CS/INFO 3300; INFO 5100
Homework 5
Due 11:59pm Monday 3/19

Goals: Create a discretized bar plot from lists of numbers. Sample from distributions and see connections between distributions. Learn about the reliability (or lack thereof) of mean values. Compare colors based on analytical properties. Update properties interactively.

Distributions and histograms

There's an important result in statistics called the central limit theorem. It says that, under fairly mild conditions, the mean of a random sample from a distribution behaves like a Gaussian random variable, even if the distribution you're sampling from behaves nothing like a Gaussian. We're going to see if this is true for a number of different probability distributions.

1. Create a function `plotHistogram` that creates an SVG histogram for continuous (ie floating point) data. The function should take two arguments: a string representing an element id (eg "`#normal`") and an array of numbers. The body of the function should select the element with that id, append an SVG element inside it, construct a linear scale for the x-axis that is appropriate for the values in the data array, and then construct a histogram from the provided data array, with x- and y-axes. Use `d3.histogram()` to organize arrays into bins and use the d3 data join (ie `selectAll(), data(), enter()` functions) to create bars with line, rect, or path elements. All plots should have axes, but axis labels are not required. Axis ticks do not have to line up with bars. (30 pts)

2. **Normal distribution**. Use `d3.randomNormal(0.0, 1.0)` to generate a function called `gaussian`, which will return random values drawn from a normal (Gaussian) distribution with mean 0.0 and standard deviation 1.0. Create a function `gaussian1000` that will generate an array of 1000 samples from your `gaussian` function, *adding 1.0 to each sample*. For example, if the function returns 0.48, record 1.48 in the array. Generate an array from this function and use your `plotHistogram` function to create a density plot of this data. (5 pts)

3. Now create an array of length 1000, where each element is the mean of an array returned by your `gaussian1000` function (that is, generate the means of 1000 different arrays, each of length 1000). The function `d3.mean()` will be useful. Use your `plotHistogram` function to create a density plot of this array of means. Does it have roughly the same shape as the plot from Problem 2, and does it have the same x-scale? If not, how is it different? (5 pts)

4. **Lognormal distribution**. You can generate a sample from a lognormal distribution by generating a sample from a Gaussian distribution, and then exponentiating that value with

`Math.exp()`. Create a function `lognormal1000` that will generate an array of 1000 samples from a lognormal distribution. You can use your `gaussian` function from the previous problems (don't add 1.0 this time). Sample an array from this function and use your `plotHistogram` function to create a density plot of this data. (5 pts)

5. Create an array of length 1000, where each element is the mean of an array returned by your `lognormal1000` function (that is, generate the means of 1000 different arrays, each of length 1000). The function `d3.mean()` will be useful. Use your `plotHistogram` function to create a density plot of this array of means. Does it have roughly the same shape as the plot from Problem 2, and does it have the same x-scale? If not, how is it different? (5 pts)

6. **Exponential distribution**. You can generate a sample from this distribution with this expression:

```
–Math.log(Math.random())
```

Create a function `exponential1000` that will generate an array of 1000 numbers drawn from an exponential distribution. Sample an array from this function and use your `plotHistogram` function to create a density plot of this data. (5 pts)

7. (Seeing a pattern?) Create an array of length 1000, where each element is the mean of an array returned by your `exponential1000` function. Use your `plotHistogram` function to create a density plot of these means. Does the histogram of the distribution (from the previous question) look like the histogram of the mean of samples from the distribution? If not, how is it different? (5 pts)

8. **Gumbel distribution**. The Gumbel distribution is often used to estimate the probability of extreme events like floods. Just as you generated an exponential random variable by taking the negative log of a uniform random variable, you can generate a Gumbel random variable by taking the negative log of an exponential random variable and multiplying that value by 1.732455 (I am not making this up). Create a function `gumbel1000` that will generate an array of 1000 numbers drawn from a Gumbel distribution. Sample an array from this function and use your `plotHistogram` function to create a density plot. (5 pts)

9. Create an array of length 1000, where each element is the mean of an array returned by your `gumbel1000` function. Use your `plotHistogram` function to create a density plot of these means. Does the histogram of the distribution (from the previous question) look like the histogram of the mean of samples from the distribution? If not, how? (5 pts)

10. **Cauchy distribution**. Remember about those "mild conditions"? This one doesn't meet them, so expect this to look... different. You can generate a sample from this distribution with this expression:

```
gaussian() / gaussian()
```

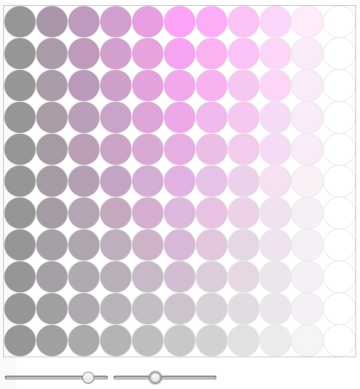
where `gaussian` is the Gaussian random variable generator function you created in Problem 2. The expression samples two independent Gaussian random variables and returns their *ratio*.

Create a function `cauchy1000` that will generate an array of 1000 numbers drawn from a Cauchy distribution. Sample an array from this function and use your `plotHistogram` function to create a density plot of this data. (5 pts)

11. Create an array of length 1000, where each element is the mean of an array returned by your `cauchy1000` function. Use your `plotHistogram` function to create a density plot of these means. Does the histogram of the means from this distribution look like the histograms of means from Problems 3, 5, 7, and 9? If not, how is it different? Pay particular attention to the x-axis. (5 pts)

HSLa Colors

We're used to indexing colors with RGB values. We can also use Hue, Saturation, and Lightness. The CSS function `hsla(h, s, l, a)` defines a color based on hue (0-360), saturation, lightness, and alpha or opacity (all 0.0-1.0). Here we'll look at the differences between these and add some interactive effects.



12. In a <p> tag, place an SVG element of height and width of at least 500px (you may add extra padding as needed). Write code to create an array of 121 objects that will represent a 11x11 grid, as shown to the right. Each object should have a `saturation` variable with value from 0 to 100 and a `lightness` variable with value from 0 to 100, evenly spaced in multiples of 10. Create a function `showCircles(hue, opacity)` that uses the d3 "data join" (ie `selectAll(), data(), enter(), attr(),` and `style()` functions) to create or modify one circle for each object in your list. Set the radius of each circle to 20 and the stroke to `#eee`. Set the location of each circle to create a grid based on the associated values: lightness for `y`, saturation for `x`, with centers 40px apart. Set the fill of each circle to an HSLa color specified by the circle's lightness and saturation, and the hue and opacity values passed to the function. You may want to use `d3.hsl()`. Add two slider inputs, one for hue and one for opacity. Use d3 to attach event listener functions to the "input" event for these sliders to call your `showCircles` function with the current values of the sliders. (20 pts)