Applications & Libraries

VU Visual Data Science
Johanna Schmidt

WS 2019/20
Visual Data Science Tools

• Several different tools available
Visual Data Science Tools

• Differentiate between
  • Charting libraries
  • Applications
Visual Data Science Tools

• Differentiate between
  • **Charting libraries**
    • Python, R, Matlab, ...
    • Embedded in programming language environment
    • Require programming skills
    • Often Open Source
  • Applications
Visual Data Science Tools

• Differentiate between
  • **Charting libraries**
    • Python Plotly
    • Python Matplotlib
    • D3
    • Highcharts
    • GGPlot
  • Applications
Visual Data Science Tools

• Differentiate between
  • Charting libraries
  • Applications
    • Standalone applications
    • Provide means for data handling & visualization
    • Usually no programming skills required
    • In many cases commercial
Visual Data Science Tools

• Differentiate between
  • Charting libraries
  • **Applications**
    • Excel
    • Tableau
    • Microsoft Power BI
    • Cognos
    • QlikView
    • ...

Visual Data Science Tools

• Categorization / Evaluation
Visual Data Science Tools

- **Categorization / Evaluation**
  - Categorization based on comparison

https://source.opennews.org/articles/what-i-learned-recreating-one-chart-using-24-tools/
Visual Data Science Tools

- **Categorization / Evaluation**
  - Categorization based on comparison
Visual Data Science Tools

• **Categorization / Evaluation**
  • Evaluation of existing applications
Visual Data Science Tools

• Differentiate between
  • Charting libraries
  • Applications
Tool Comparison

• Study by **Lisa Charlotte Rost** (Datawrapper): *What I Learned Recreating One Chart Using 24 Tools*

• Study in **2016**

• Compared **12 charting libraries** and **12 applications**

Tool Comparison
Tool Comparison

[Excel]

[Google Sheets]
Tool Comparison

Tableau

Plotly
Tool Comparison

• Task
Tool Comparison

- Flexibility
Tool Comparison

• Learning curve
Tool Comparison

- Environment
Tool Comparison

• **Conclusion:**
  • “There Are No Perfect Tools, Just Good Tools for People with Certain Goals”
Tool Comparison

- Conclusion
Charting Libraries

• Differentiate between
  • Charting libraries
  • Applications (next lecture)
Tool Comparison

• Task
## Tool Comparison

- **Task**

<table>
<thead>
<tr>
<th>Tool</th>
<th>SEABORN</th>
<th>R</th>
<th>GG PLOT2</th>
<th>GGVIS</th>
<th>MAT PLOT LIB</th>
<th>BOKEH</th>
<th>VEGA-LITE</th>
<th>PROCESSING</th>
<th>HIGH CHARTS</th>
<th>VEGA</th>
<th>D3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C3</strong></td>
<td>C3</td>
<td>NVD3</td>
<td>D4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[2]
Charting Libraries

- **Libraries**
  - Seaborn
  - R
  - ggplot2
  - ggvis
  - matplotlib
  - bokeh
  - Vega-Lite
  - Vega
  - Processing
  - Highcharts
  - D3
  - D4
  - C3
  - NVD3
Charting Libraries

- Libraries
  - Python
  - R
  - ggplot
  - Vega
  - Processing
  - JavaScript
Charting Libraries

• Libraries
  • Python (matplotlib, bokeh, Seaborn)
  • R (R, ggvis)
  • Vega (Vega, Vega-Lite)
  • Chartbuilder
  • Processing
  • JavaScript (D3, D4, C3, NVD3)
Charting Libraries

• **Libraries**
  • **Python** (matplotlib, bokeh, Seaborn)
  • R (R, ggvis)
  • Vega (Vega, Vega-Lite)
  • Chartbuilder
  • Processing
  • JavaScript (D3, D4, C3, NVD3)
Python

• **Scripting** language
• **Open-source** development
• **Well-known** programming language for data science
• Many **plotting libraries** available (plotly, matplotlib, Bokeh, Seaborn)
• Flexible, allow **explorative data analysis**
Python

Pandas

Seaborn

bokeh
```python
import numpy as np
import pandas as pd
mortality_age = pd.read_csv("mortality_by_age.csv")
mortality_age = mortality_age.sort_values('Deaths per 100,000 Live Births', ascending=False)[5:]
mortality_age.plot(kind="bar")
```

```python
from bokeh.charts import Bar, output_file, show
mortality_age = pd.read_csv("mortality_by_age.csv")
data = {
    'Age Range': mortality_age['Age Range'],
    'Deaths per 100,000 Live Births': mortality_age['Deaths per 100,000 Live Births']
}

# x axis labels pulled from the interpreter column, stacking labels from sample column
bar = Bar(data, values='Deaths per 100,000 Live Births',
label='Deaths per 100,000 Live Births',
stack='Age Range', agg='mean',
title='Maternal Deaths per 100,000 Live Births',
legend='top_right', width=400)

output_file("barchart.html")
show(bar)
```
Python

- **Pandas**: for simple plots
- **Seaborn**: more complex visualization, requires matplotlib knowledge
- **ggplot**: lot of promise
- **bokeh**: robust tool, overkill for simple scenarios
- **pygal**: interactive svg graphs, but not as flexible as others
- **Plotly**: for highly interactive graphs and rich web-based visualizations
Python

• Task

ANALYSIS ← SEABORN R GGPILOT2 GGVIS MATPLOTLIB ← BOKEH → PRESENTATION

VEGA-LITE PROCESSING HIGHCHARTS VEGA D3 C3 NVD3 D4
Python

- Flexibility
Python

- Learning curve
Charting Libraries

• Libraries
  • Python (matplotlib, bokeh, Seaborn)
  • R (R, ggvis)
  • Vega (Vega, Vega-Lite)
  • Chartbuilder
  • Processing
  • JavaScript (D3, D4, C3, NVD3)
R

- **Scripting** language
- **Open-source** development
- **Well-known** programming language for data science Flexible, allow explorative data analysis
• **Graphics** package for basic charts
• **ggplot2** for more advanced visualizations

[https://rkabacoff.github.io/datavis/Models.html](https://rkabacoff.github.io/datavis/Models.html)
Charting Libraries

• **Libraries**
  • Python (matplotlib, bokeh, Seaborn)
  • R (R, ggvis)
  • **Vega** (Vega, Vega-Lite)
  • Chartbuilder
  • Processing
  • JavaScript (D3, D4, C3, NVD3)
Vega

• Defines a **visualization grammar**
• Library for **creating**, **saving**, and **sharing** interactive visualizations
• Defines visualizations as **JSON format**
• **Open-source**
Vega

- **JSON Format**
  - Display format (e.g., size)
  - Data
  - Scales (axes scales & visual mapping)
  - Axes (orientation, ticks, labels)
  - Marks (visual elements)
  - Signals (interaction)
Vega

• **Creating Vega JSONs**
  • Vega editors
  • Export functionality of current tools/applications
Vega

- Interpretation & display
  - Vega viewers
  - Embedded in web applications
Vega-Lite: A Grammar of Interactive Graphics
Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer

Fig. 1. Example visualizations authored with Vega-Lite. From left-to-right: layered line chart combining raw and average values, dual-axis layered bar and line chart, brushing and linking in a scatterplot matrix, layered cross-filtering, and an interactive index chart.

Abstract—We present Vega-Lite, a high-level grammar that enables rapid specification of interactive data visualizations. Vega-Lite combines a traditional grammar of graphics, providing visual encoding rules and a composition algebra for layered and multi-view displays, with a novel grammar of interaction. Users specify interactive semantics by composing selections. In Vega-Lite, a selection is an abstraction that defines input event processing, points of interest, and a predicate function for inclusion testing. Selections parameterize visual encodings by serving as input data, defining scale extents, or by driving conditional logic. The Vega-Lite compiler automatically synthesizes requisite data flow and event handling logic, which users can override for further customization. In contrast to existing reactive specifications, Vega-Lite selections decompose an interaction design into concise, enumerable semantic units. We evaluate Vega-Lite through a range of examples, demonstrating succinct specification of both customized interaction methods and common techniques such as panning, zooming, and linked selection.

Index Terms—Information visualization, interaction, systems, toolkits, declarative specification
Vega-Lite

• **Version 2** released in 2017, current version: 3
• Extends Vega grammar to add **view composition** and **interaction (selection)**
Vega-Lite

- **Version 2** released in 2017, current version: 3
- Extends Vega grammar to add *view composition* and *interaction* (selection)
  - **View Composition**
    - Subdivide data into groups and creates chart for every group (*facet*)
    - Combine charts into one layout (*concat*, *repeat*)
Vega-Lite

• Demo

https://vega.github.io/editor/#/examples
Vega-Lite

- **Version 2.0** released in 2017
- Extends Vega grammar to add *view composition* and *interaction (selection)*
  - **View composition**
    - Subdivide data into groups and creates chart for every group (*facet*)
    - Combine charts into one layout (*concat, repeat*)
  - **Interaction (selection)**
    - Different types of selections
    - In combination with view composition, enables linking & brushing
Vega-Lite

• **Advantages**
  • All you need to create a visualization
  • Grammar-based specification
  • Flexibility
  • Transferability

• **Disadvantages**
  • Limited possibilities to create charts
  • Understanding the JSON structure
Vega / Vega-Lite

• Task
Vega / Vega-Lite

- Flexibility
Vega / Vega-Lite

• Learning curve
Charting Libraries

- **Libraries**
  - Python (matplotlib, bokeh, Seaborn)
  - R (R, ggvis)
  - Vega (Vega, Vega-Lite)
  - **Chartbuilder**
  - Processing
  - JavaScript (D3, D4, C3, NVD3)
Chartbuilder

https://quartz.github.io/Chartbuilder/
Charting Libraries

• **Libraries**
  • Python (matplotlib, bokeh, Seaborn)
  • R (R, ggvis)
  • Vega (Vega, Vega-Lite)
  • Chartbuilder
  • **Processing**
  • JavaScript (D3, D4, C3, NVD3)
Processing

• **Visualization language** built on Java

```java
void setup() {
    size(640, 360);
    background(102);
}

void draw() {
    // Call the variableEllipse() method and send it the
    // parameters for the current mouse position
    // and the previous mouse position
    variableEllipse(mouseX, mouseY, pmouseX, pmouseY);
}

// The simple method variableEllipse() was created specifically
// for this program. It calculates the speed of the mouse
// and draws a small ellipse if the mouse is moving slowly
// and draws a large ellipse if the mouse is moving quickly

void variableEllipse(int x, int y, int px, int py) {
    float speed = abs(x-px) + abs(y-py);
    stroke(speed);
    ellipse(x, y, speed, speed);
}
```
Processing

• **Visualization language** built on Java
• JavaScript interpretator available for **web-based usage**

[https://processing.org/examples/penrosetile.html](https://processing.org/examples/penrosetile.html)
Charting Libraries

• **Libraries**
  • Python (matplotlib, bokeh, Seaborn)
  • R (R, ggvis)
  • Vega (Vega, Vega-Lite)
  • Chartbuilder
  • Processing
  • **JavaScript** (D3, D4, C3, NVD3)
VU Visualisierung 1 (186.827)

d3 Tutorial

https://www.cg.tuwien.ac.at/courses/Visualisierung1/VU.html

Manuela Waldner

Institute of Computer Graphics and Algorithms, TU Wien, Austria
- JavaScript library
- Generate and modify Document Object Model (DOM) based on data
- Uses web standards HTML, SVG, CSS
- Versions:
  - Latest version: 5
  - Most wide-spread version 3 is quite different
DOM, CSS & HTML
JavaScript
  JavaScript for Java developers
  JavaScript ES 6
  DOM access
SVG

D3
  Selections
  Data Binding
  Events
  Scaling
  Array Manipulation
  Data Loading
  Same-origin / cross-origin
  Debugging
Document Object Model (DOM)

- When loading a web page, browser creates DOM of a page
- HTML DOM: W3C standard to get, change, add, delete HTML elements
- Tree of objects:

```
Document
   Root element: <html>
     Element: <head>
       Element: <title>
         Text: "My title"
     Element: <body>
       Element: <a>
         Attribute: "href"
         Text: "My link"
       Element: <h1>
         Text: "My header"
```

https://www.w3schools.com/js/js_htmldom.asp
- Data array
  - Numbers
  - Strings
  - Objects
  - ...
- DOM elements
- **HTML5 `<svg>` element**
  - Container for Scalable Vector Graphics
  - Language for describing 2D graphics in XML
  - Modifies the DOM

- **Example:**

  ```xml
  <svg width="100" height="100">
    <circle cx="50" cy="50" r="40" stroke="green" stroke-width="4" fill="yellow" />
  </svg>
  ```
Empty SVG container (HTML body):

```html
<svg width="200" height="100" id="svg"></svg>
```

d3:

```javascript
d3.select("#svg").append("circle")
  .attr("cx", 50)
  .attr("cy", 50)
  .attr("r", 40)
  .attr("stroke", "green")
  .attr("stroke-width", 4)
  .attr("fill", "yellow");
```
Generate a bar chart from this array of numbers

```javascript
var data = [4, 8, 15, 16, 23, 42];
```
<!DOCTYPE html>
<html lang="en">
<head>
  <style>
    .chart div {
      font: 10px sans-serif;
      background-color: steelblue;
      text-align: right;
      padding: 3px;
      margin: 1px;
      color: white;
    }
  </style>
</head>
<meta charset="UTF-8">
<title>DIV Bar Chart</title>
<script src="https://d3js.org/d3.v5.min.js"></script>
</head>
<body>
<div class="chart"></div>
<script>
  var data = [4, 8, 15, 16, 23, 42];

  d3.select(".chart")
  .selectAll("div")
   .data(data)
   .enter().append("div")
   .style("width", function(d) { return d * 10 + "px"; })
   .text(function(d) { return d; });
</script>
</body>
D3 Events

- User interaction:
  - **on**: adds event listener to each element in current selection
  - Types: click, mouseover, mouseout... (all DOM events types)
  - **d3.event** contains current event properties (e.g., x and y mouse coordinates of **d3.mouse**)

```javascript
d3.selectAll("div")
  .on("mouseover", function(){
    d3.select(this)
    .style("background-color", "orange");

    // Get current event info
    console.log(d3.event);

    // Get x & y co-ordinates
    console.log(d3.mouse(this));
  })
  .on("mouseout", function(){
    d3.select(this)
    .style("background-color", "steelblue")
  });
```

http://www.tutorialsteacher.com/d3js/event-handling-in-d3js
Processing vs. D3

- **Processing** easier to learn and better for making quick prototypes
- **D3** not suited for making quick prototypes
- **D3** has a steep learning curve, but a large community for getting help/ideas
- More tools available for **D3**
- Both libraries allow to publish results online
JavaScript

• Task
JavaScript

• Flexibility
JavaScript

• Learning curve
Web-based Visualisation

• **Client-server** environment
• Necessary to transfer **data** to client
• Data processing/visualisation done on the **client**
Conclusion

• Charting libraries require **programming skills**
• No fully-featured **applications** (e.g., data loading)
• Limited support for **interaction** (e.g., multiple views)
• More **flexibility**
• Better **integration** of analysis and visualization
• **Open source**
Next lecture

• Comparison / evaluation of applications for visual data science
• Please note: different lecture room
References

[1] https://practicalanalytics.files.wordpress.com/2012/01/implementingbusinessanalytics.png
[10] https://processing.org/examples/pattern.html