Tuto2: Developing Data visualizations with D3.js

Nicolas Médoc   Luxembourg Inst. of Science and Technology

Outline

1. Scales transformation: from data space to visual space

2. Interactions

The scales

- the scales allows to apply scale transformations from the data space (the domain values) to a range of values corresponding to any chosen visual encoding

- different types of scale are available for different types of transformation: linear, pow, log, ordinal, quantile, ..

- `https://d3js.org/d3-scale`

Exercise: Use appropriate scales to map data to visual variables

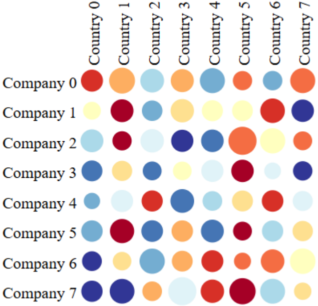
*In the generated dataset, each cell represent a company that sells a certain number of products (nbProductSold) in different countries. A second attribute is **salesGrowth**, a positive or negative sales growth rate*

`https://github.com/nicolasmedoc/Tuto2-D3js`

Exercise: Use appropriate scales to map data to visual variables

*In the generated dataset, each cell represent a company that sells a certain number of products (nbProductSold) in different countries. A second attribute is **salesGrowth**, a positive or negative sales growth rate*

`https://github.com/nicolasmedoc/Tuto2-D3js`



Size and color of cells

```
// build the size scale
const radiusMin = 2;
const radiusMax = cellSize / 2;
const minNbProductSold = d3.min(genData.map(cellData=>cellData.nbProductSold));
const maxNbProductSold = d3.max(genData.map(cellData=>cellData.nbProductSold));
const cellSizeScale = d3.scaleLinear()
    .domain([minNbProductSold, maxNbProductSold])
    .range([radiusMin, radiusMax-1])
;
// build the color scale
const colorScheme = d3.schemeRdYlBu[11];
const cellColorScale = d3.scaleQuantile()
    .domain(genData.map(cellData=>cellData.salesGrowth))
    .range(colorScheme)
;
```

Example: using scales for encoding the size and color of circles

And we use the scales to compute the circle attributes by using a function with the data item in parameter:

```
...
    .attr("r",(cellData)=>cellSizeScale(cellData.nbProductSold))
    .attr("fill",(cellData) =>{
        const color =  cellColorScale(cellData.salesGrowth);
        return color;
    })
...
```

Adding interactions

In D3js you can declare events with .on() function:

```
.on("click",(event, cellData)=>{// do something with event and/or cellData})
.on("mouseenter",(event, cellData)=>{// do something with event/cellData})
.on("mouseleave",(event, cellData)=>{// do something with event/cellData})
```

Exercise 2: highlighting the cell border on click

```
function renderMatrix(genData)
...
      .append("g")
      .attr("class","cellG")
      .attr("transform",(cellData)=>{
          return "translate("+(cellData.colPos*cellSize)+ ...
      })
      .on("click", (event,cellData)=>{
          handleOnClickCell(cellData);
      })
    ;
...
    cellG.append("circle")
      .attr("class","CellCircle")
      .attr("stroke", "black")
      .attr("stroke-width", (cellData)=>cellData.selected?2:0)
```

Exercise 2: highlighting the cell border on click

```
function handleOnClickCell(cellData){
    genData=genData.map(item=>{
        if (item.index===cellData.index){
            return {...item,selected:!cellData.selected};
        }else{
            return item;
        }
    })
    renderMatrix(genData);
}
```

Exercise 2: highlighting the cell border on click

Since genData is updated and no new item is added, enter() selection is empty and nothing happens when calling renderMatrix() on click event. A quick-and-dirty solution consists in removing all elements and rebuild the vis with a new data binding,

```
...
function removeMatrix(){
    matSvgG.selectAll(’*’).remove();
}
function renderMatrix(genData) {
    removeMatrix();
...
```

But it is not optimal. It would be preferable to render only the updated item. This is possible by using the update pattern proposed by D3js.

General update pattern with join()

The general update pattern of D3js allows declaring different behaviors after binding new/updated data to a selection. The **join()** function called just after the data binding takes in parameter 3 functions to declare these behaviors:

- **enter function** to define what to do with new items;
- **update function** to handle the items matching with the previous data binding;
- **exit function** for old items that does not exist anymore.

See illustrations at `https://bost.ocks.org/mike/selection/#enter-update-exit` and `https://observablehq.com/@d3/selection-join`.

Exercise 3: implement the update pattern

In renderMatrix avoid calling removeMatrix() and call join() function following the example below:

```
const cellG = this.matSvg.selectAll(".cellG")
    .data(matrixData.genData,(cellData)=>cellData.index)
    .join(
        enter =>{// appends elements with fixed atributes
            // append cellG
            // append CellRect with the color
            // append CellCircle at center position
        },
        update =>{ // select elements and declare changing attributes
            // the cell position (<g> translation)
            // the circle size
            // the circle color
        },
        exit =>{ // declare what to do with items that don’t exist anymore
            exit.remove();
        }
    )
.
```

Exercise 4: Optimizing updates

In certain cases we want to re-render only a few number of items, e.g. highlighting clicked or hovered cell(s). In that case we don’t need to re-render all the cells. So we create a specific function to declare this behavior with update pattern:

```
function handleOnClickCell(cellData){
    const cellsToUpdate=[{...cellData,selected:!cellData.selected}]
    updateCellHighlighting(cellsToUpdate)
}
...
function updateCellHighlighting(cellsToUpdate){
    matSvgG.selectAll(".cellG")
        .data(cellsToUpdate, cellData=>cellData.index)
        // no need to call join() because we don’t need enter or exit
        // update selection is already returned by data()
        .select(".CellCircle")
        .attr("stroke-width",cellData=>cellData.selected?2:0);
    ;
}
```

Exercise 5: using animated transitions

Before removing elements or updating attributes we can declare an animated transition with a specific duration to smoothly observe the transitions between updated positions or colors:

```
cellG.transition()
    .duration(transitionDuration)
    .attr("transform",(cellData)=>{
        return "translate("+(cellData.colPos*this.cellSize)...
    })
;
```

Add transitions before update and exit of the join

Exercise 5 (optional)

Use mouseenter and mouseleave events to implement the mouse hover interaction on matrix cells

```
.on("mouseenter",(event, cellData)=>{// do something with event/cellData})
.on("mouseleave",(event, cellData)=>{// do something with event/cellData})
```

Exercise 6 (optional)

In renderMatrix(), add labels in top and left margin.

Exercise 7 (optional)

- When the user clicks on country labels, sort the products by their decreasing nbProducSold
- When the user clicks on company labels, sort the countries by their decreasing nbProducSold