Interactive Visualization

Héctor Corrada Bravo

University of Maryland, College Park, USA
CMSC320: 2018-05-08
Why Interactivity?

*Reduce data dimension:* allow user to explore large datasets by quickly switching between dimensions

*Overview first, zoom and filter, details on demand:* Provide big picture, let the user explore details as they desire

*Linked views for high dimensions:* There is a limit to the number of aesthetic mappings in a single graphic, make multiple graphics but link data objects between them
Examples

Politics: http://www.nytimes.com/interactive/2012/11/02/us/politics/paths-to-the-white-house.html?_r=0


Web-based interactive visualization

Take advantage of HTML document description and the Document Object Model interface to bind data to page elements.

- **Shiny**: bind data to controls
- **Data-driven Documents (d3.js)**: bind data to svg elements directly
HTML and DOM

Web pages are structured using Hypertext Markup Language

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>Page Title</h1>
    <p>This is a really interesting paragraph.</p>
  </body>
</html>
```
HTML and DOM

Web pages are structured using Hypertext Markup Language

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>Page Title</h1>
    <p>This is a really interesting paragraph.</p>
  </body>
</html>
```
HTML and DOM

Web pages are structured using Hypertext Markup Language

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Page Title</title>
  </head>
  <body>
    <h1>Page Title</h1>
    <p>This is a really interesting paragraph.</p>
  </body>
</html>
```
CSS

Cascading Style Sheets are used to style elements in the DOM.

```css
body {
    background-color: white;
    color: black;
}
```
CSS

In general:

selectorA,
selectorb,
selectorC {
  property1: value;
  property2: value;
  property3: value;
}
SVG

Scalable Vector Graphics (SVG) is special element used to create graphics with text.

```xml
<svg width="50" height="50">
  <circle cx="25" cy="25" r="22" fill="blue" stroke="gray" stroke-width="2"/>
</svg>
```
SVG

Elements have geometric attributes and style attributes.

```xml
<circle cx="250" cy="25" r="25"/>
```

cx: x-coordinate of circle center
cy: y-coordinate of circle center
r: radius of circle
SVG

Elements have *geometric* attributes and *style* attributes.

```xml
<rect x="0" y="0" width="500" height="50"/>
```

**x**: x-coordinate of left-top corner

**y**: y-coordinate of left-top corner

**width**, **height**: width and height of rectangle
SVG

*style* attributes

```html
<circle cx="25" cy="25" r="22" fill="yellow" stroke="orange" stroke-width="5"/>
```

can be styled by class as well

```css
svg .pumpkin {
    fill: yellow;
    stroke: orange;
    stroke-width: 5;
}
```

```html
<circle cx="25" cy="25" r="22" class="pumpkin">
```
Shiny and D3

Shiny: construct DOM and bind data (variables for example) to elements (a slide control for example) [http://shiny.rstudio.com](http://shiny.rstudio.com)

D3: bind data to SVG element attributes (position, size, color, transparency, etc.) [http://d3js.org](http://d3js.org)
Reactivity

Interactivity and binding in Shiny achieved using *reactive programming*. Where objects *react* to changes in other objects.
Reactivity

Example:

```r
code

shinyServer(function(input, output) {
  output$plotOut <- renderPlot({
    hist(faithful$eruptions, breaks = as.numeric(input$nBreaks))
    if (input$individualObs)
      rug(faithful$eruptions)
  })

  output$tableOut <- renderTable({
    if (input$individualObs)
      faithful
    else
      NULL
  })
})
```

Diagram:

- input$nBreaks to output$plotOut
- input$individualObs to output$plotOut
- input$individualObs to output$tableOut
Reactivity

With intermediate objects:

```r
fib <- function(n) ifelse(n<3, 1, fib(n-1)+fib(n-2))

shinyServer(function(input, output) {
  currentFib <- reactive({
    fib(as.numeric(input$n))
  })

  output$nthValue <- renderText({ currentFib() })
  output$nthValueInv <- renderText({ 1 / currentFib() })
})
```

Here is the new graph structure:
Reactivity

A standard paradigm for interactive (event-driven) application development

A nice review paper: http://dl.acm.org/citation.cfm?id=2501666
Binding data to graphical elements

With Shiny we can bind data objects to document elements. More examples: http://shiny.rstudio.com/gallery/

We can also bind data directly to \textit{graphical} elements since using SVG these are also document elements (D3).
D3 Alternatives

- If you want to use a toolkit of standard charts based on d3: NVD3
- An alternative declarative library: Vega
D3 and R

- We saw previously that D3 can access external data through JSON.
- That's how we can pass data from R to the Javascript browser.
D3 and R

- **rCharts**: Most mature. Provides binding between R and a small set of javascript viz libraries.
- **ggvis**: Uses grammar of graphics like ggplot2, bindings to Vega to define JS charts.
- **htmlwidgets** a formalization of how to bind R to JS libraries.
- **Roll your own**