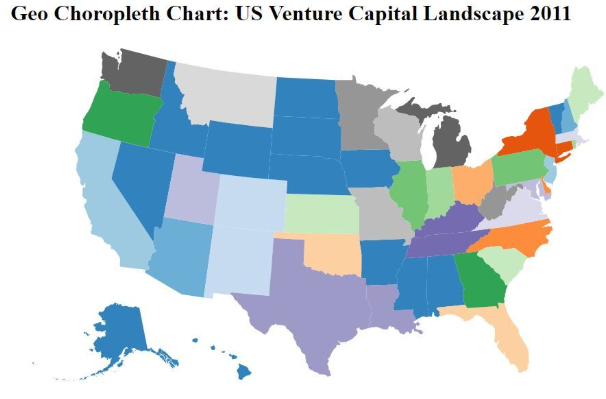
return d.properties.name;

});

dc.renderAll();

});

Output:



You can see the state are of represented by different colors depends the amount raised for each state. But if you want a single color of the map as in the first image of the tutorial. then add this code to your above code.

.colorDomain([0,200])

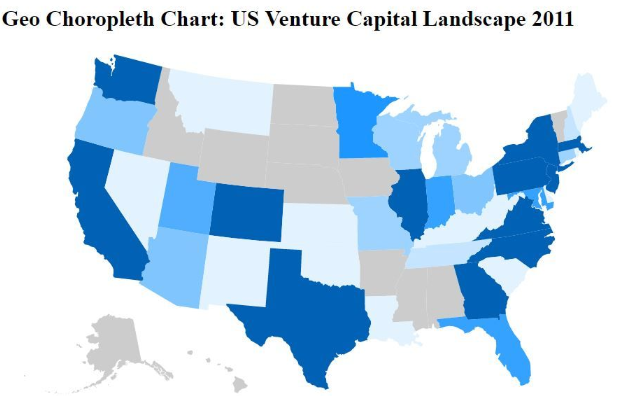
.colorCalculator(function(d){

return d ? geoChoroplethChart.colors()(d) : '#ccc';

});

* .**colors([])** : set the, only show these colors.
* .**colorDomain([])** : this method Set /et the current domain for the color mapping method.
* .colorCalculator([]): this method determining the min and max values as retrieved by .colorAccessor and sets the domain.

Output:



You can see colors the a set according to our requirement.

You can also set the projection of chart over world map, to do this

var centre = d3.geo.centroid(jsonData);

var projection = d3.geo.mercator().center(centre).scale(500).translate([200,100]);

Create the projection through d3.js, the library includes several [common projections](https://github.com/d3/d3-3.x-api-reference/blob/master/Geo-Projections.md) by default. Here we use the d3.geo.mercator you can use whatever you want

Now pass this project to our chart.

.projection(projection)

The code will :

var geoChoroplethChart = dc.geoChoroplethChart("#geoChoroplethChart")

.width(1024)

.height(600)

.dimension(dimensionByState)

.projection(projection)

.group(groupByRaised)

.overlayGeoJson(jsonData.features, "State", function(d){

return d.properties.name;

})

.colors(d3.scale.quantize().range(["#E2F2FF","#C4E4FF","#9ED2FF","#81C5FF","#6BBAFF","#51AEFF","#36A2FF","#1E96FF","#0089FF","#0061B5"]))

.colorDomain([0,200])

.colorCalculator(function(d){ return d ? geoChoroplethChart.colors()(d) : '#ccc'; });

dc.renderAll();

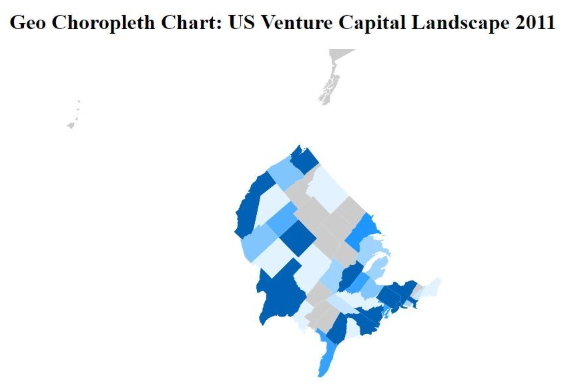
});

You can see our chart become little smaller and moved toward left, you can set the chart where ever you want by just changing the translate([]);method. Every projection in d3.js is also provided with translation.

Here we tried another projection to make you clear and easier for you to use. We use d3.geo.conicEqualArea . Replace this code with you old projection code,

var projection = d3.geo.conicEqualArea().center(centre).scale(500).translate([480, 250]);

Output:



You can see how our view of the chart changes.

**Source Code:**

<!DOCTYPE html>

<html lang="EN">

<head>

<meta charset="utf-8">

<title>Geo Choropleth Chart</title>

<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/d3/3.5.1/d3.min.js"></script>

<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/crossfilter/1.3.12/crossfilter.min.js"></script>

<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/dc/2.1.9/dc.min.js"></script>

<link rel="stylesheet" type="text/css" href="https://cdnjs.cloudflare.com/ajax/libs/dc/2.1.9/dc.min.css" />

</head>

<body>

<h1>Geo Choropleth Chart: US Venture Capital Landscape 2011</h1>

<div id="geoChoroplethChart"></div>

<script type="text/javascript">

// https://raw.githubusercontent.com/dc-js/dc.js/develop/web/vc/vc.csv

// https://raw.githubusercontent.com/dc-js/dc.js/develop/web/geo/us-states.json

d3.csv("vc.csv", function(error, csvData){

console.log(csvData);

csvData.forEach(function(d){

d.Raised = +d.Raised

});

var facts = crossfilter(csvData);

var dimensionByState = facts.dimension(function(d){ return d.State})

var groupByRaised = dimensionByState.group().reduceSum(function(d){ return d.Raised});

d3.json("us-states.json", function(error, jsonData){

console.log(jsonData);

var centre = d3.geo.centroid(jsonData);

var projection = d3.geo.mercator().center(centre).scale(500).translate([200,100]);

var geoChoroplethChart = dc.geoChoroplethChart("#geoChoroplethChart")

.width(1024)

.height(600)

.dimension(dimensionByState)

.projection(projection)

.group(groupByRaised)

.overlayGeoJson(jsonData.features, "State", function(d){

return d.properties.name;

})

.colors(d3.scale.quantize().range(["#E2F2FF","#C4E4FF","#9ED2FF","#81C5FF","#6BBAFF","#51AEFF","#36A2FF","#1E96FF","#0089FF","#0061B5"]))

.colorDomain([0,200])

.colorCalculator(function(d){ return d ? geoChoroplethChart.colors()(d) : '#ccc'; });

dc.renderAll();

});

});

</script>

</body>

</html>