Data visualization

COSC 480B

Reyan Ahmed

rahmed1@colgate.edu

Lecture 7

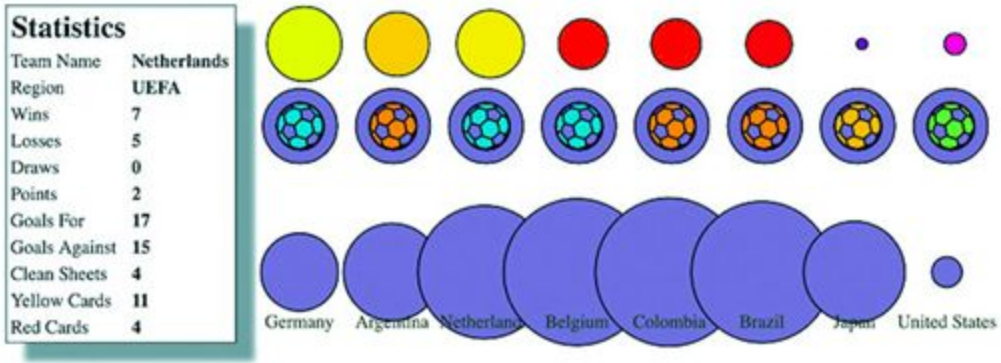
Data-driven design and interaction

Overview

- Enabling interactivity for graphical elements
- Working with color effectively
- Loading traditional HTML for use as pop-ups
- Loading external SVG icons into charts

Overview

This chapter covers loading HTML from an external file and updating it, as well as loading external images for icons, animating transitions, and working with color



Project architecture

Data:

```
"team","region","win","loss","draw","points","gf","ga","cs","yc","rc"
Germany,UEFA,7,6,0,1,19,18,4,14,4,6,0
Argentina,CONMEBOL,7,5,1,1,16,8,4,4,4,8,0
Netherlands,UEFA,7,5,0,2,17,15,4,11,4,11,0
Belgium,UEFA,5,4,1,0,12,6,3,3,2,7,1
Colombia,CONMEBOL,5,4,1,0,12,12,4,8,2,5,0
Brazil,CONMEBOL,7,3,2,2,11,11,14,-3,1,14,0
Japan,AFC,3,0,2,1,1,2,6,-4,1,4,0
United States,CONCACAF,4,1,2,1,4,5,6,-1,0,4,0
```

Project architecture

- Resources
  - SVG files, we can consider as code
- Images
  - PNG files
  - They are static

Project architecture

d3ia.css

```
text {
  font-size: 10px;
  text-anchor: middle;
  fill: #4f442b;

}
g > text.active {
  font-size: 30px;
}
circle {
  fill: #75739F;
  stroke: black;
  stroke-width: 1px;
}
circle.active {
  fill: #FE9922;
}
circle.inactive {
  fill: #C4B9AC;
}
```

Project architecture

- CSS
  - Remember that for text coloring in in SVG we need to use fill
  - The greater than sign means immediate descendent

```
<div class='outer'>
    <div class="middle">
        <div class="inner">...</div>
    </div>
    <div class="middle">
        <div class="inner">...</div>
    </div>
</div>
```

```
.outer > div {
    ...
}
```

- Here the style will be applied only to middle class

Project architecture

External libraries:

- soccerviz.js will be developed by us
- colorbrewer.js is a library available for selecting colors

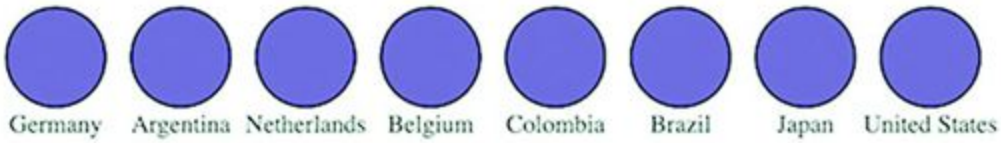
Project architecture

d3ia_2.html

```
<html>
<head>
  <title>D3 in Action Examples</title>
  <meta charset="utf-8" />
  <link type="text/css" rel="stylesheet" href="d3ia.css" />
</head>
<script src="d3.v4.min.js"></script>
<script src="colorbrewer.js"></script>
<script src="soccerviz.js"></script>
<body onload="createSoccerViz()">
<div id="viz">
<svg style="width:500px;height:500px;border:1px lightgray
solid;" />
</div>
<div id="controls" />
</body>
</html>
```

Project architecture

Circles and labels created from a CSV representing 2014 World Cup statistics.



Project architecture

soccerviz.js

```
function createSoccerViz() {
  d3.csv("worldcup.csv", data => {overallTeamViz(data)})

function overallTeamViz(incomingData) {
  d3.select("svg")
    .append("g")
    .attr("id", "teamsG")
    .attr("transform", "translate(50,300)")
    .selectAll("g")
    .data(incomingData)
    .enter()
    .append("g")
    .attr("class", "overallG")
    .attr("transform", (d, i) =>"translate(" + (i * 50) + ", 0)")
  var teamG = d3.selectAll("g.overallG");
}
```

```
teamG
  .append("circle")
  .attr("r", 20)
 teamG
  .append("text")
  .attr("y", 30)
  .text(d => d.team)
  }
```

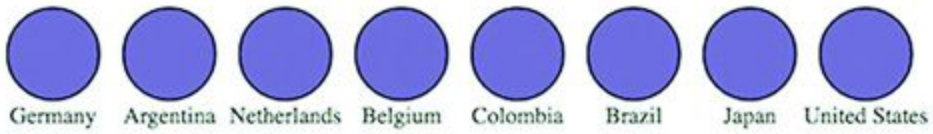
Interactive style and DOM

Instead of creating buttons manually, we add buttons depending on the data size.

```
const dataKeys = Object.keys(incomingData[0])
  .filter(d => d !== "team" && d !== "region")
d3.select("#controls").selectAll("button.teams")
   .data(dataKeys).enter()
   .append("button")
   .on("click", buttonClick)
.html(d => d);
function buttonClick(datapoint) {
  var maxValue = d3.max(incomingData, d =>
parseFloat(d[datapoint]))
var radiusScale = d3.scaleLinear()
  .domain([ 0, maxValue ]).range([ 2, 20 ])
d3.selectAll("g.overallG").select("circle")
   .attr("r", d => radiusScale(d[datapoint]))
}
```

Interactive style and DOM

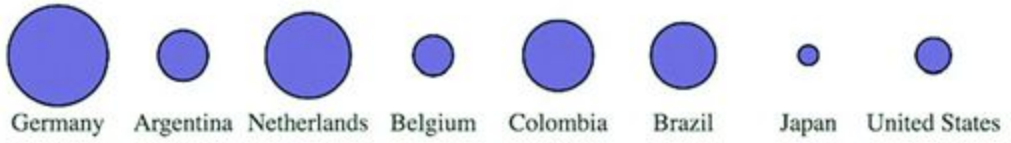
Buttons for each numerical attribute are appended to the controls div behind the viz div. When a button is clicked, the code runs buttonClick.



win loss draw points gf ga cs yc rc

Interactive style and DOM

Our initial buttonClick function resizes the circles based on the numerical value of the associated attribute. The radius of each circle reflects the number of goals scored against each team, kept in the ga attribute of each datapoint.



Interactive style and DOM

To check whether a team is in the same fifa group:

```
teamG.on("mouseover", highlightRegion);
function highlightRegion(d) {
   d3.selectAll("g.overallG").select("circle")
     .attr("class", p => p.region === d.region ? "active" : "inactive")
}
```

Interactive style and DOM

The effect of our initial highlightRegion selects elements with the same region attribute and colors them orange, while coloring gray those that aren’t in the same region.

Interactive style and DOM

When user removes the mouse set back the original color.

```
teamG.on("mouseout", function() {
  d3.selectAll("g.overallG")
  .select("circle").classed("inactive", false).classed("active", false)
})
```

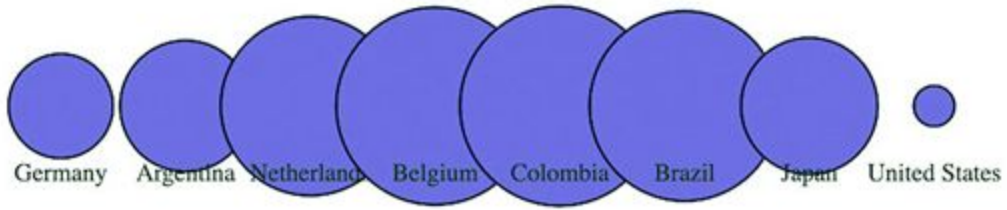
Interactive style and DOM

To avoid jarring use delay:

```
d3.selectAll("g.overallG").select("circle").transition().duration(1000)
  .attr("r", d => radiusScale(d[datapoint]))
```

Interactive style and DOM

A screenshot of your data visualization in the middle of its initial drawing, showing the individual circles growing to an exaggerated size and then shrinking to their final size in the order in which they appear in the bound dataset.

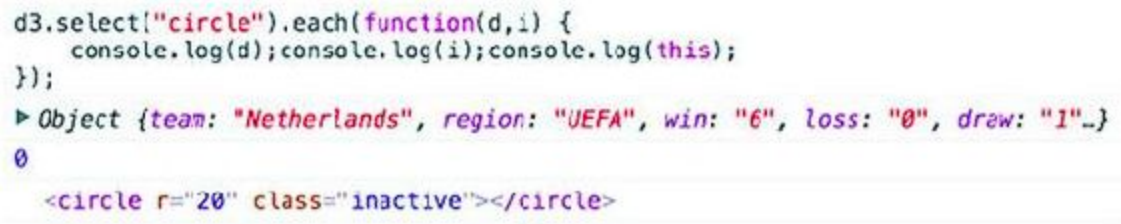


Interactive style and DOM

```
teamG
   .append("circle").attr("r", 0)
   .transition()
   .delay((d, i) => i * 100)
   .duration(500)
   .attr("r", 40)
   .transition()
   .duration(500)
   .attr("r", 20)
```

Interactive style and DOM

The console results of inspecting a selected element, which show first the datapoint in the selection, then its position in the array, and then the SVG element itself.




```
d3.select("circle").each(function(d,i) {
    console.log(d);console.log(i);console.log(this);
})
```

Interactive style and DOM

The results of running the node function of a selection in the console, which is the DOM element itself—in this case, an SVG <circle> element.

```
d3.select("circle").node()
  <circle r="20" class="inactive"></circle>
```

Interactive style and DOM

```
teamG.on("mouseover", highlightRegion)
function highlightRegion(d,i) {
   d3.select(this).select("text").classed("active", true).attr("y", 10)
   d3.selectAll("g.overallG").select("circle").each(function (p) {
    p.region == d.region ?
       d3.select(this).classed("active",true) :
       d3.select(this).classed("inactive",true)
   })                                          1
}
```

- 1 By turning on “active” class for the <g> that we hover over, we take advantage of the “g > text.active” rule in CSS that makes any text elements in that <g> increase their font size

Interactive style and DOM

```
teamG.on("mouseout", unHighlight)
function unHighlight() {
  d3.selectAll("g.overallG").select("circle").attr("class", "")
  d3.selectAll("g.overallG").select("text")
  .classed("active", false).attr("y", 30)
}
```

Interactive style and DOM


```
function highlightRegion (d) {
 d3.select(this).select("text").classed("active", true).attr("y",
10);
    d3.selectAll("g.overallG").select("circle")
       .each(function (p) {
         p.region == d.region ?
              d3.select(this).classed("active", true) :
              d3.select(this).classed("inactive", true);
       });
 this.parentElement.appendChild(this);
}
```

Interactive style and DOM

- New in D3v4 are several helper functions to let you bump elements up and down in the DOM: selection.raise and selection.lower.

```
d3.select("g.overallG").raise()
d3.select("g.overallG").lower()
```

- When mouseover triggers the text overlaps with neighbor circles, hence mouseout does not works properly. In that case use this:

```
teamG.select("text").style("pointer-events","none");
```

Interactive style and DOM

- For reference: http://en.wikipedia.org/wiki/Web_colors#X11_color_names

```
"rgb(255,0,0)"          1
"#ff0000"             2
"red"               3
```

- 1 RGB, or red-green-blue, encoded color
- 2 Hex, or hexadecimal, formatted
- 3 CSS3 web color name

Interactive style and DOM

```
teamColor = d3.rgb("red");
teamColor = d3.rgb("#ff0000");
teamColor = d3.rgb("rgb(255,0,0)");
teamColor = d3.rgb(255,0,0);
```

- These color objects have two useful methods: .darker() and .brighter().
- They do exactly what you’d expect: return a color that’s darker or brighter than the color you started with.

Interactive style and DOM

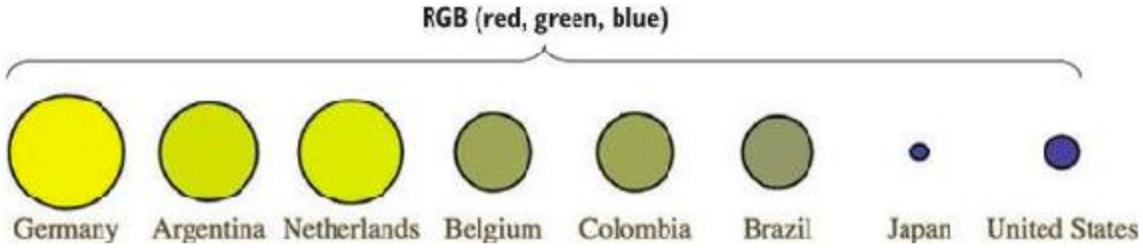
```
var ybRamp = d3.scaleLinear()
  .domain([0,maxValue]).range(["blue", "yellow"])          1

d3.selectAll("g.overallG").select("circle")
   .attr("r", d => radiusScale(d[datapoint]))
   .style("fill", d => ybRamp(d[datapoint]))
```

- 1 This is the same kind of color ramp we built earlier, using the maxValue we calculated for our circle radius scale

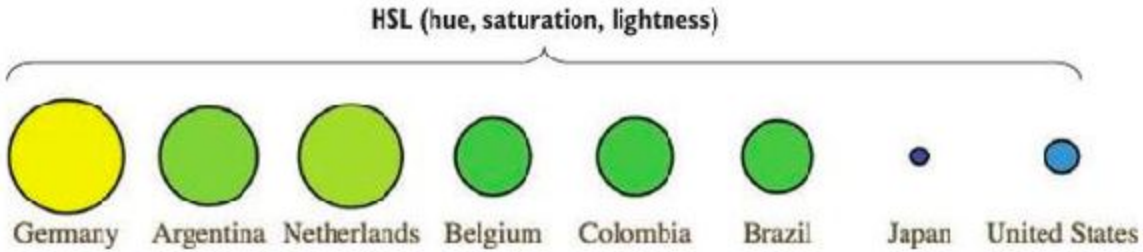
Interactive style and DOM

Color mixing between yellow and blue in the RGB (red-green-blue) scale results in muddy, grayish colors displayed for the values between yellow and blue.



Interactive style and DOM

Interpolation of yellow to blue based on hue, saturation, and lightness (HSL) results in a different set of intermediary colors from the same two starting values.



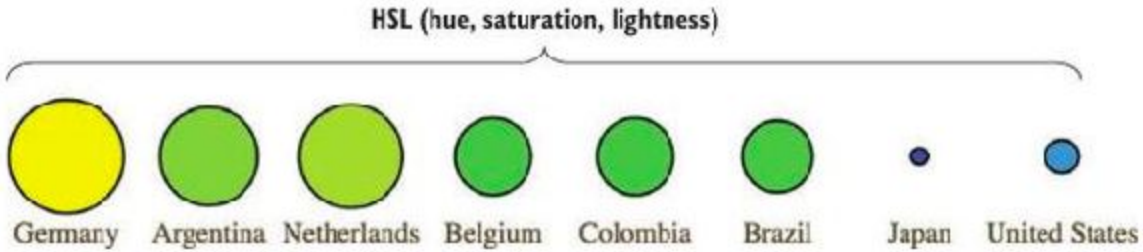
Interactive style and DOM

```
var ybRamp = d3.scaleLinear()
  .interpolate(d3.interpolateHsl)                    1
  .domain([0,maxValue]).range(["yellow", "blue"]);
```

- 1 Setting the interpolation method for a scale is necessary when we don’t want it to use its default behavior, such as when we want to create a color scale with a method other than interpolating the RGB values

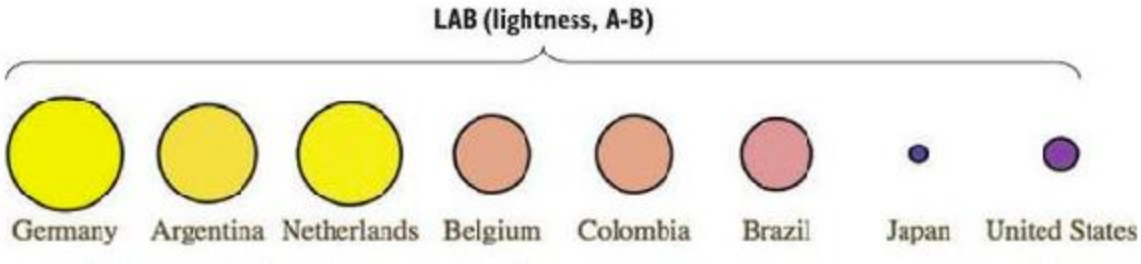
Interactive style and DOM

Interpolation of color based on hue, chroma, and luminosity (HCL) provides a different set of intermediary colors between yellow and blue.



Interactive style and DOM

Interpolation of color based on lightness and color-opponent space (known as LAB; L stands for lightness and A-B stands for the color-opponent space) provides yet another set of intermediary colors between yellow and blue.



Interactive style and DOM

```
var ybRamp = d3.scaleLinear()
  .interpolate(d3.interpolateHcl)
  .domain([0,maxValue]).range(["yellow", "blue"]);
```

```
var ybRamp = d3.scaleLinear()
  .interpolate(d3.interpolateLab)
  .domain([0,maxValue]).range(["yellow", "blue"]);
```

- As a general rule, you'll find that the colors interpolated in RGB tend toward muddy and gray, unless you break the color ramp into multiple stops.

Interactive style and DOM

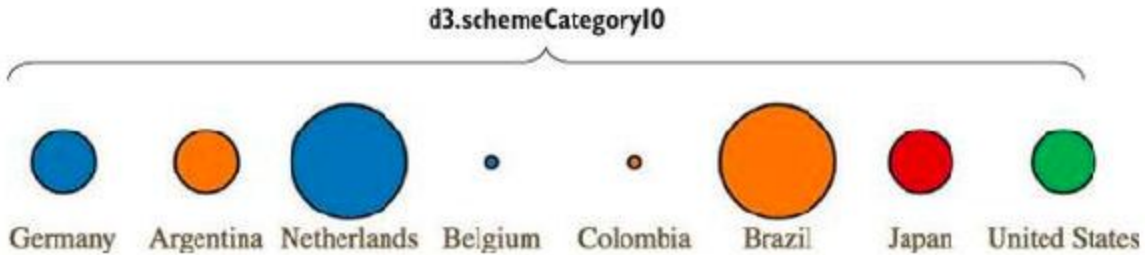
- Discrete colors
- D3 includes four collections of discrete color categories:
- d3.schemeCategory10
- d3.schemeCategory20
- d3.schemeCategory20b
- d3.schemeCategory20c.

Interactive style and DOM

```
function buttonClick(datapoint) {
   var maxValue = d3.max(incomingData, function(el) {
      return parseFloat(el[datapoint ])
   })
   var tenColorScale = d3.scaleOrdinal()
      .domain(["UEFA", "CONMEBOL", "CAF",  "AFC"])
      .range(d3.schemeCategory10)
   var radiusScale = d3.scaleLinear().domain([0,maxValue]).range([2,20])
   d3.selectAll("g.overallG").select("circle").transition().duration(1000)
      .style("fill", p => tenColorScale(p.region))
      .attr("r", p => radiusScale(p[datapoint ]))
}
```

Interactive style and DOM

Application of the schemeCategory10 to an ordinal scale in D3 assigns distinct colors to each class applied, in this case, the four regions in your dataset.



Interactive style and DOM

Utilizing the .unknown() method of an ordinal scale to serve back values for data that doesn’t have a corresponding entry in the scale’s domain


```
...
var tenColorScale = d3.scaleOrdinal()
  .domain(["UEFA", "CONMEBOL"])
  .range(d3.schemeCategory10)
  .unknown("#c4b9ac")
```

Interactive style and DOM

Color ramps for numerical data

```
Reds: {
3: ["#fee0d2","#fc9272","#de2d26"],
4: ["#fee5d9","#fcae91","#fb6a4a","#cb181d"],
5: ["#fee5d9","#fcae91","#fb6a4a","#de2d26","#a50f15"],
6: ["#fee5d9","#fcbba1","#fc9272","#fb6a4a","#de2d26","#a50f15"],
7:
["#fee5d9","#fcbba1","#fc9272","#fb6a4a","#ef3b2c","#cb181d","#99
000d"],
8: ["#fff5f0","#fee0d2","#fcbba1","#fc9272",
    "#fb6a4a","#ef3b2c","#cb181d","#99000d"],
9: ["#fff5f0","#fee0d2","#fcbba1","#fc9272","#fb6a4a",
    "#ef3b2c","#cb181d","#a50f15","#67000d"]
}
```

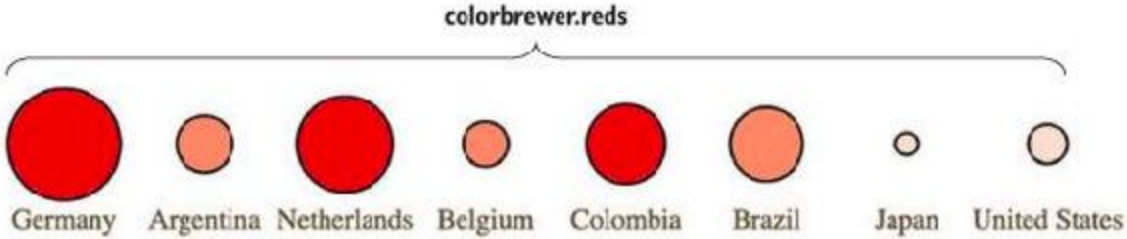
Interactive style and DOM

```
function buttonClick(datapoint) {                                    1
  var maxValue = d3.max(incomingData, d => parseFloat(d[datapoint]));
  var colorQuantize = d3.scaleQuantize()
    .domain([0,maxValue]).range(colorbrewer.Reds[3]);                2
  var radiusScale = d3.scaleLinear()
    .domain([0,maxValue]).range([2,20]);
  d3.selectAll("g.overallG").select("circle").transition().duration(1000)
    .style("fill", d => colorQuantize(d[datapoint]))
    .attr("r", d => radiusScale(d[datapoint]))
}
```

- 1 Our new buttonClick function sorts the circles in our visualization into three categories with colors associated with them
- 2 The quantize scale sorts the numerical data into as many categories as there are in the range. Because colorbrewer.Reds[3] is an array of three values, the dataset is sorted into three discrete categories, and each category has a different shade of red assigned

Interactive style and DOM

Automatic quantizing linked with the ColorBrewer 3-red scale produces distinct visual categories in the red family.



Pregenerated content

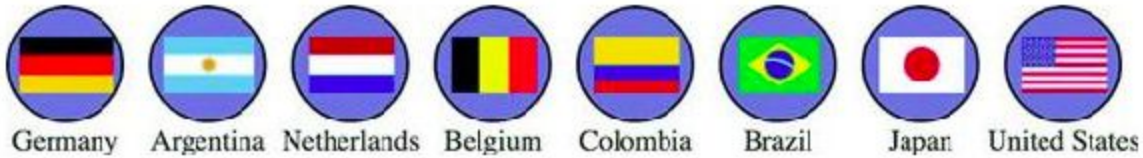
- Images
- In SVG, the image element is <image>, and its source is defined using the xlink:href

```
d3.selectAll("g.overallG").insert("image", "text")
  .attr("xlink:href", d => `images/${d.team}.png`)
  .attr("width", "45px").attr("height", "20px")
  .attr("x", -22).attr("y", -10)
```

- Use insert() instead of append() because that gives you the capacity to tell D3 to insert the images before the text
- Here the x and y attributes are set to a negative value of one-half the respective height and width, which centers the images

Pregenerated content

Our graphical representations of each team now include a small PNG national flag, downloaded from Wikipedia and loaded using an SVG <image> element.



Infoviz term: chartjunk, it’s suggested by the vis community not to use images/clip arts if it is not necessary.

Pregenerated content

- HTML fragments
- infobox.html

```
<table>
   <tr>
      <th>Statistics</th>
   </tr>
   <tr><td>Team Name</td><td class="data"></td></tr>
   <tr><td>Region</td><td class="data"></td></tr>
   <tr><td>Wins</td><td class="data"></td></tr>
   <tr><td>Losses</td><td class="data"></td></tr>
   <tr><td>Draws</td><td class="data"></td></tr>
   <tr><td>Points</td><td class="data"></td></tr>
   <tr><td>Goals For</td><td class="data"></td></tr>
   <tr><td>Goals Against</td><td class="data"></td></tr>
   <tr><td>Clean Sheets</td><td class="data"></td></tr>
   <tr><td>Yellow Cards</td><td class="data"></td></tr>
   <tr><td>Red Cards</td><td class="data"></td></tr>
</table>
```

Pregenerated content

Update to d3ia.css

```
#infobox {
  position: fixed;
  left: 150px;
  top: 20px;
  z-index: 1;
  background: white;
  border: 1px black solid;
  box-shadow: 10px 10px 5px #888888;
}
tr {
  border: 1px gray solid;
}
td {
  font-size: 10px;
}
td.data {
  font-weight: 900;
}
```

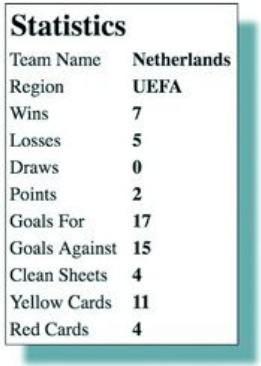
Pregenerated content

```
d3.text("resources/infobox.html", html => {
d3.select("body").append("div").attr("id", "infobox").html(html)
})                                                          1
teamG.on("click", teamClick)
function teamClick (d) {
d3.selectAll("td.data").data(d3.values(d))                          2
  .html(p => p)
}
```

- 1 Creates a new div with an id corresponding to one in our CSS, and populates it with HTML content from infobox.html
- 2 You could also simply use Object.values if you’re developing for browsers that support this functionality
- 3 Selects and updates the td.data elements with the values of the team clicked

Pregenerated content

The infobox is styled based on the defined style in CSS. It’s created by loading the HTML data from infobox.html and adding it to the content of a newly created div.

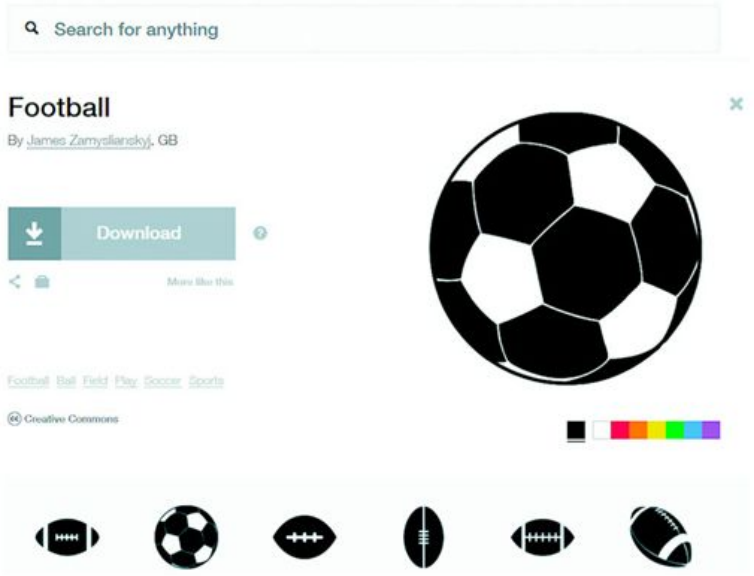


Statistics

Team Name	**Netherlands**
Region	**UEFA**
Wins	**7**
Losses	**5**
Draws	**0**
Points	**2**
Goals For	**17**
Goals Against	**15**
Clean Sheets	**4**
Yellow Cards	**11**
Red Cards	**4**

Pregenerated content

- Pregenerated SVG
- An icon for a soccer ball created by James Zamyslianskyj and available at http://thenounproject.com/term/football/1907/ from The Noun Project



Pregenerated content

- After loading an SVG file you can manipulate it like other SVG elements
- For the table we can just add it as a text
- For SVG there might be many redundant things
- For example, the SVG container, the groups are not that important
- We only care about geometric primitives like paths, circles, etc.

```
d3.html("resources/icon_1907.svg", data => {console.log(data)});
```

Pregenerated content

An SVG loaded using d3.html() that was created in Inkscape. It consists not only of the graphical <path> elements that make up the SVG but also much data that's often extraneous.

Pregenerated content

```
d3.html("resources/icon_1907.svg", loadSVG);                    1
function loadSVG(svgData) {
  while(!d3.select(svgData).selectAll("path").empty()) {
    d3.select("svg").node().appendChild(
        d3.select(svgData).select("path").node());
   }
    d3.selectAll("path").attr("transform", "translate(50,50)");
}
```

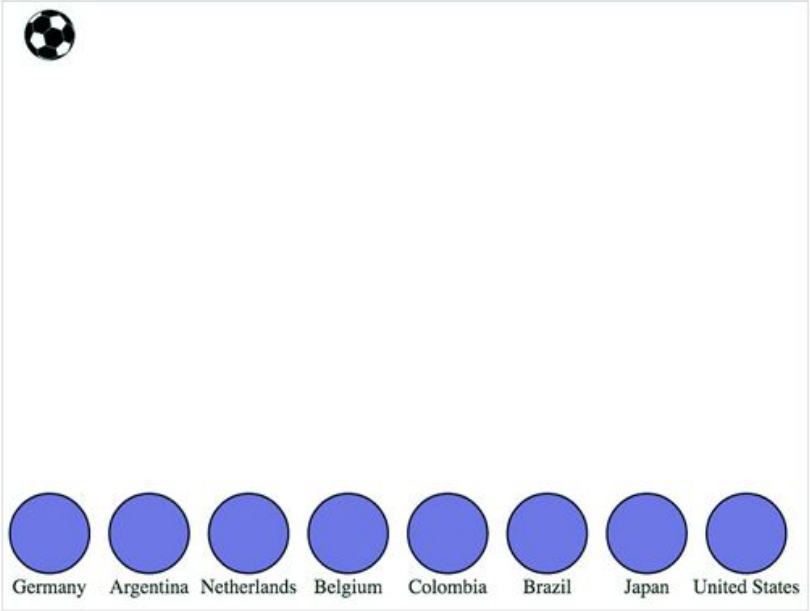
- 1 The data variable will automatically be passed to loadSVG().

Pregenerated content

```
function loadSVG(svgData) {
  d3.select(svgData).selectAll("path").each(function() {
    d3.select("svg").node().appendChild(this);
  });
  d3.selectAll("path").attr("transform", "translate(50,50)");
}
```

Pregenerated content

A hand-drawn soccer ball icon is loaded onto the <svg> canvas, along with the other SVG and HTML elements we created in our code.



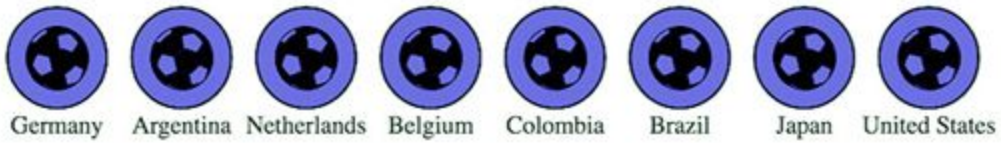
Pregenerated content

```
d3.html("resources/icon_1907.svg", loadSVG);
function loadSVG(svgData) {
  d3.selectAll("g").each(function() {                    1
  var gParent = this;
  d3.select(svgData).selectAll("path").each(function() {      1
      gParent.appendChild(this.cloneNode(true))
  });
});
};
```

- 1 Note that we can’t use arrow functions here because we need to have access to this context within the selection that corresponds to the DOM node

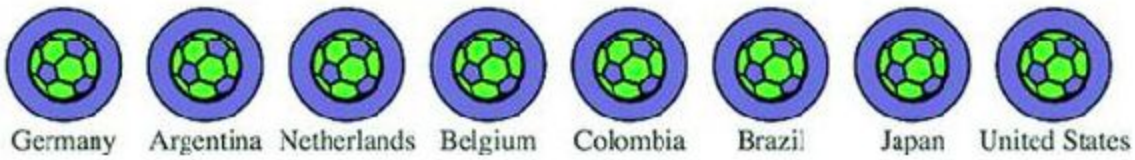
Pregenerated content

Each <g> element has its own set of paths cloned as child nodes, resulting in soccer ball icons overlaid on each element.



Pregenerated content

Football icons with a fill and stroke set by D3




```
d3.selectAll("path").style("fill", "#93C464")
  .style("stroke", "black").style("stroke-width", "1px");
```

Pregenerated content

- Some drawbacks:
  - Since we haven’t use insert method, the elements will not be placed behind the labels
  - Because of using cloneNode, the element have no data bound to them
  - So we need to explicitly bind the data now

Pregenerated content

```
d3.selectAll("g.overallG").each(function(d) {
    d3.select(this).selectAll("path").datum(d)
});
var fourColorScale = d3.scaleOrdinal()
    .domain(["UEFA", "CONMEBOL", "CAF", "AFC"])
    .range(["#5eafc6", "#FE9922", "#93C464", "#fcbc34" ])
d3.selectAll("path").style("fill", p => fourColorScale(p.region))
.style("stroke", "black").style("stroke-width", "2px");
```

Pregenerated content

The paths now have the data from their parent element bound to them and respond accordingly when a discrete color scale based on region is applied.