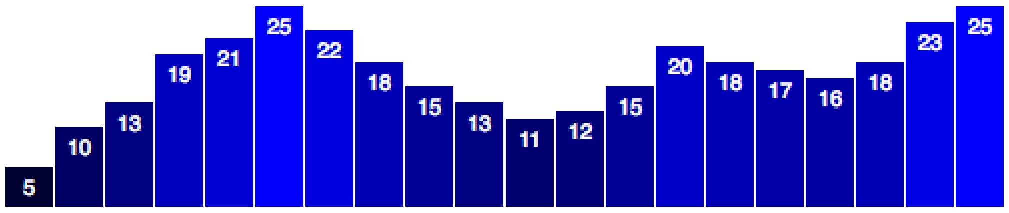
Transition and Motion – Interactivity

Modernizing the Bar Chart



The bar chart, as seen last

we used this static dataset:

```
var dataset = [ 5, 10, 13, 19, 21, 25, 22, 18, 15, 13,
                11, 12, 15, 20, 18, 17, 16, 18, 23, 25 ];
```

Since then, weâ□™ve learned how to write more flexible code, so our chart elements resize to accommodate different-sized datasets (meaning shorter or longer arrays) and different data values (smaller or larger numbers). We accomplished that flexibility using D3 scales, so Iâ□™d like to start by bringing our bar chart up to speed.

Ready? Okay, just give me a secâ€¦

Aaaaaand, done! Thanks for waiting.

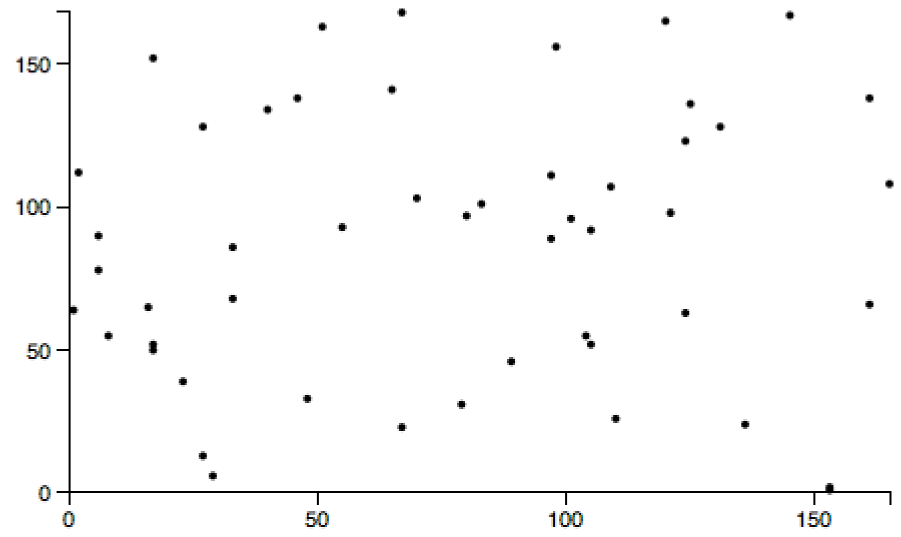
Figure 9-2. A scalable, flexible bar chart

To start, I adjusted the width ...

Updating Axes

The bar chart doesn’t have any axes, but our scatterplot from the last chapter

Click on this text to update the chart with new data values as many times as you like!



. Updated scatterplot, now with data updates and dynamic scales

To summarize the changes to the scatterplot:

- You can now click the text at top to generate and update with new data.
- Animated transitions are used after data updates.
- I eliminated the staggered delay, and set all transitions to occur over a full second (1,000 ms).
- Both the x- and y-axis scales are updated, too.
- Circles now have a constant radius.

Try clicking the text and watch all those little dots zoom around. Cute! I sort of wish they represented some meaningful information, but hey, random data can be fun, too.

What’s *not* happening yet is that the axes aren’t updating. Fortunately, that is simple to do.

First, I am going to add the class names `x` and `y` to our x- and y-axes, respectively. This will help us select those axes later:

```
//Create x-axis
svg.append("g")
    .attr("class", "x axis")    // <-- Note x added here
    .attr("transform", "translate(0," + (h - padding) + ")")
    .call(xAxis);

//Create y-axis
svg.append("g")
    .attr("class", "y axis")    // <-- Note y added here
    .attr("transform", "translate(" + padding + ",0)")
    .call(yAxis);
```

Then, down in our click function, we simply add:

```
//Update x-axis
svg.select(".x.axis")
    .transition()
    .duration(1000)
    .call(xAxis);

//Update y-axis
svg.select(".y.axis")
    .transition()
    .duration(1000)
    .call(yAxis);
```

For each axis, we do the following:

1. Select the axis.
2. Initiate a transition.
3. Set the transition's duration.
4. Call the appropriate axis generator.

Remember that each axis generator is already referencing a scale (either `xScale` or `yScale`). Because those scales are being updated, the axis generators can calculate what the new tick marks should be.

Open up *20_axes_dynamic.html* and give it a try.

Once again, `transition()` handles all the interpolation magic for you—watch those ticks fade in and out. Just beautiful, and you barely had to lift a finger.

each() Transition Starts and Ends

There will be times when you want to make something happen at the start or end of a transition. In those times, you can use `each()` to execute arbitrary code for each element in the selection.

`each()` expects two arguments:

- Either `"start"` or `"end"`
- An anonymous function, to be executed either at the start of a transition, or as soon as it has ended

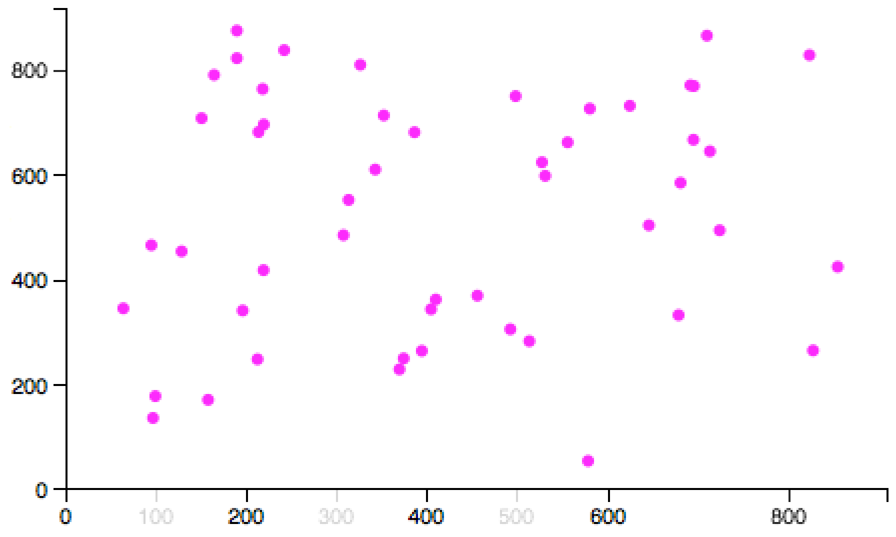
For example, here is our circle-updating code, with two `each()` statements added:

```
//Update all circles
svg.selectAll("circle")
   .data(dataset)
   .transition()
   .duration(1000)
   .each("start", function() {      // <-- Executes at start of transition
       d3.select(this)
         .attr("fill", "magenta")
         .attr("r", 3);
   })
   .attr("cx", function(d) {
```

```
             return xScale(d[0]);
        })
        .attr("cy", function(d) {
             return yScale(d[1]);
        })
        .each("end", function() {          // <-- Executes at end of transition
            d3.select(this)
              .attr("fill", "black")
              .attr("r", 2);
        });
```

You can see this in action in *21_each.html*.

Now you click the trigger, and immediately each circle’s fill is set to magenta, and its radius is set to 3 Then the transition is run, per usual. When complete, the fills and radii are restored to their original values.



. Hot pink circles, midtransition

Something to note is that within the anonymous function passed to `each()`, the context of `this` is maintained as “the current element.” This is handy because then `this` can be referenced with the function to easily reselect the current element and modify it, as done here:

```
        .each("start", function() {
            d3.select(this)                // Selects 'this', the current element
              .attr("fill", "magenta")     // Sets fill of 'this' to magenta
              .attr("r", 3);               // Sets radius of 'this' to 3
        })
```

Warning: Start carefully

You might be tempted to throw another transition in here, resulting in a smooth fade from black to magenta. Don’t do it! Or do it, but note that this will break:

```
        .each("start", function() {
            d3.select(this)
```

```
      .transition()               // New transition
      .duration(250)              // New duration
      .attr("fill", "magenta")
      .attr("r", 3);
})
```