CSCI 4210 U – Information Visualization
Lab 5. Time-Varying Scatterplots

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

PURPOSE

Create an animated visualization of the Gapminder dataset Exercise using marks and channels in D3. Solidify understanding of d3 transitions.

TASKS

The Gapminder dataset contains economic indicators for countries over the years.
Open gapminder.csv with your tool of choice and familiarize yourself with the dataset attributes.
Open index.html with your favourite code editor. Then serve the page with python's http.server:

```
cd labfolder
python -m http.server 3000
```

When writing code, make sure you introduce and test one change at a time. This way you know which part of the code has a bug. If you are not sure a function call will work, open Chrome Developer Tools (Ctrl + Shift + I), place a breakpoint where you plan to insert the call, and test the call on the Console. Check if it returns what you expect. If you don't know how to use a debugger, ask the TA.

1. Load the dataset

Use d3.csv to load the dataset. d3.csv makes a HTTP GET request to a URL pointing to a csv file, then transforms the file into an array of objects, each of which representing a row. Note that this function is asynchronous; that is, the result will be passed to a callback. As a consequence, *any code placed after the call will not have access to the data*. Data access is guaranteed only within the callback.

```
d3.csv(“gapminder.csv”, function(records){

// Any code that relies on the data goes here.

});
```

Note that quotation marks are not usually well understood by code editors when copied from somewhere else. You may want to type the commands as opposed to copy and paste.

Then parse all numerical attributes to Number. d3.csv assumes all values are strings. This can cause bugs in later when you perform calculations.

```
records.forEach(function(record){
    record.year = Number(record.year);
    // continue for all other numerical attributes
});
```

The code above uses Array’s forEach function. Feel free to write a traditional for loop instead.

2. Group the records by year

```
  recordsByYear = {};
  records.forEach(function(record){
    let year = record.year;
```

```
    if (recordsByYear[year] == undefined)
      recordsByYear[year] = [];
    recordsByYear[year].push(record);
  });
```

3. Create a function that displays Life Expectancy vs GDP per Capita for an array of records. The size of a data point should be proportional to Population. The colour should encode Continent.

3.1. Create a function with the following signature:

```
function plot(records){


}
```

In order to test this function, call **plot(recordsByYear[1960])** from inside the callback supplied to d3.csv.

3.2. Create scales for every mapping of continuous attributes (life expectancy to y position, GDP per capita to x position, population to radius)[1].

For instance:

```
var yScale = d3.scaleLinear()
  .domain(d3.extent(records, function(record){
        return record.lifexp;
  }))
  .range([450, 20]);


// yScale(records[0].lifexp) // testing the scale
```

It's a good idea to use a Log scale for population. For x and y, use d3.scaleLinear.

---

[1] Many examples containing scales and selections can be found in https://veras.works/lectures/d3-intro-one/. Check also the Diamond dataset jsfiddle: https://jsfiddle.net/rafa_veras/hqvxhehb/

3.3. Create an ordinal scale for color (which will encode continent). This scale is different from the others because both the domain and range are discrete.

```
d3.scaleOrdinal()
  .domain(arrayOfDistinctContinentValues) // e.g. [“Africa”,…]
  .range(arrayOfColors); // one colour for each continent
```

To easily extract the distinct values of an array, you can pass the array to Set, then back to Array with the spread operator (…). Of course, you can accomplish this in many other different ways.

```
var arr = new Set([1,1,1,1,1,2,4,4,5]);
var uniq = […arr]; // [1,2,4,5]
```

3.4. Create and/or Update and/or Remove elements.

index.html has an SVG element already defined. Select this element, then select all circles inside it and bind the records to it. Store this selection in a variable.

The highlighted variables or values below should be replaced with the name of the appropriate variables.

```
var selection = d3.select(‘svg’)
  .selectAll(‘element’)
  .data(dataArray);
```

Use the enter selection to create elements, if needed:

```
selection.enter()
  .append(‘element’)
  .attr(‘attrName’, function(d){ return aScale(d.anAttribute); });
  // chain .attr calls until you have set all attributes
  // call .style if you need to set a CSS attribute
```

Update existing elements.

```
// here you will repeat the same calls from above
// (as in our app, existing elements are no different than new ones)
selection
  .attr('attrName', function(d){ return aScale(d.anAttribute); });
```

Remove elements when needed

```
selection.exit().remove();
```

With the code blocks above, our plot function will treat every possible case. When called the first time (e.g., for the first year, 1960), the block using selection.enter will make sure new DOM elements are created.
When called a second time (e.g., for the next year, 1961), the update block will ensure the existing elements (created in the first call) are repositioned and resized. The enter selection will ensure new elements are created, if this time the array of records is larger (e.g., there's a new country), and the exit selection will remove elements if the new array is shorter (e.g, a country disappeared, as it happened with USSR).

4. Create axes

In D3, axes are created from scales:

```
var yAxis = d3.axisLeft(yScale);
var xAxis = d3.axisBottom(xScale);
```

This is very convenient because the scales are already defined.
The axis is then formed by creating an SVG group and calling the axis on the corresponding selection.

```
d3.select('svg')
        .append("g")
        .attr("transform", "translate(30,0)")
        .call(yAxis);
```

In the code above, yAxis will “create itself” inside the SVG group (“g element”). It knows the proper axis values from the scale we passed at creation time. Note the transform attribute. It sets the offset of the axis.

5. Call plot(recordsByYear[1960]) from your d3.csv callback function.
This should display a scatterplot.

BONUS TASK - ANIMATE THE PLOT ACROSS THE YEARS

Animations in d3 are called transitions. Any attribute set on a transition is interpolated from the old value to the new over a defined duration, and after a defined delay (0 by default). For instance, in the code below, all elements under the transition will transition smoothly to the new position we set:

```
circles.transition()
  .attr('cx', function(d){ calculate and return new value });
```

Most of the time, all we need to do is add a call to .transition() before the calls to .attr().

Additionally, in order to animate yearly changes in the Gapminder data, we need to make repeated calls to plot, one for each year. These calls should be chained so that a call for year 1974, for instance, only occurs after the transition for year 1973 has finished.

Fortunately, d3.transition triggers an event when a transition ends. We can listen to this event; when it is fired, we know it is time to plot the next year. However, there is catch: the event is fired for every element of a selection. We will only know when all elements have finished if we count the events. The skeleton goes like this:

```
var counter = data.length; // in our case, this is 83 (countries)
selection.transition()
```

```
    .attr(...)
    .attr(...)
    .on('end', function(){  // this function is called upon 'end'
      counter -= 1;
      if (counter == 0) { // if animation ended for all elements
           // increment year (you may want to have year as a global)
           // call plot(recordsByYear[newYear])
       }
    });
```

If you want want to make the function plot agnostic of application state (that's good design), you can require a callback:

```
  function plot(records, callback){...}
```

Then instead of calling plot when the transition ends, you call back the supplied function.