Introduction to D3

Maneesh Agrawala

CS 448B: Visualization
Fall 2021

1

Reading Response Questions/Thoughts

How do you balance details with overarching views? How to decide when there is too much vs. too little data in a view?

In the VDQI reading, it was stated that we can increase data density in two ways: increasing the number of data points or decreasing the size of the graph... When it is advisable to reduce the size of our graph to increase the density? ... Are there any quantitative metrics to assess visualization interpretability?

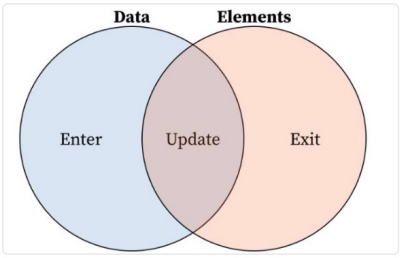
From an accessibility standpoint, I wonder if there are ways to facilitate micro-macro reading through haptic and audio means?

And more generally, what are instances where poor design choices on the micro level can obscure the types of insights gained at the macro level?

When is it appropriate to exclude outlying data points? For datasets of the Challenger launch tests, the outliers were everything. All data is prone to some degree of variation, so how much significance should we place on outliers?

2

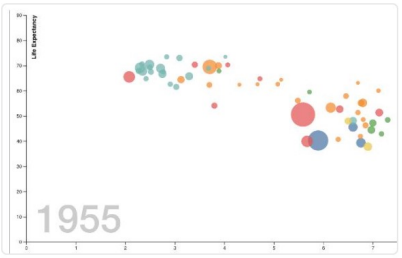
D3 Notebooks



Team • Published

Introduction to D3

You republished 14 hours ago



You • Published

Making D3 Charts Interactive

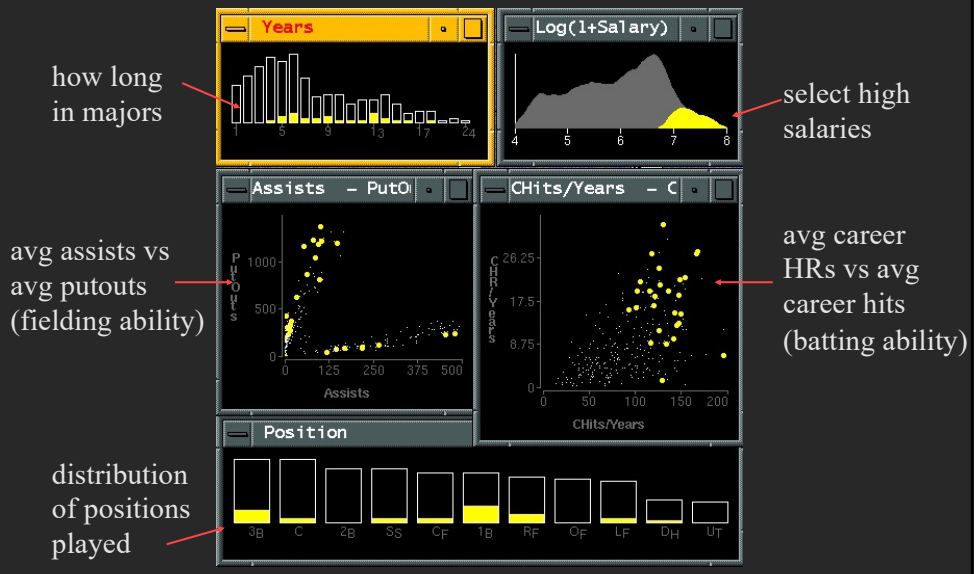
You republished 14 hours ago

3

Last Time: Interaction

4

Baseball statistics [from Wills 95]



6

Dynamic Queries

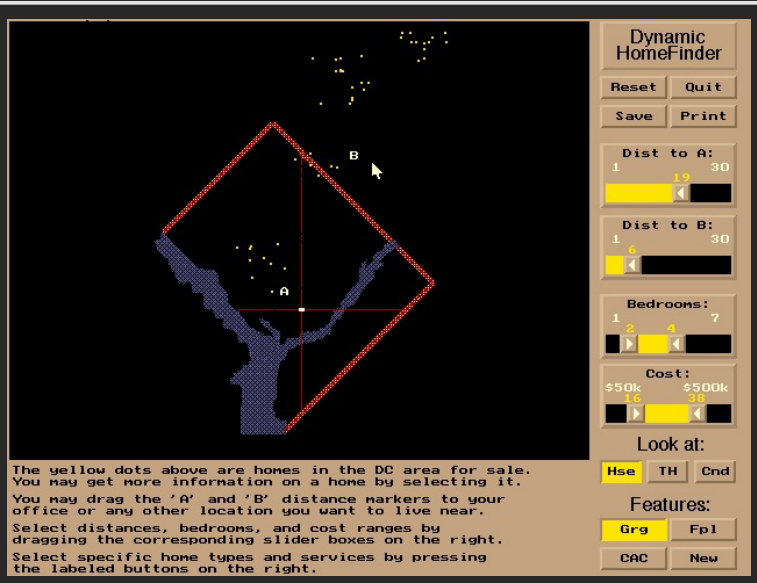
7

Query and results

SELECT house
FROM east bay
WHERE price < 1,000,000 AND bedrooms > 2
ORDER BY price

8

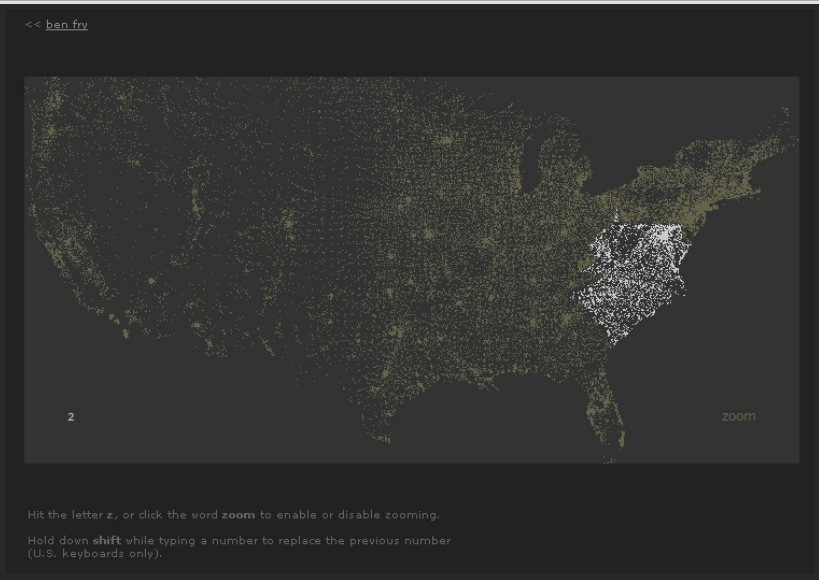
HomeFinder



[Ahlberg and Schneiderman 92]

10

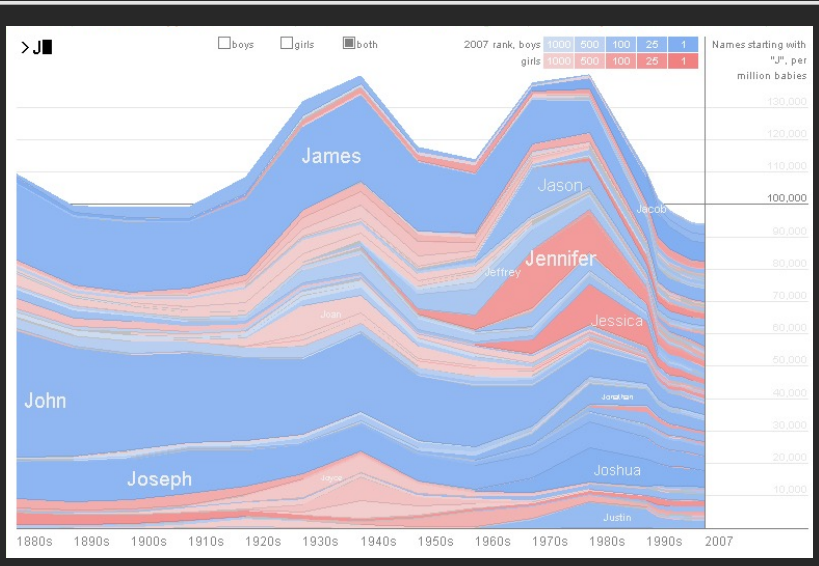
Zipdecode [from Fry 04]



https://benfry.com/zipdecode/

18

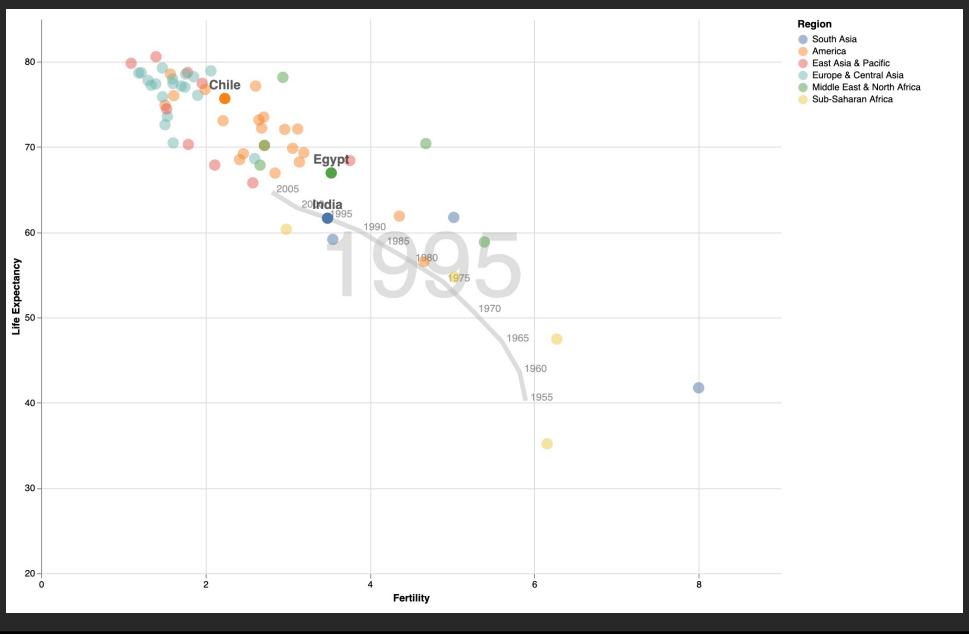
NameVoyager



http://www.babynamewizard.com/voyager

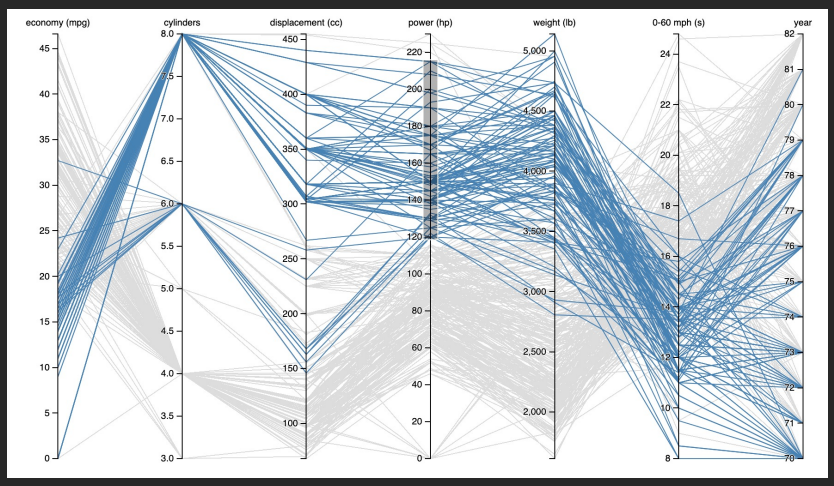
19

DimpVis [Kondo 14]



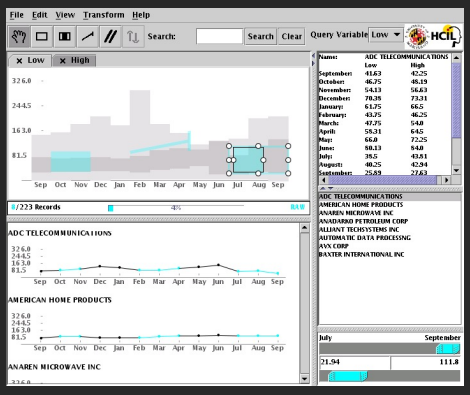
20

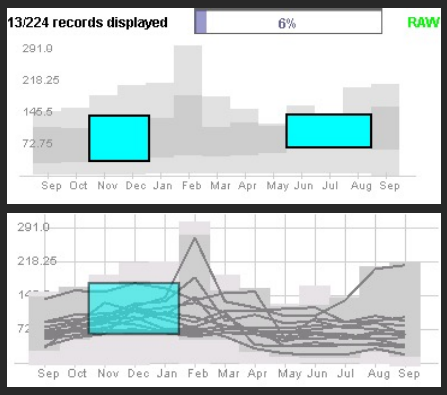
Parallel Coordinates [Inselberg]



21

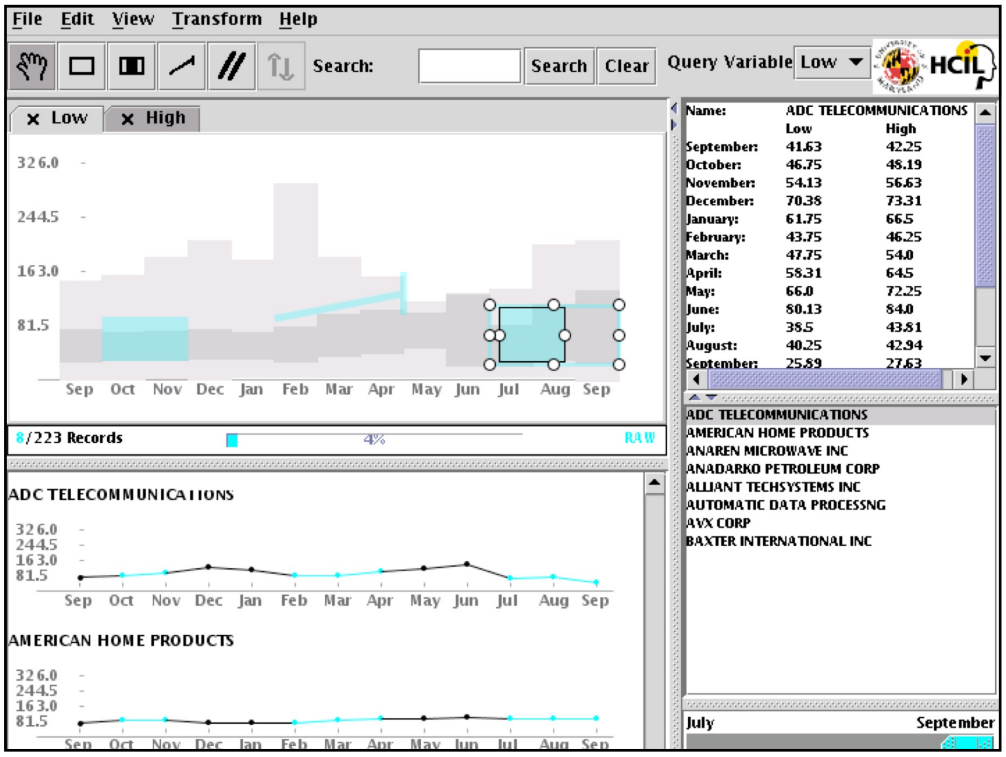
TimeSearcher [Hochheiser & Schneiderman 02]





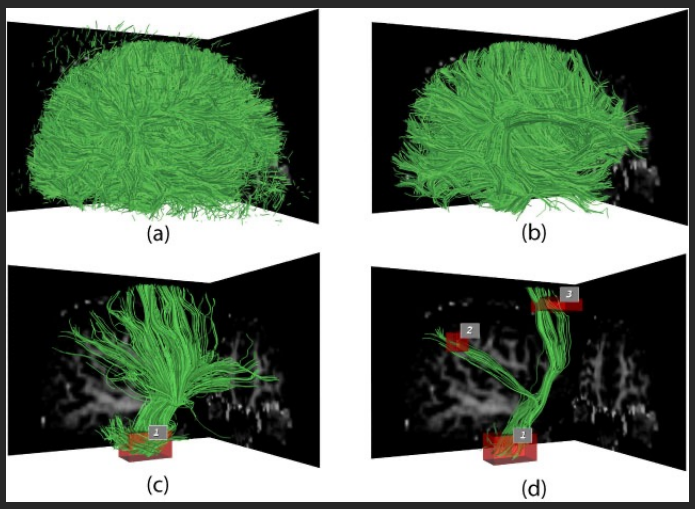
Based on Wattenberg's [2001] idea for sketch-based queries of time-series data

22



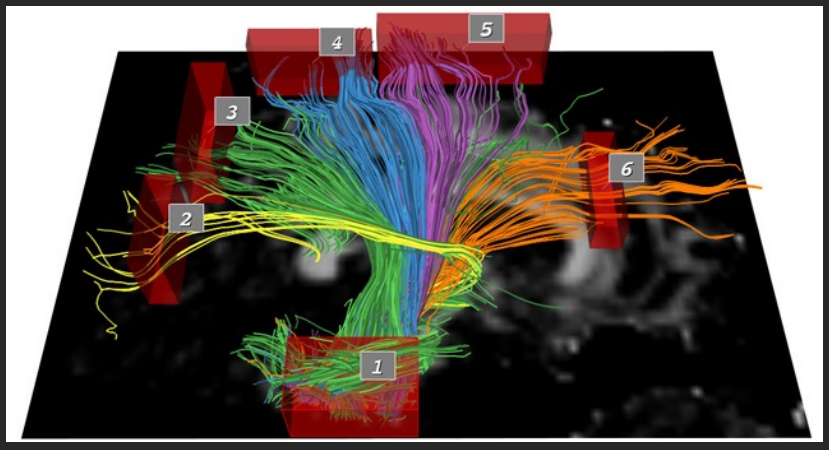
23

3D dynamic queries [Akers et al. 04]



24

3D dynamic queries [Akers et al. 04]



25

Pros and cons

Pros

- Controls useful for both novices and experts
- Quick way to explore data

Cons

- Simple queries
- Lots of controls
- Amount of data shown limited by screen space

26

Summary

Good visualizations are task dependent

- Pick the right interaction technique

Fundamental interaction techniques

- Selection, Brushing & Linking, Dynamic Queries

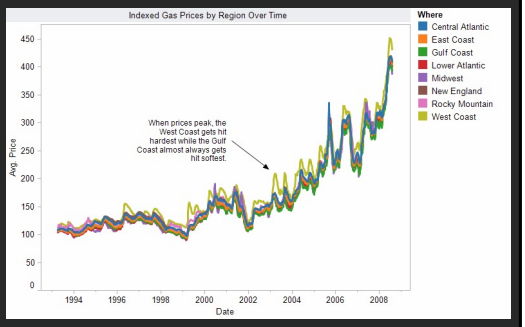
27

Announcements

28

A2: Exploratory Data Analysis

Use Tableau or Vega-Lite to formulate & answer questions



First steps

- Step 1: Pick domain & data
- Step 2: Pose questions
- Step 3: Profile data
- Iterate as needed

Create visualizations

- See different views of data
- Refine questions

Author a report

- Screenshots of most insightful views (8+)
- Include titles and captions for each view

Due before class on Oct 11, 2021

29

Assignment 3: Dynamic Queries

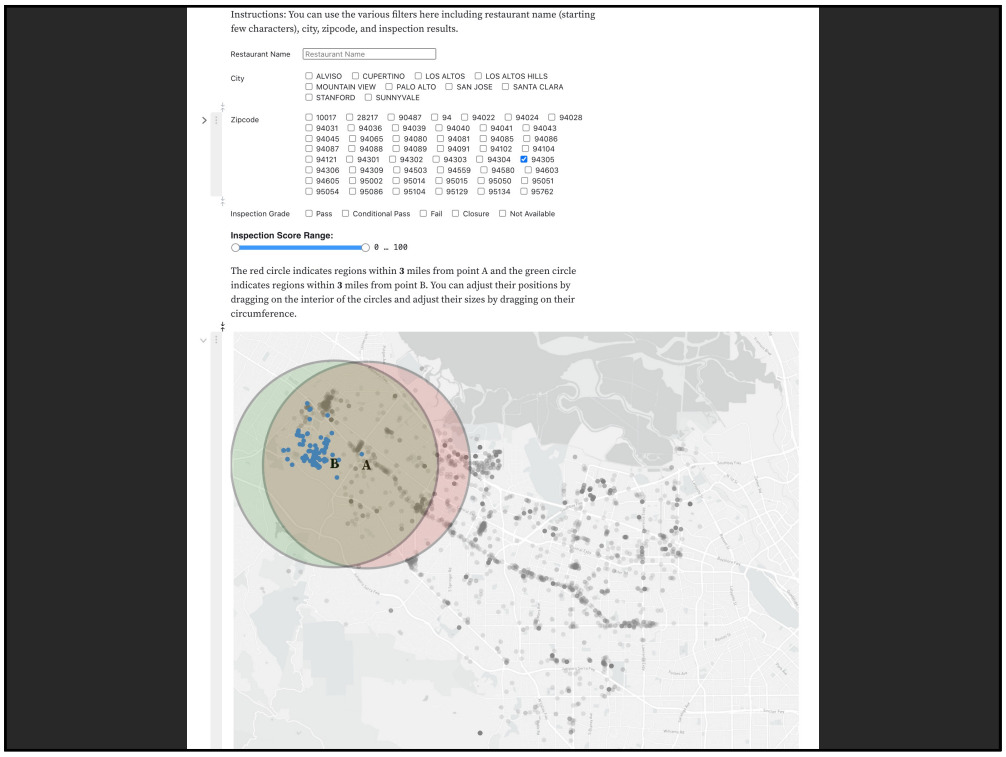
Create a small interactive dynamic query application similar to HomeFiner, but for restaurants data.

1. **Implement interface**
2. **Submit the application and a short write-up on canvas**

Can work alone or in pairs

Due before class on Oct 25, 2021

30



31

Introduction to D3

32

What is D3?

D3: "Data-Driven Documents"

Data visualization built on top of HTML, CSS, JavaScript, and SVG

Pros:

- Highly-customizable
  - Developing and debugging tools
  - Documentation, resources, community
- Integrates with the web!

Cons:

- Very "low-level"

33

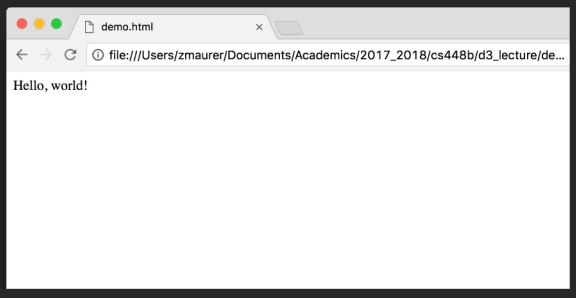
hello-world.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
</head>

<body>
  Hello, world!
</body>

</html>
```

34

hello-world.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
</head>

<body>
  Hello, world!
</body>

</html>
```



35

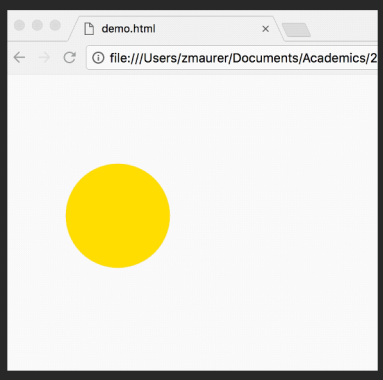
hello-svg.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <style> /* CSS */ </style>
</head>

<body>
    <svg width="960" height="500">
      <circle cx='120' cy='150' r='60' style='fill: gold;'>
          <animate
            attributeName='r'
            from='2' to='80' begin='0' dur='3'
            repeatCount='indefinite' />
      </circle>
    </svg>
</body>
</html>
```

38

hello-svg.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <style> /* CSS */ </style>
</head>

<body>
    <svg width="960" height="500">
      <circle cx='120' cy='150' r='60' style='fill:
          gold;'>
          <animate attributeName='r'
            from='2' to='80' begin='0' dur='3'
            repeatCount='indefinite' />
      </circle>
    </svg>
</body>
</html>
```



39

DOM: Document Object Model

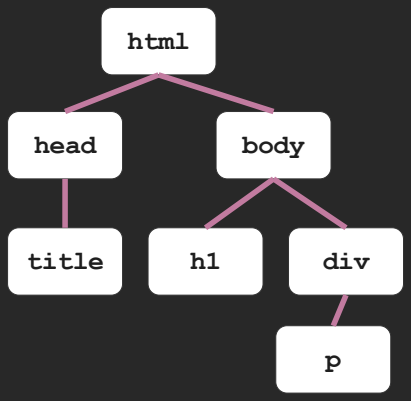
```
<html>
        <head>

        <title></title>
        </head>
        <body>
                <h1></h1>
                <div>

        <p></p>
                </div>
        </body>
</html>
```

[Adapted from Victoria Kirst's cs193x slides.]

42

DOM: Document Object Model

```
<html>
        <head>

        <title></title>
        </head>
        <body>
                <h1></h1>
                <div>

        <p></p>
                </div>
        </body>
</html>
```

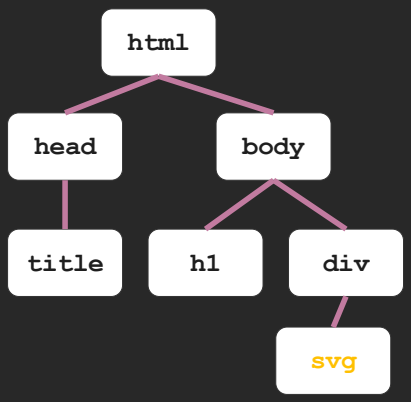


[Adapted from Victoria Kirst's cs193x slides.]

43

DOM: Document Object Model

```
<html>
        <head>

        <title></title>
        </head>
        <body>
                <h1></h1>
                <div>

        <svg></svg>
                </div>
        </body>
</html>
```



[Adapted from Victoria Kirst's cs193x slides.]

44

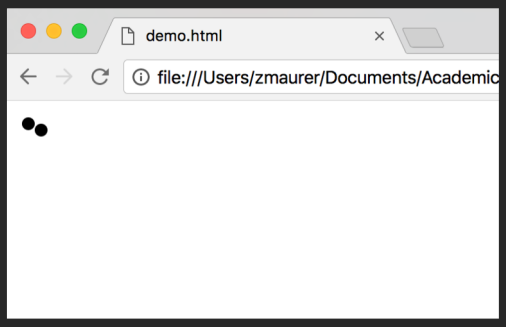
hello-d3.html

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8">
 <style> /* CSS */ </style>
</head>

<body>
 <script src="https://d3js.org/d3.v7.min.js"></script>
 <script>

  // JavaScript code that handles the logic of adding SVG elements
  // that make up the visual building blocks of your data visualization

 </script>
</body>
</html>
```

45

D3: Selection

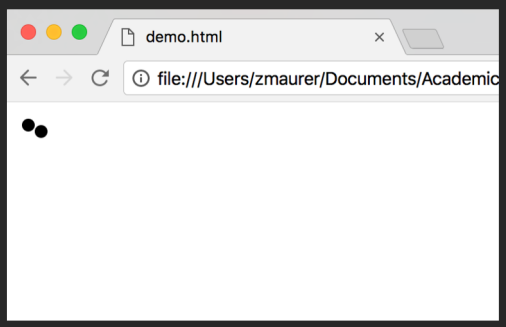
```
<html>
…
<svg width="960" height="500">
  <circle cx="10" cy="10" r="5"></circle>
  <circle cx="20" cy="15" r="5"></circle>
</svg>
```



46

D3: Selection

```
<html>
…
<svg width="960" height="500">
  <circle cx="10" cy="10" r="5"></circle>
  <circle cx="20" cy="15" r="5"></circle>
</svg>
```




```
<script>

// select all SVG circle
    elements
var circles =
    d3.selectAll("circle");

</script>
```

47

D3: Selection & Manipulation

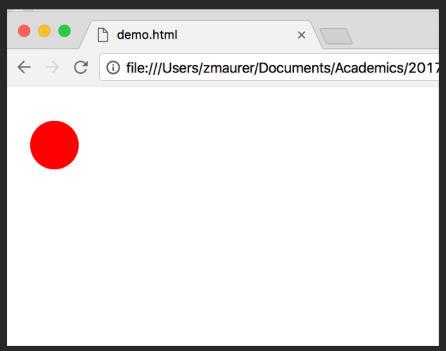
```
<html>
...
<svg width="960" height="500">
  <circle cx="10" cy="10" r="5"></circle>
  <circle cx="20" cy="15" r="5"></circle>
</svg>
```

```
<script>

// select all SVG circle
    elements
var circles =
    d3.selectAll("circle");


// set attributes and styles
circles.attr("cx", 40);
circles.attr("cy", 50);
circles.attr("r", 24);
circles.style("fill", "red");

</script>
```

48

D3: Selection & Manipulation

```
<html>
...
<svg width="960" height="500">
  <circle cx="10" cy="10" r="5"></circle>
  <circle cx="20" cy="15" r="5"></circle>
</svg>
```




```
<script>

// select all SVG circle
    elements
var circles =
    d3.selectAll("circle");


// set attributes and styles
circles.attr("cx", 40);
circles.attr("cy", 50);
circles.attr("r", 24);
circles.style("fill", "red");

</script>
```

49

D3: Selection & Manipulation

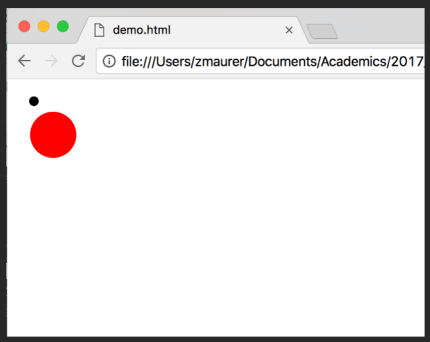
```
<html>
...
<svg width="960" height="500">
  <circle cx="10" cy="10" r="5"></circle>
  <circle cx="20" cy="15" r="5"></circle>
</svg>
```

```
<script>

// select all SVG circle
    elements
var circles =
    d3.select("circle");


// set attributes and styles
circles.attr("cx", 40);
circles.attr("cy", 50);
circles.attr("r", 24);
circles.style("fill", "red");

</script>
```

50

D3: Selection & Manipulation

```
<html>
...
<svg width="960" height="500">
  <circle cx="10" cy="10" r="5"></circle>
  <circle cx="20" cy="15" r="5"></circle>
</svg>
```

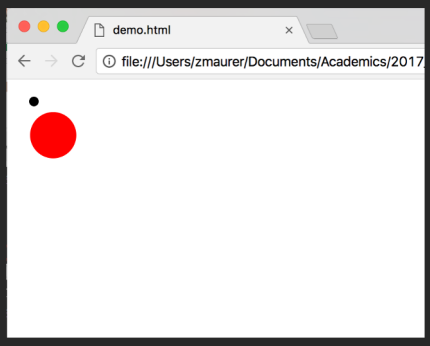



```
<script>

// select all SVG circle
    elements
var circles =
    d3.select("circle");


// set attributes and styles
circles.attr("cx", 40);
circles.attr("cy", 50);
circles.attr("r", 24);
circles.style("fill", "red");

</script>
```

51

D3: Selection & Manipulation

```
<html>
...
<svg width="960" height="500">
  <circle cx="10" cy="10" r="5"></circle>
  <circle cx="20" cy="15" r="5"></circle>
</svg>
```




```
<script>

// all together!!
d3.select("circle")
      .attr("cx", 40)
      .attr("cy", 50)
      .attr("r", 24)
      .style("fill", "red");

</script>
```

52

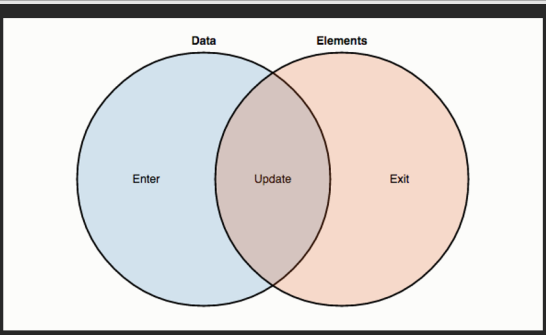
Binding Data & Joining DOM Elements

1. China: 1303182268
2. India: 1080264388
3. United States: 295734134
4. Indonesia: 218465000
5. Brazil: 186112794
6. Pakistan: 162419946
7. Bangladesh: 144319628
8. Nigeria: 128765768
9. Japan: 127417244
10. Mexico: 106202903

```
{
  const ol = d3.create('ol');

  ol.selectAll('li') // select all list elements (orange circle above)
    .data(listData)  // bind all our data values (blue circle above)
    .join(
      enter  => enter.append('li'), // append an li element for each entering item
      update => update,             // do nothing with items that match an existing element
      exit   => exit.remove()       // remove li elements whose backing data is now gone
    )
      .text(d => `${d.country}: ${d.pop}`)

  return ol.node();
}
```

53

Binding Data & Joining DOM Elements



A *join* creates three sub-selections:

- ***Enter***: selection containing placeholders for every data value that did not have a corresponding DOM element in the original selection
- ***Update***: selection containing *existing* DOM elements that match a bound data value
- ***Exit***: selection that also contains *existing* DOM elements, but for which a matching data value was not found

55

Binding Data & Joining DOM Elements

Exercise

Modify the *enter*, *update*, and *exit* functions in the code below such that entering items are colored green, updating items are colored blue, and exiting items are *not* removed but rather colored red.

1. China: 1303182268
2. India: 1080264388
3. United States: 295734134
4. Indonesia: 218465000
5. Brazil: 186112794
6. Pakistan: 162419946
7. Bangladesh: 144319628
8. Nigeria: 128765768
9. Japan: 127417244
10. Mexico: 106202903

```
undefined
```

```
{
  const ol = d3.select('ol#enter-update-exit');

  // use a new dataset, manipulable with the variables defined below
  const newData = gapminder
    .filter(d => d.year === year)
    .sort((a, b) => b.pop - a.pop)
    .slice(0, n);

  ol.selectAll('li') // select all list elements (orange circle above)
    .data(newData)   // bind all our data values (blue circle above)
    .join(
      enter  => enter.append('li').style('color', 'green'),
      update => update.style('color', 'blue'),
      exit => exit.remove()
    )
    .text(d => `${d.country}: ${d.pop}`);
}
```

You can use the variables below to change the elements in the `newData` extracted in the cell above to test your changes to `enter`, `update`, and `exit`. *Note that we are using Observable's ability to automatically figure out the dependency structure between cells so that changes to the cells below correctly update the cells above.*

```
year = 2005
```

```
year = 2005
```

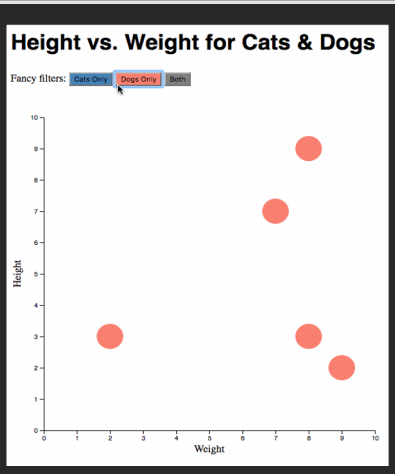
```
n = 10
```

```
n = 10
```

56

Let's make a scatterplot 🐱 🐶

```
id,animal,weight,height,name
1,cat,10,3,phyllis
2,cat,3,3,oreo
3,cat,9,9,sam
4,cat,3,5,dog
5,cat,6,5,fred
6,cat,5,6,jane
7,cat,1,8,esmerelda
8,dog,9,2,garfield
9,dog,8,9,alpha
10,dog,7,7,omega
11,dog,2,3,zeta
12,dog,8,3,cupcake
```



https://observablehq.com/@stanfordvis/lets-make-a-scatterplot

57