Toolkits for Visualization and UIs in Data Science

Guest Lecture in Advanced UI Software

Dominik Moritz, April 8, 2020
Hi!
my name is

Dominik Moritz
@domoritz

I build visualization tools
Finished my PhD from UW in 2019
Currently at Apple working on Human Centered ML and Visualization
Starting at CMU HCII in the Fall
After today, I want you to be able to:

Articulate the difference between chart typologies and component architectures.

Decompose a graphic into the building blocks of visualization.

Know why most visualization toolkits are declarative.

Explain tradeoff between expressiveness and ease-of-use.

Know the history of visualization tools.

Understand the **basics** of D3.

Give examples of manifestations of design principles in Vega-Lite.
Material for this lecture is a combination of:

- Jeffrey Heer’s Tools lecture in CSE 512
- Vega-Lite talk at OpenVis Conf
- My job talk
- Some new slides
Toolkits for Visualization and UIs in Data Science

Guest Lecture in Advanced UI Software

Dominik Moritz, April 8, 2020
The ability to take data—to be able to understand it, to process it, to extract value from it, to visualize it, to communicate it—that's going to be a hugely important skill in the next decades, ...

because now we really do have essentially free and ubiquitous data. So the complimentary scarce factor is the ability to understand that data and extract value from it.

Hal Varian, Google's Chief Economist
The McKinsey Quarterly, Jan 2009
Four major influences act on data analysis today:

1. The formal theories of statistics.
2. Accelerating developments in computers and display devices.
3. The challenge, in many fields, of more and ever larger bodies of data.
4. The emphasis on quantification in an ever wider variety of disciplines.

---

Data Analysis & Statistics. Turkey and Wilk. 1965.
Effective Data Visualization. Heer. 2015.
While some of the influences of statistical theory on data analysis have been helpful, others have not.
Exposure, the effective laying open of the data to display the unanticipated, is to us a major portion of data analysis. Formal statistics has given almost no guidance to exposure; indeed, it is not clear how the informality and flexibility appropriate to the exploratory character of exposure can be fitted into any of the structures of formal statistics so far proposed.
Nothing - not the careful logic of mathematics, not statistical models and theories, not the awesome arithmetic power of modern computers - nothing can substitute here for the flexibility of the informed human mind.

Accordingly, both approaches and techniques need to be structured so as to facilitate human involvement and intervention.
How do people create visualizations?

<table>
<thead>
<tr>
<th>Chart Typology</th>
<th>Component Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pick from a stock of templates</td>
<td>Permits more combinational possibilities</td>
</tr>
<tr>
<td>Easy-to-use but limited expressiveness</td>
<td>Novel views require new operators, which requires software engineering</td>
</tr>
<tr>
<td>Prohibits novel designs, new data types</td>
<td></td>
</tr>
</tbody>
</table>
Graphics APIs

Processing, OpenGL, Java2D
Drawing Visualizations with Imperative Programs

Program by giving explicit steps.

e.g.

"Put a red bar here and a blue bar there."
"Draw a line and some text."

Specification and execution are intertwined.

"You have unlimited power on this canvas. You can literally move mountains." — Bob Ross
```java
ey = y;
size = s;

void update(int mx, int my) {
    angle = atan2(my-ey, mx-ex);
}

void display() {
    pushMatrix();
    translate(ex, ey);
    fill(255);
    ellipse(0, 0, size, size);
    rotate(angle);
    fill(153);
    ellipse(size/4, 0, size/2, size/2);
    popMatrix();
}
```
Graphics APIs
Processing, OpenGL, Java2D
Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D
Data State Model

[Chi 98]
Prefuse & Flare
Operator-based toolkits for visualization design
Vis = (Input Data -> Visual Objects) + Operators

Prefuse (http://prefuse.org)  Flare (http://flare.prefuse.org)
Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D
Chart Typologies
Excel, Many Eyes, Google Charts

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D
## Data Sets: State Quick Facts

Uploaded By: zingpost  Created at: Friday May 18, 3:08 PM
Data Source: US Census Bureau
Description:
Tags: people census

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Alabama</td>
<td>4557808</td>
<td>0.03</td>
<td>4447100</td>
<td>0.1</td>
<td>0.07</td>
<td>0.24</td>
<td>0.13</td>
</tr>
<tr>
<td>2 Alaska</td>
<td>635661</td>
<td>0.06</td>
<td>626932</td>
<td>0.14</td>
<td>0.08</td>
<td>0.29</td>
<td>0.06</td>
</tr>
<tr>
<td>3 Arizona</td>
<td>6939292</td>
<td>0.16</td>
<td>6130632</td>
<td>0.4</td>
<td>0.08</td>
<td>0.27</td>
<td>0.13</td>
</tr>
<tr>
<td>4 Arkansas</td>
<td>2779154</td>
<td>0.04</td>
<td>2673400</td>
<td>0.14</td>
<td>0.07</td>
<td>0.25</td>
<td>0.14</td>
</tr>
<tr>
<td>5 California</td>
<td>36132147</td>
<td>0.07</td>
<td>33671648</td>
<td>0.14</td>
<td>0.07</td>
<td>0.27</td>
<td>0.11</td>
</tr>
<tr>
<td>6 Colorado</td>
<td>4866177</td>
<td>0.08</td>
<td>4301261</td>
<td>0.31</td>
<td>0.07</td>
<td>0.25</td>
<td>0.11</td>
</tr>
<tr>
<td>7 Connecticut</td>
<td>3510297</td>
<td>0.03</td>
<td>3405865</td>
<td>0.04</td>
<td>0.06</td>
<td>0.24</td>
<td>0.14</td>
</tr>
<tr>
<td>8 Delaware</td>
<td>843524</td>
<td>0.09</td>
<td>7895600</td>
<td>0.18</td>
<td>0.07</td>
<td>0.23</td>
<td>0.13</td>
</tr>
<tr>
<td>9 Florida</td>
<td>17788884</td>
<td>0.11</td>
<td>16882378</td>
<td>0.24</td>
<td>0.06</td>
<td>0.23</td>
<td>0.17</td>
</tr>
<tr>
<td>10 Georgia</td>
<td>9072576</td>
<td>0.11</td>
<td>8186453</td>
<td>0.26</td>
<td>0.08</td>
<td>0.26</td>
<td>0.1</td>
</tr>
<tr>
<td>11 Hawaii</td>
<td>1275194</td>
<td>0.05</td>
<td>1211537</td>
<td>0.08</td>
<td>0.07</td>
<td>0.24</td>
<td>0.14</td>
</tr>
<tr>
<td>12 Idaho</td>
<td>1429095</td>
<td>0.1</td>
<td>1253953</td>
<td>0.29</td>
<td>0.07</td>
<td>0.27</td>
<td>0.11</td>
</tr>
<tr>
<td>13 Illinois</td>
<td>12763371</td>
<td>0.03</td>
<td>12415293</td>
<td>0.09</td>
<td>0.07</td>
<td>0.25</td>
<td>0.12</td>
</tr>
</tbody>
</table>
Choosing a visualization type for State Quick Facts

Analyze a text

**Tag Cloud**
How are you using your words? This enhanced tag cloud will show you the words popularity in the given set of text.
Learn more

**Wordle**
Wordle is a toy for generating "word clouds" from text that you provide. The clouds give greater prominence to words that appear more frequently in the source text.
Learn more

**Word Tree**
See a branching view of how a word or phrase is used in a text. Navigate the text by zooming and clicking.
Learn more

Compare a set of values

**Bar Chart**
How do the items in your data set stack up? A bar chart is a simple and recognizable way to compare values. You can display several sets of bars for multivariate comparisons.
Learn more

**Block Histogram**
This versatile chart lets you get a quick sense of how a single set of data is distributed. Each item in the data is an individually identifiable block.
Learn more
Visualizations: Federal Spending by State, 2004

Creator: Anonymous
Tags: census people

Federal spending 2004 ($1000)
States colored by People QuickFacts

Bubble Size: Federal spending 2004 ($1000)
Label: People QuickFacts
Color: People QuickFacts

Data file: Federal spending 2004 ($1000)
Land area 2000 (square miles)
Persons per square mile 2000
FIPS Code

Comments: This data set has not yet been rated.
Every Wednesday, when I get home from school, I have a piano lesson. My teacher is a very strict house. Her name is Hillary Clinton. Our piano is a Steinway Concert tree and it has 88 cups. It also has a soft pedal and a/an Smily pedal. When I have a lesson, I sit down on the piano Alberto and play for 16 minutes. I do scales to exercise my cats, and then I usually play a minuet by Johann Sebastian Washington. Teacher says I am a natural Haunted House and have a good musical leg. Perhaps when I get better I will become a concert vet and give a recital at Carnegie hospital.
Grammars

- *Context-free grammars* good for describing textual dialogs which have a formal syntax.
  - “BNF” for languages (Backus–Naur Form)
  - alphabet of symbols
  - sentences - legal sequences of symbols
  - language - set of all legal sentences
  - grammar - set of terminals, set of non-terminals, set of productions, start symbol
  - context-free-grammar: all productions of the form \(<a> ::= B\) where \(<a>\) is a single nonterminal, and \(B\) is a non-empty string of terminals and/or non-terminals
Most charting packages channel user requests into a **rigid array of chart types**. To atone for this lack of flexibility, they offer a kit of post-creation editing tools to return the image to what the user originally envisioned. **They give the user an impression of having explored data rather than the experience.**

Chart Typologies
Excel, Many Eyes, Google Charts

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D
ggplot(diamonds, aes(x=price, fill=cut)) + geom_bar(position="dodge")
ggplot(diamonds, aes(x=price, fill=cut)) + geom_bar(position="dodge")
qplot(long, lat, data = expo, geom = "tile", fill = ozone, 
       facets = year ~ month) +
scale_fill_gradient(low = "white", high = "black") + map
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D

Ease-of-Use
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2

Visualization Grammars
Protovis, D3.js

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D
Protovis & D3
Today's first task is not to invent wholly new [graphical] techniques, though these are needed. Rather we need most vitally to recognize and reorganize the essential of old techniques, to make easy their assembly in new ways, and to modify their external appearances to fit the new opportunities.
Protovis: A Grammar for Visualization

A graphic is a composition of data-representative marks.

Jeffrey Heer, Mike Bostock & Vadim Ogievetsky
Visualization Grammar
Visualization Grammar

Data

Input data to visualize
Visualization Grammar

Data  Input data to visualize
Transforms  Grouping, stats, projection, layout
Visualization Grammar

Data
Input data to visualize

Transforms
Grouping, stats, projection, layout

Scales
Map data values to visual values

Jacques Bertin
Sémiologie Graphique, 1967
Visualization Grammar

Data: Input data to visualize
Transforms: Grouping, stats, projection, layout
Scales: Map data values to visual values
Guides: Axes & legends visualize scales
Visualization Grammar

Data
Input data to visualize

Transforms
Grouping, stats, projection, layout

Scales
Map data values to visual values

Guides
Axes & legends visualize scales

Marks
Data-representative graphics

Area
Rect
Symbol
Image

Line
Text
Rule
Arc
Deconstructions
Exports and Imports to and from Denmark & Norway from 1700 to 1780.
William Playfair, 1786

X-axis: year (Q)
Y-axis: currency (Q)
Color: imports,exports (N, O)
Wattenberg’s Map of the Market

Rectangle Area: market cap (Q)
Rectangle Position: market sector (N), market cap (Q)
Color Hue: loss vs. gain (N, O)
Color Value: magnitude of loss or gain (Q)
Minard 1869: Napoleon’s March
Single-Axis Composition
Mark Composition

Y-axis: temperature (Q)

X-axis: longitude (Q) / time (O)

Temp over space/time (Q x Q)
Mark Composition

Y-axis: longitude (Q)

X-axis: latitude (Q)

Width: army size (Q)

Army position (Q x Q) and army size (Q)
Minard 1869: Napoleon’s March

Depicts at least 5 quantitative variables. Any others?
MARKS: Protovis graphical primitives
<table>
<thead>
<tr>
<th>MARK</th>
<th>$\lambda : D \rightarrow R$</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>$\lambda$</td>
</tr>
<tr>
<td>visible</td>
<td>$\lambda$</td>
</tr>
<tr>
<td>left</td>
<td>$\lambda$</td>
</tr>
<tr>
<td>bottom</td>
<td>$\lambda$</td>
</tr>
<tr>
<td>width</td>
<td>$\lambda$</td>
</tr>
<tr>
<td>height</td>
<td>$\lambda$</td>
</tr>
<tr>
<td>fillStyle</td>
<td>$\lambda$</td>
</tr>
<tr>
<td>strokeStyle</td>
<td>$\lambda$</td>
</tr>
<tr>
<td>lineWidth</td>
<td>$\lambda$</td>
</tr>
<tr>
<td>...</td>
<td>$\lambda$</td>
</tr>
</tbody>
</table>
### RECT

<table>
<thead>
<tr>
<th>data</th>
<th>1</th>
<th>1.2</th>
<th>1.7</th>
<th>1.5</th>
<th>0.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>visible</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>left</td>
<td>$\lambda$: index * 25</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bottom</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>width</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>height</td>
<td>$\lambda$: datum * 80</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fillStyle</td>
<td>blue</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>strokeStyle</td>
<td>black</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lineWidth</td>
<td>1.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
### RECT

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>1 1.2 1.7 1.5 0.7</td>
</tr>
<tr>
<td>visible</td>
<td>true</td>
</tr>
<tr>
<td>left</td>
<td>0 * 25</td>
</tr>
<tr>
<td>bottom</td>
<td>0</td>
</tr>
<tr>
<td>width</td>
<td>20</td>
</tr>
<tr>
<td>height</td>
<td>1 * 80</td>
</tr>
<tr>
<td>fillStyle</td>
<td>blue</td>
</tr>
<tr>
<td>strokeStyle</td>
<td>black</td>
</tr>
<tr>
<td>lineWidth</td>
<td>1.5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### RECT

<table>
<thead>
<tr>
<th>data</th>
<th>1</th>
<th>1.2</th>
<th>1.7</th>
<th>1.5</th>
<th>0.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>visible</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>left</td>
<td>1 * 25</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bottom</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>width</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>height</td>
<td>1.2 * 80</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fillStyle</td>
<td>blue</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>strokeStyle</td>
<td>black</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lineWidth</td>
<td>1.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \lambda : D \rightarrow R \]
<table>
<thead>
<tr>
<th>RECT</th>
<th>( \lambda : D \rightarrow R )</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>1</td>
</tr>
<tr>
<td>visible</td>
<td>true</td>
</tr>
<tr>
<td>left</td>
<td>2 * 25</td>
</tr>
<tr>
<td>bottom</td>
<td>0</td>
</tr>
<tr>
<td>width</td>
<td>20</td>
</tr>
<tr>
<td>height</td>
<td>1.7 * 80</td>
</tr>
<tr>
<td>fillStyle</td>
<td>blue</td>
</tr>
<tr>
<td>strokeStyle</td>
<td>black</td>
</tr>
<tr>
<td>lineWidth</td>
<td>1.5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>RECT</td>
<td>λ : D → R</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td>data</td>
</tr>
<tr>
<td></td>
<td>visible</td>
</tr>
<tr>
<td></td>
<td>left</td>
</tr>
<tr>
<td></td>
<td>bottom</td>
</tr>
<tr>
<td></td>
<td>width</td>
</tr>
<tr>
<td></td>
<td>height</td>
</tr>
<tr>
<td></td>
<td>fillStyle</td>
</tr>
<tr>
<td></td>
<td>strokeStyle</td>
</tr>
<tr>
<td></td>
<td>lineWidth</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
</tr>
<tr>
<td>data</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>1.7</td>
</tr>
<tr>
<td></td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
</tr>
<tr>
<td>visible</td>
<td>true</td>
</tr>
<tr>
<td>left</td>
<td>4 * 25</td>
</tr>
<tr>
<td>bottom</td>
<td>0</td>
</tr>
<tr>
<td>width</td>
<td>20</td>
</tr>
<tr>
<td>height</td>
<td>0.7 * 80</td>
</tr>
<tr>
<td>fillStyle</td>
<td>blue</td>
</tr>
<tr>
<td>strokeStyle</td>
<td>black</td>
</tr>
<tr>
<td>lineWidth</td>
<td>1.5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

$\lambda : D \rightarrow R$
<table>
<thead>
<tr>
<th>RECT</th>
<th>( \lambda : D \rightarrow R )</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>1.2 1.7 1.5 0.7</td>
</tr>
<tr>
<td>visible</td>
<td>true</td>
</tr>
<tr>
<td>left</td>
<td>( \lambda : \text{index} \times 25 )</td>
</tr>
<tr>
<td>bottom</td>
<td>0</td>
</tr>
<tr>
<td>width</td>
<td>20</td>
</tr>
<tr>
<td>height</td>
<td>( \lambda : \text{datum} \times 80 )</td>
</tr>
<tr>
<td>fillStyle</td>
<td>blue</td>
</tr>
<tr>
<td>strokeStyle</td>
<td>black</td>
</tr>
<tr>
<td>lineWidth</td>
<td>1.5</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
var vis = new pv.Panel();
vis.add(pv.Bar)
  .data([1, 1.2, 1.7, 1.5, 0.7])
  .visible(true)
  .left((d) => this.index * 25);
  .bottom(0)
  .width(20)
  .height((d) => d * 80)
  .fillStyle("blue")
  .strokeStyle("black")
  .lineWidth(1.5);
vis.render();
```javascript
var army = pv.nest(napoleon.army, "dir", "group");
var vis = new pv.Panel();

var lines = vis.add(pv.Panel).data(army);
lines.add(pv.Line)
  .data(() => army[this.idx])
  .left(lon).top(lat)
  .size(d => d.size/8000)
  .strokeStyle(() => color[army[paneIndex][0].dir]);

vis.add(pv.Label).data(napoleon.cities)
  .left(lon).top(lat)
  .text((d) => d.city).font("italic 10px Georgia")
  .textAlign("center").textBaseline("middle");

vis.add(pv.Rule).data([[0,-10,-20,-30]])
  .top(d => 300 - 2*d - 0.5).left(200).right(150)
  .lineWidth(1).strokeStyle("#ccc")
  .anchor("right").add(pv.Label)
  .top((d) => 5 + tmp(d))
  .text((d) => d.temp+"° "+d.date.substr(0,6));
```

PRELUDE NO.1 IN C MAJOR, BWV 846
(FROM WELL-TEMPERED CLAVIER, BOOK 1)
by J.S. BACH

Bach’s Prelude #1 in C Major | Jieun Oh
Protovis

*Specialized mark types*

+ Streamlined design
- Limits expressiveness
- More overhead (slower)
- Harder to debug
- Self-contained model

*Specify a scene (nouns)*

+ Quick for static vis
- Delayed evaluation
- Animation, interaction are more cumbersome
## Protovis

*Specialized mark types*
- Streamlined design
- Limits expressiveness
- More overhead (slower)
- Harder to debug
- Self-contained model

*Specify a scene (nouns)*
- Quick for static vis
- Delayed evaluation
- Animation, interaction are more cumbersome

## D3

*Bind data to DOM*
- Exposes SVG/CSS/…
- More complex model
- Immediate evaluation
- Dynamic data, anim, and interaction natural

*Transform a scene (verbs)*
- Less overhead (faster)
- Debug in browser
- Use with other tools
D3 Selections

The core abstraction in D3 is a *selection*.
D3 Selections

The core abstraction in D3 is a selection.

// Add and configure an SVG element
var svg = d3.append("svg")  // add new SVG to page body
  .attr("width", 500)       // set SVG width to 500px
  .attr("height", 300);    // set SVG height to 300px
The core abstraction in D3 is a selection.

```javascript
// Add and configure an SVG element
var svg = d3.append("svg")
  .attr("width", 500)  // set SVG width to 500px
  .attr("height", 300); // set SVG height to 300px

// Select & update existing rectangles contained in the SVG element
svg.selectAll("rect")
  .attr("width", 100)  // set rect widths to 100px
  .style("fill", "steelblue"); // set rect fill colors
```
Data Binding

Selections can *bind data and DOM elements.*

```javascript
var values = [ {...}, {...}, {...}, ... ]; // input data as JS objects
```
Data Binding

Selections can *bind* data and DOM elements.

```javascript
var values = [ {…}, {…}, {…}, … ]; // input data as JS objects

// Select SVG rectangles and bind them to data values.
var bars = svg.selectAll("rect.bars").data(values);
```
Data Binding

Selections can **bind** data and DOM elements.

```javascript
var values = [ {…}, {…}, {…}, … ]; // input data as JS objects

// Select SVG rectangles and bind them to data values.
var bars = svg.selectAll("rect.bars").data(values);

// What if the DOM elements don’t exist yet? The `enter` set represents data values that do not yet have matching DOM elements.
bars.enter().append("rect").attr("class", "bars");
```
Data Binding

Selections can **bind data and DOM elements**.

```javascript
var values = [ {…}, {…}, {…}, … ]; // input data as JS objects

// Select SVG rectangles and bind them to data values.
var bars = svg.selectAll("rect.bars").data(values);

// What if the DOM elements don’t exist yet? The **enter** set represents data
// values that do not yet have matching DOM elements.
bars.enter().append("rect").attr("class", "bars");

// What if data values are removed? The **exit** set is a selection of existing
// DOM elements who no longer have matching data values.
bars.exit().remove();
```
The Data Join

ENTER
Data values without matching DOM elements.

DATA VALUES

UPDATE
Existing DOM elements, bound to valid data.

ELEMENTS

EXIT
DOM elements whose bound data has gone “stale”.

DATA VALUES

ELEMENTS
The Data Join

\[ \text{var } s = \text{d3.selectAll(...).data(...)} \]

**ENTER**
- Data values without matching DOM elements.
  \[ s.\text{enter().append(...)} \]

**UPDATE**
- Existing DOM elements, bound to valid data.
  \[ s \]

**EXIT**
- DOM elements whose bound data has gone "stale".
  \[ s.\text{exit()} \]
D3 Modules

Data Parsing / Formatting (JSON, CSV, ...)
Shape Helpers (arcs, curves, areas, symbols, ...)
Scale Transforms (linear, log, ordinal, ...)
Color Spaces (RGB, HSL, LAB, ...)
Animated Transitions (tweening, easing, ...)
Geographic Mapping (projections, clipping, ...)
Layout Algorithms (stack, pie, force, trees, ...)
Interactive Behaviors (brush, zoom, drag, ...)
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2

Visualization Grammars
Protovis, D3.js

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D
A Visualization Tool Stack
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2

Visualization Grammars
Protovis, D3.js

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2

Visualization Grammars
Protovis, D3.js

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2

Visualization Grammars
Protovis, D3.js

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D
Declarative Languages

- For dialog boxes, forms and menus, just list the contents
- But need to also list a *lot* of properties for each field
- Used by
  - CMU's COUSIN system (~1985), and
  - Apollo's Domain/Dialog (later HP/Apollo's Open Dialog) ~1985-87.
  - Html forms (to some extent)
- Goals:
  - less overall effort
  - Better separation UI and application
  - Multiple interfaces for same application
  - Consistency since UI part uses same package
What is a Declarative Language?
What is a Declarative Language?

Programming by describing *what*, not *how*
What is a Declarative Language?

Programming by describing *what*, not *how*

Separate **specification** (*what you want*) from **execution** (*how it should be computed*)
What is a Declarative Language?

Programming by describing what, not how

Separate specification (what you want) from execution (how it should be computed)

In contrast to imperative programming, where you must give explicit steps.
What is a Declarative Language?

Programming by describing *what*, not *how*

Separate **specification** (*what you want*) from **execution** (*how it should be computed*)

In contrast to **imperative programming**, where you must give explicit steps.

```javascript
d3.selectAll("rect")
  .data(my_data)
  .enter().append("rect")
  .attr("x", (d) => xscale(d.foo))
  .attr("y", (d) => yscale(d.bar))
```
SELECT customer_id, customer_name, COUNT(order_id) as total
FROM customers
INNER JOIN orders ON customers.customer_id = orders.customer_id
GROUP BY customer_id, customer_name
HAVING COUNT(order_id) > 5
ORDER BY COUNT(order_id) DESC
Why Declarative Languages?

Better visualization? *Smart defaults.*

**Reuse.** *Write-once, then re-apply.*

**Performance.** *Optimization, scalability.*

**Portability.** *Multiple devices, renderers, inputs.*

**Programmatic generation.**

*Write programs which output visualizations.*

*Automated search & recommendation.*
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2

Visualization Grammars
Protovis, D3.js

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D
Chart Typologies
- Excel, Many Eyes, Google Charts

Visual Analysis Grammars
- VizQL, ggplot2, **Vega-Lite**

Visualization Grammars
- Protovis, D3.js, **Vega**

Component Architectures
- Prefuse, Flare, Improvise, VTK

Graphics APIs
- Processing, OpenGL, Java2D
Chart Typologies
Excel, Many Eyes, Google Charts

Visual Analysis Grammars
VizQL, ggplot2, **Vega-Lite**

Visualization Grammars
Protovis, D3.js, **Vega**

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D
Visual Analysis Grammars
- VizQL, ggplot2, **Vega-Lite**

Visualization Grammars
- Protovis, D3.js, **Vega**

Component Architectures
- Prefuse, Flare, Improvise, VTK

Graphics APIs
- Processing, OpenGL, Java2D
Interactive Data Exploration
Tableau, *Lyra, Voyager*

Visual Analysis Grammars
VizQL, ggplot2, *Vega-Lite*

Visualization Grammars
Protovis, D3.js, *Vega*

Component Architectures
Prefuse, Flare, Improvise, VTK

Graphics APIs
Processing, OpenGL, Java2D
Grammar of Graphics for Customized Designs

Offer fine-grained control for composing interactive graphics.

But require technical expertise and verbose specifications (e.g., 100 LOCs for a bar chart)
Grammar of Graphics for Exploration

Facilitate rapid exploration with concise specifications by omitting low-level details.

Infer sensible defaults and allow customization by overriding defaults.
Vega-Lite's Mission

Facilitate exploratory data analysis with an expressive yet concise language to specify interactive multi-view graphics
Visualization Grammar

Data
Input data to visualize

Transforms
Grouping, stats, projection, layout

Scales
Map data values to visual values

Guides
Axes & legends visualize scales

Marks
Data-representative graphics
Specifying Visualizations in Vega-Lite

Abstract Data  →  Visual Representation
Specifying Visualizations in Vega-Lite

Abstract Data

Weather Data for Seattle

<table>
<thead>
<tr>
<th>date</th>
<th>temperature</th>
<th>precipitation</th>
<th>weather</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/1</td>
<td>10.6</td>
<td>10.9</td>
<td>&quot;rain&quot;</td>
</tr>
<tr>
<td>1/2</td>
<td>11.7</td>
<td>0.8</td>
<td>&quot;drizzle&quot;</td>
</tr>
<tr>
<td>1/3</td>
<td>12.2</td>
<td>10.2</td>
<td>&quot;rain&quot;</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

Visual Representation

Strip Plot of Temperature
Strip Plot = (**Tick** with \(x\)=field)

```javascript
{
  data: {url: "weather-seattle.json"},
  mark: "tick",
  encoding: {
    x: {
      field: "temperature",
      type: "quantitative"
    }
  }
}
```
Strip Plot = (Tick with x=field)

Vega-Lite is portable JSON specification
Strip Plot = (Tick with x=field)

```json
{
  data: {url: "weather-seattle.json"},
  mark: "tick",
  encoding: {
    x: {
      field: "temperature",
      type: "quantitative"
    }
  }
}
```
Strip Plot: Default Scales and Axes

```json
{
  data: {url: "weather-seattle.json"},
  mark: "tick",
  encoding: {
    x: {
      field: "temperature",
      type: "quantitative",
      scale: {type: "linear", domain: [-10, 40], ...}
    }
  }
}
```
Strip Plot

{  
data: {url: "weather-seattle.json"},  
mark: "tick",  
encoding: {  
  x: {  
    field: "temperature",  
    type: "quantitative"  
  }  
}
}

How many days?

temperature
Histogram

Goal
Histogram = (Bar with \(x=\text{binned field}, y=\text{count}\))
Histogram = (Bar with $x=$ binned field, $y=$ count)

```
{
  data: {url: "weather-seattle.json"},
  mark: "tick",
  encoding: {
    x: {
      field: "temperature",
      type: "quantitative"
    }
  }
}
```
Histogram = (Bar with $x=binned\ field$, $y=count$)

```
{
  data: {url: "weather-seattle.json"},
  mark: "tick",
  encoding: {
    x: {
      bin: true,
      field: "temperature",
      type: "quantitative"
    }
  }
}
```
Histogram = (Bar with \( x=binned \) field, \( y=count \))

```json
{
  data: {url: "weather-seattle.json"},
  mark: "tick",
  encoding: {
    x: {
      bin: true,
      field: "temperature",
      type: "quantitative"
    },
    y: {
      aggregate: "count",
      type: "quantitative"
    }
  }
}
```
Histogram = (Bar with $x=binned\ field$, $y=count$)

```json
{
    data: {url: "weather-seattle.json"},
    mark: "bar",
    encoding: {
        x: {
            bin: true,
            field: "temperature",
            type: "quantitative"
        },
        y: {
            aggregate: "count",
            type: "quantitative"
        }
    }
}
```
Histogram = (Bar with $x=$binned field, $y=$count)

```json
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {
      bin: true,
      field: "temperature",
      type: "quantitative"
    },
    y: {
      aggregate: "count",
      type: "quantitative"
    }
  }
}
```

```sql
SELECT bin_temp, count(*)
FROM (SELECT floor(weather.temp, range.min, range.max, 9) bin_temp
      FROM weather,
      (SELECT min(temp) min, max(temp) max
       ) range)
```
**Sensible Defaults for Binning**

Channel determines guide and bin parameters

<table>
<thead>
<tr>
<th>Guide</th>
<th>Color/Opacity/Shape</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Bins</td>
<td>Legend with range labels</td>
<td>Quantitative axis</td>
</tr>
<tr>
<td></td>
<td>Fewer bins</td>
<td>Many bins</td>
</tr>
</tbody>
</table>

**Hottest Temperature**

-10—0

50

0

5 10 15 20 25 30 35 40

**BIN(temperature)**
Histogram

```javascript
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {
      bin: true,
      field: "temperature",
      type: "quantitative"
    },
    y: {
      aggregate: "count",
      type: "quantitative"
    }
  }
}
```
Histogram + Color

```json
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {
      bin: true,
      field: "temperature",
      type: "quantitative"
    },
    y: {
      aggregate: "count",
      type: "quantitative"
    },
    color: {
      field: "weather",
      type: "nominal"
    }
  }
}
```
Histogram + Color

```json
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {
      bin: true,
      field: "temperature",
      type: "quantitative"
    },
    y: {
      aggregate: "count",
      type: "quantitative"
    },
    color: {
      field: "weather",
      type: "nominal",
      "scale": {
        "domain": ["sun", "fog", "drizzle", "rain", "snow"],
        "range": ["#e7ba52", "#c7c7c7", "#aec7e8", "#1f77b4", "#9467bd"]
      }
    }
  }
}
```
Histogram + Color = Stacked Histogram

```json
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {
      bin: true,
      field: "temperature",
      type: "quantitative"
    },
    y: {
      aggregate: "count",
      type: "quantitative"
    },
    color: {
      field: "weather",
      type: "nominal",
      ...""
    }
  }
}
```
Stacked Histogram: **Sensible Defaults**

**no stack**

**stack (default)**
Stacked Histogram: **Sensible Defaults**

Channel (color) + Mark (bar) automatically enables stacking: a layout transform.
Stacked Histogram: **Sensible Defaults**

Channel (color) + Mark (bar) automatically enables stacking: a layout transform.

**no stack → overlap**

**stack (default)**
Histogram + Color = Stacked Histogram

```json
{
    data: {
        url: "weather-seattle.json",
        mark: "bar",
        encoding: {
            x: {
                bin: true,
                field: "temperature",
                type: "quantitative"
            },
            y: {
                aggregate: "count",
                type: "quantitative"
            },
            color: {
                field: "weather",
                type: "nominal"
            }
        }
    }
}
```

hard to compare without common baseline
Histogram + Color = Stacked Histogram

```javascript
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {bin: true, field: "temperature", type: "quantitative"},
    y: {aggregate: "count", type: "quantitative"},
    color: {field: "weather", type: "nominal"}
  }
}
```
Histogram + **Column** = **Trellis Histogram**

```json
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {bin: true, field: "temperature", type: "quantitative"},
    y: {aggregate: "count", type: "quantitative"},
    column: { field: "weather", type: "nominal"}
  }
}
```
Vega-Lite is an Expressive Language.
Vega-Lite is an Expressive Language for **Statistical** Graphics.
Vega-Lite is an Expressive Language for Statistical **Multi-View** Graphics.

Seattle Weather, 2012-2015

- **Date**
- **Temperature (°C)**
- **Precipitation**
  - 0
  - 10
  - 20
  - 30
  - 40
  - 50
- **Weather**
  - Sun
  - Fog
  - Drizzle
  - Rain
  - Snow

Layering & Error Marks

Concatenation
Vega-Lite is an Expressive Language for Statistical Interactive Multi-View Graphics.
Vega-Lite – A Grammar of Interactive Graphics

Vega-Lite is a high-level grammar of interactive graphics. It provides a concise JSON syntax for rapidly generating visualizations to support analysis. Vega-Lite specifications can be compiled to Vega specifications.

Vega-Lite specifications describe visualizations as mappings from data to properties of graphical marks (e.g., points or bars). The Vega-Lite compiler automatically produces visualizations in components including axes, legends, and scales. It then determines properties of these components based on a set of carefully designed rules. This approach allows specifications to be succinct and expressive, but also provide user control. As Vega-Lite is designed for analysis, it supports data transformations such as aggregation, binning, filtering, sorting, and visual transform metrics including stacking and faceting. Moreover, Vega-Lite specifications can be composed into layered and multi-view displays, and made interactive with selections.

Read our Introduction article to Vega-Lite v2 on Medium, watch our OpenVizConf talk about the new features in Vega-Lite v1, check out the documentation and take a look at our example gallery. Follow us on Twitter at @vega_viz to stay informed about updates.

Example

With Vega-Lite, we can start with a bar chart of the average monthly precipitation in Seattle, overlay a line for the overall yearly average, and have it represent an interactive moving average for a dragged region.

vega.github.io/vega-lite
Recap: Vega-Lite Design Principles
As a Tool

Favor composition over templates

Provide sensible defaults, but allow customization

Automatic optimization of visual presentation and processing

Support programmatic generation, sharing, and reuse

https://medium.com/hci-design-at-uw/introducing-vega-lite-438f9215f09e

https://medium.com/@uwdata/introducing-vega-lite-2-0-de6661c12d58
Recap: Vega-Lite Design Principles

As a Language

**Approachable.** You don’t need to understand everything before you can understand anything.

**Consistent.** Building blocks can be reused.

**Explains itself.** Affordances like signposts.

**Teaches.** Vega-Lite implements best practices.

**For humans.** Reduce surprises and anticipate the need for learning and debugging.

https://medium.com/@mbostock/what-makes-software-good-943557f8a488
Is Vega-Lite really easier?

Let's make a histogram

Vega-Lite

```json
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {
      bin: true,
      field: "temperature",
      type: "quantitative"
    },
    y: {
      aggregate: "count",
      type: "quantitative"
    }
  }
}
```

Chart Typology (e.g. Seaborn)

```python
sns.distplot(x);
```

The code looks simple but brushes complexity under the rug.

It does not teach you that a histogram is a bar chart.

Cannot be adapted to use different aggregation (e.g. sum).
What’s next?

Work we didn’t have time to talk about today

Protovis, D3, Vega, Vega-Lite, etc. allow us to create a combinatorial number of visualizations. Can tools help us make good charts?
→ Draco [Moritz et. al. 2018]

Use in data science environments (e.g. Python) → Altair (altair-viz.github.io)

Make animated visualizations.

Scale visualizations to large data.

Visualization tools (Tableau, Voyager, etc…).

Debugging tools → [Hoffswell et al. EuroVis’16]
What you can do now:

Articulate the difference between chart typologies and component architectures.
Decompose a graphic into the building blocks of visualization.
Know why most visualization toolkits are declarative.
Explain tradeoff between expressiveness and ease-of-use.

Know the history of visualization tools.
Understand the basics of D3.
Give examples of manifestations of design principles in Vega-Lite.