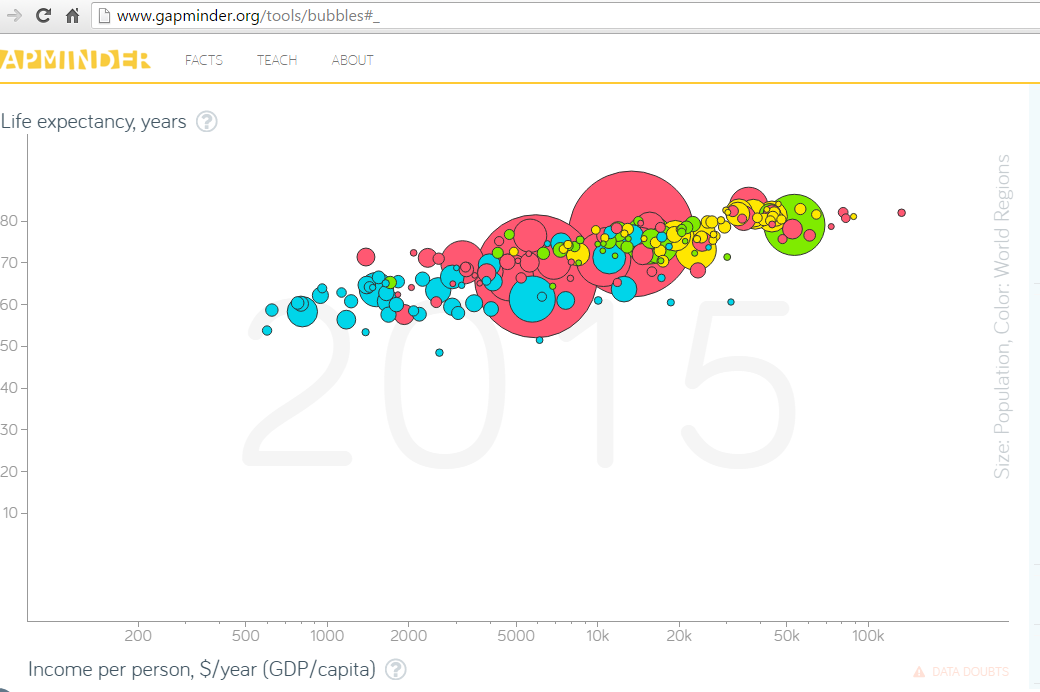
## Owen Galvin

## HU Extension Assignment 09 E63 Big Data Analytics

### Handed out: 04/01/2016 Due by 11:30PM EST on Friday, 04/08/2016

Please, describe every step of your work and present all intermediate and final results in a Word document. Please, copy past text version of all essential command and snippets of results into the Word document with explanations of the purpose of those commands. We cannot retype text that is in JPG images. Please, always submit a separate copy of the original, working scripts and/or class files you used. Sometimes we need to run your code and retyping is too costly. Please include in your MS Word document only relevant portions of the console output or output files. Sometime either console output or the result file is too long and including it into the MS Word document makes that document too hard to read. PLEASE DO NOT EMBED files into your MS Word document. For issues and comments visit the class Discussion Board. You are not obliged to use Java or Eclipse. You are welcome to use any language and any IDE of your choice.

**Problem 1)** Public site GapMinder.org presents many excellent visualizations of data about the World. One such presentation (<http://www.gapminder.org/tools/bubbles#_>) displays average life expectancy in year as a function of average income per person in countries of the world. Countries are represented as circles in different colors depending on their continent, e.g. countries in Europe are yellow, countries in Asia, red, etc. Every country is presented by a circle of area proportional to its population. Radius of the circle is therefore proportional to the square root of population. As the cursor hovers over each country, its name appears over its circle. Data presented in this graph can be found in various Excel files provided by the same site (<http://www.gapminder.org/data/)>. We extracted some files which we believe contain data used in the graph bellow:



Please recreate above graph using D3 or any similar technology of your choice**.** You do not need all data present in provided Excel files. Select most recent data for every country. If you know what you are doing, keep your data in files on your OS or in a database of your choice. Otherwise, copy relevant data directly into your “HTML/JavaScript” code. Please note that the horizontal axis is logarithmic.

**Solution:**

**First a quick overview of how the data was assembled, since I already took some notes during the data munging process:**

1. **countries of the world.xls**

* **Primary data source, if there are countries in the other two xls but not present here, those countries will not make it into final dataset**

1. **indicator life\_expectancy\_at\_birth\_1800-2050.xlsx**

* **Take 2010 Life Expectancy data as “Life”, since anything after 2010 is only projected**

1. **indicator WB data GDP pc ppp.xlsx**

* **Take 2005 GDP data from here where possible – if a country from the primary xls does not report related data here, will use the per capita GDP reported in “countries of the world.xls”. That latter GDP was rounded to nearest hundred, so I’m making an assumption it is more simply sourced/calculated vs. GDP data in .xlsx and therefore should be secondary source.**

**If either Life or GDP lookup fails to match on country name (or either value = 0), that country will be skipped. To keep things simple and avoid need for quoted fields, replaced any comma in country name with a colon, though by this point that was only the Democratic Republic of Congo. Boils down to 177 countries, contained in Lec09\_Data.csv. The .xlsx containing the aggregated data, with formulas, may be included in upload, first 5 columns from Pop tab became the csv. Didn’t try to create a “2015” or similar watermark, in part because data was sourced from various years.**

**To begin the code review, give a quick overview of the javascript structure. There is really just the primary d3.csv() function: 1) path to the csv, 2) accessor function to prep data object from data, and 3) the callback function. Preferably things would have been refactored, separate .js file w/separate functions etc. but I’ve just stuffed most everything into the callback function. Also note the commented out line, which refers to a toy set of data, 5 lines or so.**

|  |
| --- |
| **\*\* lec09\_p1.html \*\***    d3.csv('../Lec09\_Data.csv', function (d) {  // d3.csv('../Lec09\_Data\_short.csv', function(d) {  ...<accessor>  }, function (data) {  ...<callback>  } |

**Now to look at the accessor function first, where the data is cleaned up a bit. After reading about how data is handled, this might have been entirely unnecessary and I could have just used lower case values in the csv header. At a minimum though, setting the population data as numeric, by using plus sign, would have been necessary in order to calculate the radius.**

|  |
| --- |
| d3.csv('../Lec09\_Data.csv', function (d) {  var row = {  country: d.Country,  region: d.Region,  pop: +d.Pop,  life: +d.Life,  gdp: +d.GDP  };  return row;  }, function (data) { |

**At beginning of callback function, sort the data by population size, so that the larger bubbles are drawn first, with smaller bubbles than being drawn on top of as appropriate. Set width, height, some of which are used later in the script to make everything scale together as much as possible. The margins are the same from the example I believe.**

|  |
| --- |
| }, function (data) {  //sort so the smaller bubbles appear in front of larger  data.sort(function (a, b) { return b.pop - a.pop; });  var width = 1150, height = 555,  margin = { top: 20, right: 20, bottom: 20, left: 70 };  //uncomment below for scaled-down version, easier to work with on smaller screen  //width = 750; height = 400; |

**Next I set the two scale variables, both of which will be used later on to sync the data (circle position) and the x and y axes. The first scale is log, as in the Gapminder visualization. Domain for both are set in terms of the min/max values from the csv , combined with an eye toward replicating the relative spacing and location of axes, ticks, etc. in the Gapminder viz. Similar with begin values for the two range() arrays.**

|  |
| --- |
| //horizontal/GDP scale  var xScale = d3.scale.log()  .domain([100, 80000])  .range([50, width]);  var yScale = d3.scale.linear()  .domain([100, -10])  .range([40,height]) |

**For the axes, use the scale variables set above. For the horizontal access, pass in a pre-determined set of tick values to (mostly) match that of the sample viz, along with a d3 format that will format any value >= 1000 with matching ###k value. Tick marks for the vertical y axis are also hard-coded, though derived from an array that goes from 10 to 80 in +10 increments.**

|  |
| --- |
| var xAxis = d3.svg.axis().scale(xScale).orient('bottom')  .tickValues([200,500,1000,5000,10000,20000,50000,100000])  .tickFormat(d3.format('sk'));  yTicks = [];  for (var i = 1; i < 9; i++) {  yTicks.push(i \* 10);  }  var yAxis = d3.svg.axis().scale(yScale).orient('left')  .tickValues(yTicks); |

**Creation of the primary SVG item is next, adding it to the body of the html as var named ‘svg’. Set the width & height as appropriate and assign a class to allow for a bit of CSS styling. Attach to the main svg the text labels for x & y axes. Again, some hard-coded values to make location approximate that of example viz. Add same class to each, for some simple CSS styling.**

|  |
| --- |
| var svg = d3.select('body').append('svg')  .attr('width', width + margin.right + margin.left)  .attr('height', height + margin.top + margin.bottom)  .attr('class', 'frame');  svg.append("text").attr("x", 25).attr("y", 30)  .text("Life expectancy, years").attr('class', 'axis-label');  svg.append("text").attr("x", 45).attr("y", height + 35)  .text("Income per person, $/year (GDP/capita)").attr('class', 'axis-label'); |

**Create the circle items that represent the main data visualization - .selectAll() combined with .data() & .enter() will cause the data points to be manifested as these svg circle objects. Each will be assigned a class representing a cleansed version of their Region value – forced to lower case and any whitespace removed, these will be revisited in the .css file. (Might have been easier to clean the Region values beforehand and consolidate into 4 main World Regions that Gapminder uses but this works well enough.)**

|  |
| --- |
| var circles = svg.selectAll('circle')  .data(data).enter().append('circle')  .attr('class', function (d) {  return d.region.toLowerCase().replace(/\W+/g, '')  }); |

**Following lines of code are all being performed on the just-create circles object.**

1. **Set cx and cy values to line up with GDP on the horizontal axis and Life expectancy on the vertical axis. Note the xScale & yScale, used to sync with axes values.**
2. **Set r value, representing the radius. Instructions were to use a function of the square root of population, so I just tries some numbers until I got appropriately-sized bubbles.**
3. **Handling mouseover and mouseout events. For mouseover the code detects the bubble over which the cursor is hovering and selects all *other* bubbles, making them less opaque and thereby making the current bubble stand out more. (Commented out line was part of orig code but examining during this review led me to believe it was extraneous and indeed everything works correctly without it. Once the mouse/cursor leaves the current bubble, all bubbles are set to full opacity = the standard display.**
4. **Finally , the text pop-up is set, by using Title attribute to invoke standard browser tooltip hover behavior. The call to .text() assembles pieces of each item’s data into three lines. (Chrome and Firefox handle the ‘\n’ line break just fine, IE not so much.)**

|  |
| --- |
| circles.attr('cx', function (d) {  return xScale(d.gdp);  })  .attr('cy', function (d) {  return yScale(d.life);  })  .attr('r', function (d) {  var scaledPop = Math.sqrt(d.pop /1000000);  return scaledPop;  })  .on('mouseover', function (d) {  var circleUnderMouse = this;  d3.selectAll('circle').filter(function (d, i) {  return (this !== circleUnderMouse);  }).style('opacity', '0.4');  //d3.select(this).style('opacity', '1.0');  })  .on('mouseout', function (d) {  d3.selectAll('circle').style('opacity', '1.0');  })  .append('title')  .text(function (d) {  return d.country + '\nIncome: $'  + Math.round(d.gdp) + '\nLife exp: '  + d.life.toFixed(1);  }); |

**The below are debug lines that put a text label on each bubble, displaying associated Country value. Overwhelming with full dataset but helpful when writing the code with only a few bubbles. The CSS hides the text by default.**

|  |
| --- |
| svg.selectAll('text')  .data(data)  .enter()  .append('text')  .text(function (d) {  return d.country;  })  .attr('class', 'country-label')  .attr('x', function (d) { return xScale(d.gdp); })  .attr('y', function (d) { return yScale(d.life); }); |

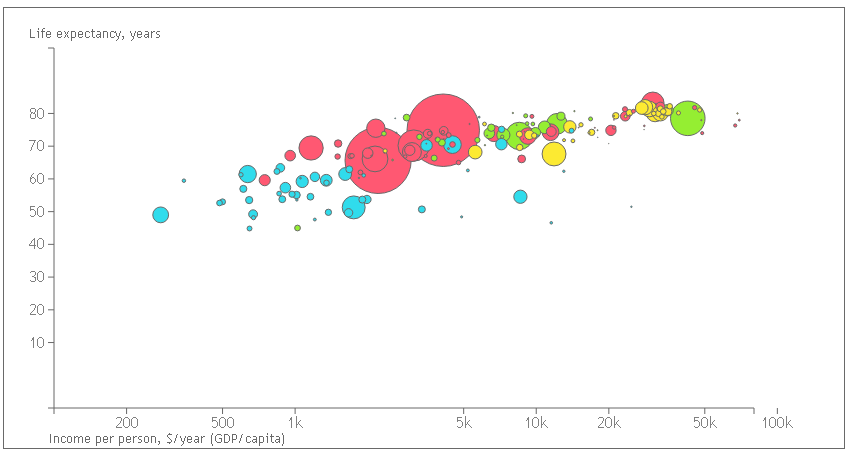
**Last bit of javascript is what draws the x & y axes onto the primary svg canvas, where xAxis & yAxis objects were created earlier. I found they were not being displayed when drawn earlier but work fine when done at the end, here.**

|  |
| --- |
| //keep these at end  svg.append('g')  .attr('class', 'axis')  .attr('transform', 'translate(0,' + height + ')')  .call(xAxis);  svg.append('g')  .attr('class', 'axis')  .attr('transform', 'translate(50,0)')  .call(yAxis);  }); |

**The “logic” behind the CSS is pretty straightforward, so I believe inline comments, in green, will be most effective form of reviewing this code.**

|  |
| --- |
| **\*\* lec09\_p1.css \*\***  /\*simple border around edge of primary svg\*/  .frame {  border: 1px solid dimgray;  }  /\*make the actual x/y axes lines resemble target data viz \*/  .axis {  fill: none;  stroke: dimgray;  stroke-width: 1;  }  /\*affects the tick mark values, i.e. 10,20,30, etc. on y axis\*/  .axis text {  font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida Sans', Arial, sans-serif;  font-size: all;  fill: gray;  stroke: none;  }  /\*same with the axes text labels, e.g. 'Life expactancy, years' for the y axis\*/  text.axis-label {  fill: dimgray;  font-size: small;  font-family: 'Trebuchet MS', 'Lucida Sans Unicode', 'Lucida Grande', 'Lucida Sans', Arial, sans-serif;  }  /\*draw a thin outline around each bubble\*/  circle {  stroke: dimgray;  }  /\*next 4 items set the colors on the circles, where the data still has the original  Region values, which span the four World Regions we want to consolidate into,  so both europe and baltics regions wind up with yellow fill for the bubble.  \*/  circle.europe, circle.baltics {  fill: #FCEA32; /\*yellow\*/  }  circle.africa {  fill: #30DCEC; /\*blue\*/  }  circle.oceania, circle.asia, circle.cwofindstates {  fill: #FF5872; /\*red\*/  }  circle.america, circle.latinamerica {  fill: #95EE32; /\*green\*/  }  /\*debug Country text labels for each bubble, set visibility attribute to visible  (or remove) in order see labels  \*/  .country-label {  fill: dimgray;  outline-color: red;  visibility: hidden;  font-size: x-small;  font-family: Consolas, Arial, sans-serif;  } |

**Screenshot of final viz:**



**Problem 2)** Display the graph of population per country as a pie chart. Color countries by the continents and group all countries by the continents. Leave no separation between different countries, however, when you hover over a country change it color to light purple and display its name and population in millions.

**Solution:**

**Same general code setup as for problem 1, most of the javascript stuffed into the callback function. In this case the accessor function does a little more – I was able to handle the differing region values in P1 using CSS but this solution will require sorting and for that the easiest thing will be to create a new property on the row object that consolidates the eight region values from the source xls into the four World Regions used in the GapMinder viz (I’m assuming this is what is meant by continents as the source xls don’t hold specific “continent” data/labels). So to do that I essentially create a javascript dictionary that will translate the region values into desired “continent” values, held in a new property named area.**

|  |
| --- |
| **\*\* lec09\_p2.html \*\***  d3.csv('../Lec09\_Data.csv', function (d) {  //d3.csv('../Lec09\_Data\_short.csv', function (d) {  var lookup = {  'AFRICA': 'africa',  'AMERICA': 'america',  'ASIA': 'asia',  'EUROPE': 'europe',  'LATIN AMERICA': 'america',  'OCEANIA': 'asia',  'BALTICS': 'europe',  'C.W. OF IND. STATES': 'asia'  };  var row = {  area: lookup[d.Region],  country: d.Country,  region: d.Region,  pop: +d.Pop,  life: +d.Life,  gdp: +d.GDP  };  return row;  }, function (data) { |

**First part of callback function holds the primary “size” variable = diameter, plus some other variables calculated off that number. The primary svg is created, using the width and height variables. The translate transformation appears to essentially center forthcoming pie object.**

|  |
| --- |
| }, function (data) {  var diameter = 720;  //uncomment below for smaller version, e.g. working with dev tools  //diameter = 360;  var width = diameter, height = diameter;  var radius = diameter / 2;  var svg = d3.select('body')  .append('svg')  .attr('class', 'frame')  .attr('width', width)  .attr('height', height)  .append('g')  .attr('transform', 'translate(' + (width/2) + ',' + (height/2) + ')'); |

**Create the pie object, using specific d3 layout.pie() function designed to create such a chart. The value is based on the population of each country. And the sort is needed so that all countries in the same area (aka continent aka World Region) group together and appear in consecutive, seemless, slices of the pie.**

|  |
| --- |
| //sort so regions are grouped together  var pie = d3.layout.pie()  .value(function (d) { return d.pop; })  .sort(function (a, b) {  return d3.ascending(a.area, b.area);  }); |

**Final stretch of code involves first creating the svg arc object, which is based off of the radius value and will be used in the creation of the actual pie chart. Then comes the creation of the path items, which I believe are essentially the pie slices – several of the lines below are mostly the result of google searches to make D3 pie charts, I don’t think this would be at all easy to assembly by just reading API documentation. The class definition is mine, setting it equal to one of the four area values, which in the CSS will lead to common coloring for all countries in the same area. The mouseover() handles the cursor entering a pie slice, causing the ‘selected-path’ class to be temporarily added, which will lead to light-purple coloring due to fill/stroke color for that class value in the CSS. On mouse-out this class is simply removed. (The two commented out sections reflect my first approach of setting the class to a single value and then replacing that with correct ‘area’ class on mouseout – worked fine but I think newer solution is more elegant. The last two items are very similar to what was in Problem 1, adding some text to the title element in order to popup the country and population info on any slice that is hovered over.**

|  |
| --- |
| var arc = d3.svg.arc()  .outerRadius(radius);  var path = svg.selectAll('path')  .data(pie(data))  .enter()  .append('path')  .attr('d', arc)  .attr('class', function (d) { return d.data.area })  .on('mouseover', function (d) {  //d3.select(this).attr('class', 'purple');  d3.select(this).classed('selected-path', true);  })  .on('mouseout', function (d) {  //d3.selectAll('path').attr('class', function (d) {  // return d.data.area;  //});  d3.select(this).classed('selected-path', false);  })  .append("title")  .text(function (d) {  return d.data.country + '\n'  + 'Population (millions): ' + (d.data.pop / 1000000).toFixed(4);  });  }); |

**For CSS it again makes more sense to enter some inline comments to provide context to the sections.**

|  |
| --- |
| **\*\* lec09\_p2.css \*\***  /\*simple border for the main graphic\*/  .frame {  border: 1px solid black;  padding: 20px 20px;  }  /\*set stroke and fill for each area/continent to correct color  removing stroke lines will leave a light border between  country pie slices.  \*/  path.europe {  stroke: #FCEA32;  fill: #FCEA32; /\*yellow\*/  }  path.africa {  stroke: #30DCEC;  fill: #30DCEC; /\*blue\*/  }  path.asia {  stroke: #FC788D;  fill: #FC788D; /\*red\*/  }  path.america {  stroke: #95EE32;  fill: #95EE32; /\*green\*/  }  /\*keep after the standard path CSS definitions for the mouseover/out  behavior to work correctly. Preferably make the CSS selector  more precisely pathed to avoid dependency...but not tonight.  \*/  path.selected-path {  stroke: thistle;  fill: thistle; /\*light purple\*/  } |

**Final pie chart, mouse is hovering over country = India in the 7 to 9 o’clock position of the chart.**

