CSE 590
Data Science Fundamentals
Using R with D3

Klaus Mueller

Computer Science Department
Stony Brook University and SUNY Korea
<table>
<thead>
<tr>
<th>Lecture</th>
<th>Topic</th>
<th>Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Intro, schedule, and logistics</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Data Science components and tasks</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Data types</td>
<td>Project #1 out</td>
</tr>
<tr>
<td>4</td>
<td>Introduction to R, statistics foundations</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td><strong>Introduction to D3, visual analytics</strong></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Data preparation and reduction</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Data preparation and reduction</td>
<td>Project #1 due</td>
</tr>
<tr>
<td>8</td>
<td>Similarity and distances</td>
<td>Project #2 out</td>
</tr>
<tr>
<td>9</td>
<td>Similarity and distances</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Cluster analysis</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Cluster analysis</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Pattern miming</td>
<td>Project #2 due</td>
</tr>
<tr>
<td>13</td>
<td>Pattern mining</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Outlier analysis</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Outlier analysis</td>
<td>Final Project proposal due</td>
</tr>
<tr>
<td>16</td>
<td>Classifiers</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Midterm</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Classifiers</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Optimization and model fitting</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Optimization and model fitting</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Causal modeling</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Streaming data</td>
<td>Final Project preliminary report due</td>
</tr>
<tr>
<td>23</td>
<td>Text data</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Time series data</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Graph data</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Scalability and data engineering</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Data journalism</td>
<td>Final Project slides and final report due</td>
</tr>
</tbody>
</table>
D3 = Data Driven Documents

JavaScript library for manipulating documents based on data
  - frequent tool to support data journalism (New York Times)

D3 helps you bring data to life using HTML, SVG, and CSS
  - great library to construct animated visualizations (D3 website)

Runs in any modern web browser (Chrome, Firefox, IE)
  - no need to download any software
  - independent of OS (Linux, Windows Mac)
Makes Use Of

- HTML  Hypertext Markup Language
- CSS   Cascading Style Sheets
- JS    JavaScript
- DOM   The Document Object Model
  - tree structured organization of HTML objects
- SVG   Scalable Vector Graphics
WHAT YOU NEED

A text editor
- textMate, eclipse/aptana, sublime text 2...
- need an editor with syntax highlighting. else it’s easy to get lost

The d3 library
- from http://d3js.org

Data files for your code

A web server (recommended)
- if your visualization is reading data from files or a database (XMLHttpRequest)
- many options: EasyPHP (windows), Mac OS X Server, MAMP
- else need to specify the data in the code

A browser
- to run the code
Suppose you defined three circles

```
<svg width="720" height="120">
  <circle cx="40" cy="60" r="10"></circle>
  <circle cx="80" cy="60" r="10"></circle>
  <circle cx="120" cy="60" r="10"></circle>
</svg>
```

This will select all circles

```
var circle = d3.selectAll("circle");
```

And enlarge and fill them

```
circle.style("fill", "steelblue");
circle.attr("r", 30);
```

```
<svg width="720" height="120">
  <circle cx="40" cy="60" r="30" style="fill:steelblue;"></circle>
  <circle cx="80" cy="60" r="30" style="fill:steelblue;"></circle>
  <circle cx="120" cy="60" r="30" style="fill:steelblue;"></circle>
</svg>
```
The selection.data method binds the numbers to the circles:

```javascript
circle.data([32, 57, 112]);
```

Assign attributes to the bound data
- typically use the name `d` to refer to bound data

```javascript
circle.attr("r", function(d) { return Math.sqrt(d); });
```

Will result in:

```html
<svg width="720" height="120">
  <circle cx="40" cy="60" r="5.656854249492381" style="fill:steelblue;"/>
  <circle cx="80" cy="60" r="7.54983443527075" style="fill:steelblue;"/>
  <circle cx="120" cy="60" r="10.583005244258363" style="fill:steelblue;"/>
</svg>
```
We can use the index $i$ of the data to define the graphics. Origin is the upper left corner.

circle.attr("cx", function(d, i) { return i * 100 + 30; });

<svg width="720" height="120">
  <circle cx="30" cy="60" r="5.656854249492381" style="fill:steelblue;"/>
  <circle cx="130" cy="60" r="7.54983443527075" style="fill:steelblue;"/>
  <circle cx="230" cy="60" r="10.583005244258363" style="fill:steelblue;"/>
</svg>
Suppose you have more data than graphics elements

- use the enter method to add them on the fly

```javascript
var svg = d3.select("svg");
var circle = svg.selectAll("circle")
    .data([32, 57, 112, 293]);
var circleEnter = circle.enter().append("circle");

circleEnter.attr("cy", 60);
circleEnter.attr("cx", function(d, i) { return i * 100 + 30; });
circleEnter.attr("r", function(d) { return Math.sqrt(d); });
```

- as usual, but now with 4 circles
(continued) we get

```xml
<svg width="720" height="120">
  <circle cx="30" cy="60" r="5.656854249492381" style="fill:steelblue;"/>
  <circle cx="130" cy="60" r="7.54983443527075" style="fill:steelblue;"/>
  <circle cx="230" cy="60" r="10.583005244258363" style="fill:steelblue;"/>
  <circle cx="330" cy="60" r="17.11724276862369" style="fill:steelblue;"/>
</svg>
```

We can even begin with no circles at all:

```javascript
svg.selectAll("circle")
  .data([32, 57, 112, 293])
  .enter().append("circle")
  .attr("cy", 60)
  .attr("cx", function(d, i) { return i * 100 + 30; })
  .attr("r", function(d) { return Math.sqrt(d); });
```
Connecting R with D3
Simple option: use ggplot2

- tries to adhere to the grammar of graphics
- ggplot2

The grammar of graphics

- governs the composition of graphical components in statistical graphics
- by directly controlling that grammar, you can generate a large set of carefully constructed graphics tailored to your particular needs
- controls positions, shapes, appearance, etc. of the primitives
- each component is added to the plot as a layer

Produces a static plot
LAYERING WITH GGPPLOT2
UL: `ggplot(mpg, aes(displ, hwy))+geom_point()
  ▪ mpg is the dataset, aes is aesthetic mapping
  ▪ geom_point adds the points

UR: `ggplot(mpg, aes(displ, hwy))+ geom_point()+geom_line()
  ▪ geom_line adds the lines

LL: `ggplot(mpg, aes(displ, hwy))+geom_point(aes(color = factor(cyl)))+geom_line()
  ▪ colored the points by cylinder and auto-added legend

LR: `ggplot(mpg, aes(displ, hwy, color = factor(cyl)))+geom_point()+geom_line()
  ▪ colored everything by cylinder and auto-added legend

For more info see [here](#)


**GGPLOT PROS AND CONS**

**Pros**

- easy to code
- get an image fast

**Cons**

- it’s a static png image
- can’t zoom to see more detail
- need to run the R script again and render a new image
Combining R and D3

Option #1:
- R does the data processing **and** the graph rendering
- then exports this as an SVG
- bind your JavaScript later on
- for 1-way communication from R to a webpage with JavaScript

Option #2:
- R does the data processing
- then sends this data to JavaScript to create an SVG image
- also enables 2-way communication → rerun R scripts based on input from a web application
Option 1: First plotting, then binding

gridSVG
- requires a lot of manual coding
- example

plotly
- commercial
- also works for Python, Matlab, NodeJS and Excel
- free part has basic, but somewhat limited functionality
- cannot use the full spectrum of the D3 library
- only suitable for ‘basic’ charts and plots.
- example
R does the processing and delivers the data
  ▪ this is then used as input for the JavaScript visualization

Example
  ▪ use R to get and clean the data
  ▪ use JS to visualize the data
**Example: Blog Site Data**

<table>
<thead>
<tr>
<th>Title</th>
<th>Author</th>
<th>Categories</th>
<th>SEO Title</th>
<th>Meta Desc.</th>
<th>Focus KW</th>
</tr>
</thead>
<tbody>
<tr>
<td>But do you love it?</td>
<td>Roman Verraest</td>
<td>Business Consulting, Enterprise Architecture</td>
<td>But do you love it? - AE Blog</td>
<td>Customers don’t really enjoy a perfect functional system. They expect it. It is the journey with your company that they can start to love.</td>
<td>Love</td>
</tr>
<tr>
<td>Why not add some 'WHY' to your decision documents?</td>
<td>Wim Van Emelen</td>
<td>Business Consulting, Enterprise Architecture, Software Engineering</td>
<td>Why not add some 'WHY' to your decision documents? - AE Blog</td>
<td>In this article, I'd like to share 3 practical tips to increase the value of any requirements document, enabling faster decision taking for stakeholders.</td>
<td>decision</td>
</tr>
<tr>
<td>5 Questions about the challenges in the energy sector</td>
<td>Brigitte Narmon</td>
<td>Business Consulting</td>
<td>5 Questions about the challenges in the energy sector</td>
<td>A Q&amp;A with Brigitte Harman, lead of the AE Utilities Community, about the challenges of the Belgian energy sector and how AE prepares for this (revolution?)</td>
<td>energy</td>
</tr>
<tr>
<td>The AE Blog: Looking back and looking forward</td>
<td>AE admin</td>
<td>Business Consulting</td>
<td>The AE Blog: Looking back and looking forward - AE Blog</td>
<td>In 2015 we want to build upon last year’s success of the AE...</td>
<td>blog</td>
</tr>
</tbody>
</table>
Steps:

- the html page contains an overview of all blog posts
- blog-data in this webpage is structured in an html-table
- save this page as a static html page so it can be parse more easily
- R has packages to easily scrape the data from such a table

R code:

```r
library(XML)

# read all html table elements
raw <- readHTMLTable("WordPress.html")

# ours is the first of two tables
# in the html document
data <- raw[[1]]
```
Data Structure:

- need to store the data in the right format – JSON
- JSON = JavaScript Object Notation, see [here](#)
- contains all relations (name/value pairs), for each single element
- aka object, record, struct, dictionary, keyed list, assoc. array, etc.
- will be the data-input for the visualization

```json
[{
  name: "Title.But do you love it?",
  size: 0,
  imports: [
    "Author.Roman Verraest",
    "Categorie.Business Consulting",
    "Tag.AE Foyer",
    "Date.2014-12",
    "Tag.Business Architecture",
    "Tag.Customer Journey",
    "Categorie.Enterprise Architecture"
  ]
},
{
  ...}
]
Pipe into D3

- plug the **data** into this **code** will produce this
- move the mouse to see the relations among the different entities.
Two-Way Communication

Rerun R scripts based on input from a web application
  - this is a bit more involved using D3 – not covered in this lecture
  - may not be needed for this course
  - Shiny webserver can do this

Shiny
  - R package that makes it easy to build interactive web apps
  - currently free and available [here](http://example.com)
  - this site has also a comprehensive tutorial
  - uses D3 but hides it from the user completely
  - this can be limiting

Structure of a Shiny App – two components
  - a user-interface script
  - a server script
ui.R

library(shiny)

# Define UI for application that draws a histogram
shinyUI(fluidPage(

titlePanel("Hello Shiny"),

# Sidebar with a slider input for the number of bins
sidebarLayout(
  sidebarPanel(
    sliderInput("bins",
      "Number of bins:",
      min = 1,
      max = 50,
      value = 30)
  ),

  # Show a plot of the generated distribution
  mainPanel(
    plotOutput("distPlot")
  )
)
))
library(shiny)

# Define server logic required to draw a histogram
shinyServer(function(input, output) {

  # Expression that generates a histogram. The expression is
  # wrapped in a call to renderPlot to indicate that:
  #
  # 1) It is "reactive" and therefore should re-execute automatically
  #    when inputs change
  # 2) Its output type is a plot

  output$distPlot <- renderPlot({
    x <- faithful[, 2]  # Old Faithful Geyser data
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })
})
**Running Shiny**

Note:
- your R session will be busy while the Hello Shiny app is active
- you will not be able to run any R commands
- R is monitoring the app and executing the app’s reactions
- to get your R session back, hit escape or click the stop sign in the RStudio console panel

Run app by:

```
> library(shiny)
> runApp("my_app")
```
Hello World!

Number of bins:

Histogram of x

Frequency

x
Most of the material presented here was from [here](#) but there is much more information on the web

What should you choose?

- ggplot2 (easy)
- shiny (a bit more difficult, but still easy)
- binding in D3 (takes a learning curve, but rewarding)
- up to you – get as much as you want out of this course

- (I would choose D3)
The page where the D3 tutorial bits came from:
http://www.lessonpaths.com/learn/i/begin-with-d3js/d3js-simplest-examples-of-d3js

Another good tutorial
http://alignedleft.com/tutorials/d3

Now to a more detailed, but still primitive example:
http://www.lessonpaths.com/learn/i/begin-with-d3js/d3js-simplest-examples-of-d3js

Here are some full-fledged implementations: