Visualization for People + Systems

Dominik Moritz @domoritz
University of Washington
import matplotlib.pyplot as plt
import pandas as pd

raw_df = pd.read_csv("weather.csv", parse_dates=True)

# filter to Seattle
df = raw_df[raw_df.location == 'Seattle']

# extract month and year
df['month'] = pd.DatetimeIndex(df.date).month
df['year'] = pd.DatetimeIndex(df.date).year

# group data and flatten it again
gb = df.groupby(['month']).temperature.mean()
grouped_df = gb.reset_index()

# initialize chart
fig, ax = plt.subplots()

# draw data as line
ax.plot(grouped_df.month.values, grouped_df.temperature.values)

# set title and axes
ax.set_title('Temperature in Seattle')
ax.set_ylabel('Temperature')
ax.set_xlabel('Month')
fig.show()
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv("weather.csv", parse_dates=True)

# extract month and year
df['month'] = pd.DatetimeIndex(df.date).month
df['year'] = pd.DatetimeIndex(df.date).year

# group data and flatten it again
gb = df.groupby(['month', 'location']).temperature.mean()
grouped_df = gb.reset_index()

color_map = dict(zip(df.location.unique(), ['orange', 'steelblue']))

# initialize chart
fig, ax = plt.subplots()

# draw data as line for each city
for city in df.location.unique():
    filtered_df = grouped_df[grouped_df.location == city]
    ax.plot(filtered_df.month.values, filtered_df.temperature.values,
            c=color_map[city], label=city)

# set axes and legend
ax.set_title('Temperature in Seattle and New York')
ax.legend(frameon=True, title='City')
ax.set_ylabel('Temperature')
ax.set_xlabel('Month')

fig.show()
Visualize Weather Data for Seattle and New York

```python
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv("weather.csv", parse_dates=True)

# extract month and year
df['month'] = pd.to_datetime(df['date'], format='%Y-%m-%d').month
df['year'] = pd.to_datetime(df['date'], format='%Y-%m-%d').year

# group data and flatten it again
gb = df.groupby(['month', 'location']).temperature.mean()
grouped_df = gb.reset_index()

# create color map
color_map = dict(zip(df.location.unique(), ['orange', 'steelblue']))

# initialize chart
fig, ax = plt.subplots()

# draw data as line for each city
for city in df.location.unique():
    filtered_df = grouped_df[grouped_df.location == city]
    ax.plot(filtered_df.month.values, filtered_df.temperature.values, c=color_map[city], label=city)

# set axes and legend
ax.set_title('Temperature in Seattle and New York')
ax.legend(frameon=True, title='City')
ax.set_ylabel('Temperature')
ax.set_xlabel('Month')
fig.show()
```
Visualize Weather Data for Seattle and New York

```python
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv("weather.csv", parse_dates=True)

# extract month and year
df['month'] = pd.DatetimeIndex(df.date).month
df['year'] = pd.DatetimeIndex(df.date).year

# group data and flatten it again
gb = df.groupby(["month", "location"]).temperature.mean()
grouped_df = gb.reset_index()

# create color map
color_map = dict(zip(df.location.unique(), ['orange', 'steelblue']))

# initialize chart
fig, ax = plt.subplots()

# draw data as line for each city
for city in df.location.unique():
    filtered_df = grouped_df[grouped_df.location == city]
    ax.plot(filtered_df.month.values, filtered_df.temperature.values,
            color=color_map[city], label=city)

# set axes and legend
ax.set_title('Temperature in Seattle and New York'
ax.legend(frameon=True, title='City')
ax.set_ylabel('Temperature')
ax.set_xlabel('Month')

fig.show()
```
Visualize Weather Data for Seattle and New York

```python
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv("weather.csv", parse_dates=True)

# extract month and year
df['month'] = pd.DatetimeIndex(df.date).month
df['year'] = pd.DatetimeIndex(df.date).year

# group data and flatten it again
gb = df.groupby(['month', 'location']).temperature.mean()
grouped_df = gb.reset_index()

# create color map
color_map = dict(zip(df.location.unique(), ['orange', 'steelblue']))

# initialize chart
fig, ax = plt.subplots()

# draw data as line for each city
for city in df.location.unique():
    filtered_df = grouped_df[grouped_df.location == city]
    ax.plot(filtered_df.month.values, filtered_df.temperature.values,
            c=color_map[city], label=city)

# set axes and legend
ax.set_title('Temperature in Seattle and New York')
ax.legend(frameon=True, title='City')
ax.set_ylabel('Temperature')
ax.set_xlabel('Month')

fig.show()
```

In R or JavaScript, this code would look very different.

Create color map

Draw a line for each city

Don't forget the legend!

The plot is static
Visualize Weather Data for Seattle and New York

import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv("weather.csv", parse_dates=True)

# extract month and year
df['month'] = pd.DatetimeIndex(df['date']).month
df['year'] = pd.DatetimeIndex(df['date']).year

# group data and flatten it again
gb = df.groupby(['month', 'location']).temperature.mean()
grouped_df = gb.reset_index()

# create color map
color_map = dict(zip(df.location.unique(), ['orange', 'steelblue']))

# initialize chart
fig, ax = plt.subplots()

# draw data as line for each city
for city in df.location.unique():
    filtered_df = grouped_df[grouped_df.location == city]
    ax.plot(filtered_df.month.values, filtered_df.temperature.values,
            c=color_map[city], label=city)

# set axes and legend
ax.set_title('Temperature in Seattle and New York')
ax.legend(frameon=True, title='City')
ax.set_ylabel('Temperature')
ax.set_xlabel('Month')

fig.show()
Visualize Weather Data for Seattle and New York

```python
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv("weather.csv", parse_dates=True)

# extract month and year
df["month"] = pd.DatetimeIndex(df.date).month
df["year"] = pd.DatetimeIndex(df.date).year

# group data and flatten it again
gb = df.groupby(["month", "location"]).temperature.mean()
grouped_df = gb.reset_index()

# create color map
color_map = dict(zip(df.location.unique(), ["orange", "steelblue"]))

# initialize chart
fig, ax = plt.subplots()

# draw data as line for each city
for city in df.location.unique():
    filtered_df = grouped_df[grouped_df.location == city]
    ax.plot(filtered_df.month.values, filtered_df.temperature.values,
            c=color_map[city], label=city)

# set axes and legend
ax.set_title("Temperature in Seattle and New York")
al.figure(frameon=True, title=city)
ax.set_ylabel("Temperature")
ax.set_xlabel("Month")

fig.show()
```

In R or JavaScript, this code would look very different.

Create color map

Draw a line for each city

This is inefficient and won’t scale to large data

Don’t forget the legend!

Is this design good?
For any good design...
...there are many poor designs.
...there are many poor designs.

Poor color choice
...there are many poor designs.

Non-optimal mark
...there are many poor designs.

- Missing legend
- Poor color choice
- Non-optimal mark

- Misleading stacking
- Misleading baseline
- Bad aspect ratio
How do I create the next generation of visualization systems where users can rapidly create good designs?
How do I create the next generation of visualization systems where users can rapidly create good designs?
How do I create the next generation of visualization systems where users can rapidly create good designs regardless of the scale of their data?
How do I create the next generation of visualization systems where users can rapidly create good designs regardless of the scale of their data?
Programming tools are designed for **manual authoring**. Good design is the **responsibility of the human designer**.

Tools do not provide computational guidance.
I design domain-specific languages where *people and systems* can meaningfully participate in the *visualization* process.
Optimizations often rely on abstracting away the person using the computer.

Tools have little understanding of the user's goals.
I design domain-specific languages where people and systems can meaningfully participate in the visualization process.

I leverage an understanding of people’s tasks and capabilities to inform system design.
I design domain-specific languages where people and systems can meaningfully participate in the visualization process.

I leverage an understanding of people’s tasks and capabilities to inform system design.
My Mission:

Develop **tools for data analysis and communication** that richly integrate the strengths of both **people** and **machines**.
**Formal Models of Visualization**

**Vega-Lite** *Infovis 2016. Best Paper*
High-Level grammar for interactive multi-view graphics
Designed for programmatic generation

**Scalable Visualization**

**Falcon** *CHI 2019.*
Real-time linked interactions with billions of records

**Optimistic Visualization** *CHI 2017.*
Fast and reliable approximations for data exploration

**Draco** *Infovis 2018. Best Paper*
Formal reasoning for visualization design
Visualization languages and recommendation

- **Vega-Lite.** Satynarayan, Moritz, Wongsuphasawat et al. *Infovis* 2016. **Best Paper**
- **Draco.** Moritz et al. *Infovis* 2018. **Best Paper**
- **CompassQL.** Wongsuphasawat, Moritz et al. *HILDA* 2015.
- **Voyager.** Wongsuphasawat, Moritz et al. *Infovis* 2015. **Invited to SIGGRAPH**
- **Learning Design.** Saket, Moritz et al. *VisGuides* 2018.

Data science

- **SQLShare.** Jain, Moritz et al. *SIGMOD* 2016.
- **Voronoi.** Schmechel, Moritz et al. *IVAPP* 2014.

Big data (visualization) systems

- **Falcon.** Moritz et al. *CHI* 2019.
- **Myria.** Wang et al. *CIDR* 2017.
- **Myria.** Halperin et al. *SIGMOD Demo* 2014.
- **Dynamic Client-Server Optimization.** Moritz et al. *DSIA* 2015.

Uncertainty

- **VSUP.** Correll, Moritz, Heer. *CHI* 2018.

Debugging systems


Searching genomes

Formal Models of Visualization

**Vega-Lite** *Infovis 2016. Best Paper*
High-Level grammar for interactive multi-view graphics
Designed for programmatic generation

**Draco** *Infovis 2018. Best Paper*
Formal reasoning for visualization design

Scalable Visualization

**Falcon** *CHI 2019.*
Real-time linked interactions with billions of records

**Optimistic Visualization** *CHI 2017.*
Fast and reliable approximations for data exploration
Guidance

- Missing legend
- Poor color choice
- Non-optimal mark
- Misleading stacking
- Misleading baseline
- Bad aspect ratio

Representation
Guidance

Formal model of design knowledge

Representation
Guidance

Formal model of design knowledge

Programmatic generation
- Declarative
- High-level

Representation
Guidance

Formal model of design knowledge

Programmatic generation
- Declarative
- High-level

Representation
Guidance

Automated reasoning

Formal model of design knowledge

Programmatic generation
  Declarative
  High-level

Representation
Guidance

Automated reasoning

Formal model of design knowledge

Programmatic generation
Declarative
High-level

Representation
How do we make visualizations?
## Visualizing Data

### Weather

<table>
<thead>
<tr>
<th>City</th>
<th>Date</th>
<th>Precipitation</th>
<th>Temperature</th>
<th>Wind</th>
<th>Weather</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>January 1, 2012</td>
<td>0.0</td>
<td>12.8</td>
<td>4.7</td>
<td>drizzle</td>
</tr>
<tr>
<td>New York</td>
<td>January 1, 2012</td>
<td>5.1</td>
<td>10.0</td>
<td>4.5</td>
<td>rain</td>
</tr>
<tr>
<td>Seattle</td>
<td>January 2, 2012</td>
<td>10.9</td>
<td>10.6</td>
<td>4.5</td>
<td>rain</td>
</tr>
<tr>
<td>New York</td>
<td>January 2, 2012</td>
<td>0.0</td>
<td>10.1</td>
<td>8.7</td>
<td>sun</td>
</tr>
<tr>
<td>Seattle</td>
<td>January 3, 2012</td>
<td>0.8</td>
<td>11.7</td>
<td>2.3</td>
<td>rain</td>
</tr>
<tr>
<td>New York</td>
<td>January 3, 2012</td>
<td>0.0</td>
<td>0.6</td>
<td>8.2</td>
<td>sun</td>
</tr>
<tr>
<td>Seattle</td>
<td>January 4, 2012</td>
<td>20.3</td>
<td>12.2</td>
<td>4.7</td>
<td>rain</td>
</tr>
</tbody>
</table>
## Visualizing Data

### Weather

- **Select Data Fields**
- **City**
  - **Date**
  - **Temperature**
  - **Wind**
  - **Weather**

### Data Fields

<table>
<thead>
<tr>
<th>City</th>
<th>Date</th>
<th>Precipitation</th>
<th>Temperature</th>
<th>Wind</th>
<th>Weather</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>January 1, 2012</td>
<td>0.0</td>
<td>12.8</td>
<td>4.7</td>
<td>drizzle</td>
</tr>
<tr>
<td>New York</td>
<td>January 1, 2012</td>
<td>5.1</td>
<td>10.0</td>
<td>4.5</td>
<td>rain</td>
</tr>
<tr>
<td>Seattle</td>
<td>January 2, 2012</td>
<td>10.9</td>
<td>10.6</td>
<td>4.5</td>
<td>rain</td>
</tr>
<tr>
<td>New York</td>
<td>January 2, 2012</td>
<td>0.0</td>
<td>10.1</td>
<td>8.7</td>
<td>sun</td>
</tr>
<tr>
<td>Seattle</td>
<td>January 3, 2012</td>
<td>0.8</td>
<td>11.7</td>
<td>8.2</td>
<td>sun</td>
</tr>
<tr>
<td>New York</td>
<td>January 3, 2012</td>
<td>0.0</td>
<td>0.6</td>
<td>8.2</td>
<td>sun</td>
</tr>
<tr>
<td>Seattle</td>
<td>January 4, 2012</td>
<td>58.7</td>
<td>12.2</td>
<td>8.7</td>
<td>rain</td>
</tr>
</tbody>
</table>
Visualizing Data

Data Fields

<table>
<thead>
<tr>
<th>City</th>
<th>Month</th>
<th>MEAN(Temperature)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>January</td>
<td>8.1</td>
</tr>
<tr>
<td>New York</td>
<td>January</td>
<td>9.1</td>
</tr>
<tr>
<td>Seattle</td>
<td>February</td>
<td>8.3</td>
</tr>
<tr>
<td>New York</td>
<td>February</td>
<td>4.4</td>
</tr>
<tr>
<td>Seattle</td>
<td>March</td>
<td>9.8</td>
</tr>
</tbody>
</table>
Visualizing Data

Weather

Select Data Fields

City

Date Temperature

Transform Data

MEAN(Temperature) BY Month of Date, City

Design Encoding

Transformed Data

Visualization

Data

Data Fields

City

Seattle

New York

Temperature

Month
Building Blocks of Visualization

Data

Input data to visualize

`weather.csv`

<table>
<thead>
<tr>
<th>City</th>
<th>Date</th>
<th>Temp.</th>
<th>Prec.</th>
<th>Wind</th>
<th>Weather</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>Jan 1</td>
<td>12.8</td>
<td>0.0</td>
<td>4.7</td>
<td>drizzle</td>
</tr>
<tr>
<td>New York</td>
<td>Jan 1</td>
<td>10.0</td>
<td>5.1</td>
<td>4.5</td>
<td>rain</td>
</tr>
<tr>
<td>Seattle</td>
<td>Jan 2</td>
<td>10.6</td>
<td>10.9</td>
<td>4.5</td>
<td>rain</td>
</tr>
<tr>
<td>New York</td>
<td>Jan 2</td>
<td>10.1</td>
<td>0.0</td>
<td>8.7</td>
<td>sun</td>
</tr>
<tr>
<td>Seattle</td>
<td>Jan 3</td>
<td>11.7</td>
<td>0.8</td>
<td>2.3</td>
<td>rain</td>
</tr>
<tr>
<td>New York</td>
<td>Jan 3</td>
<td>0.6</td>
<td>0.0</td>
<td>8.2</td>
<td>sun</td>
</tr>
<tr>
<td>Seattle</td>
<td>Jan 4</td>
<td>12.2</td>
<td>20.3</td>
<td>4.7</td>
<td>rain</td>
</tr>
<tr>
<td>New York</td>
<td>Jan 4</td>
<td>-1.7</td>
<td>0.0</td>
<td>5.5</td>
<td>sun</td>
</tr>
<tr>
<td>Seattle</td>
<td>Jan 5</td>
<td>8.9</td>
<td>1.3</td>
<td>6.1</td>
<td>rain</td>
</tr>
<tr>
<td>New York</td>
<td>Jan 5</td>
<td>5.6</td>
<td>0.0</td>
<td>5.4</td>
<td>sun</td>
</tr>
<tr>
<td>Seattle</td>
<td>Jan 6</td>
<td>4.4</td>
<td>2.5</td>
<td>2.2</td>
<td>rain</td>
</tr>
<tr>
<td>New York</td>
<td>Jan 6</td>
<td>12.2</td>
<td>2.5</td>
<td>4.6</td>
<td>sun</td>
</tr>
<tr>
<td>Seattle</td>
<td>Jan 7</td>
<td>7.2</td>
<td>0.0</td>
<td>2.3</td>
<td>rain</td>
</tr>
<tr>
<td>New York</td>
<td>Jan 7</td>
<td>16.1</td>
<td>0.0</td>
<td>4.7</td>
<td>sun</td>
</tr>
<tr>
<td>Seattle</td>
<td>Jan 8</td>
<td>10.0</td>
<td>0.0</td>
<td>2.8</td>
<td>rain</td>
</tr>
</tbody>
</table>

Building Blocks of Visualization

**Data**
Input data to visualize
weather.csv

**Transforms**
Filter, aggregation, binning, etc
aggregate temperature,
group by month of date and city

<table>
<thead>
<tr>
<th>City</th>
<th>Month</th>
<th>MEAN(Temperature)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>January</td>
<td>8.1</td>
</tr>
<tr>
<td>New York</td>
<td>January</td>
<td>9.1</td>
</tr>
<tr>
<td>Seattle</td>
<td>February</td>
<td>8.3</td>
</tr>
<tr>
<td>New York</td>
<td>February</td>
<td>4.4</td>
</tr>
<tr>
<td>Seattle</td>
<td>March</td>
<td>9.8</td>
</tr>
<tr>
<td>New York</td>
<td>March</td>
<td>4.4</td>
</tr>
<tr>
<td>Seattle</td>
<td>April</td>
<td>12.5</td>
</tr>
<tr>
<td>New York</td>
<td>April</td>
<td>9.4</td>
</tr>
<tr>
<td>Seattle</td>
<td>May</td>
<td>19.3</td>
</tr>
<tr>
<td>New York</td>
<td>May</td>
<td>15.9</td>
</tr>
<tr>
<td>Seattle</td>
<td>June</td>
<td>22.7</td>
</tr>
<tr>
<td>New York</td>
<td>June</td>
<td>21.9</td>
</tr>
</tbody>
</table>

Visualizations Encode Data as Visual Properties

<table>
<thead>
<tr>
<th>City</th>
<th>Month</th>
<th>MEAN(Temperature)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>January</td>
<td>8.1</td>
</tr>
<tr>
<td>New York</td>
<td>January</td>
<td>9.1</td>
</tr>
<tr>
<td>Seattle</td>
<td>February</td>
<td>8.3</td>
</tr>
<tr>
<td>New York</td>
<td>February</td>
<td>4.4</td>
</tr>
<tr>
<td>Seattle</td>
<td>March</td>
<td>9.8</td>
</tr>
<tr>
<td>New York</td>
<td>March</td>
<td>4.4</td>
</tr>
<tr>
<td>Seattle</td>
<td>April</td>
<td>12.5</td>
</tr>
<tr>
<td>New York</td>
<td>April</td>
<td>9.4</td>
</tr>
<tr>
<td>Seattle</td>
<td>May</td>
<td>19.3</td>
</tr>
<tr>
<td>New York</td>
<td>May</td>
<td>15.9</td>
</tr>
<tr>
<td>Seattle</td>
<td>June</td>
<td>22.7</td>
</tr>
</tbody>
</table>
Visualizations Encode Data as Visual Properties

- y-position
- x-position
- color
- line marks

<table>
<thead>
<tr>
<th>City</th>
<th>Month</th>
<th>MEAN(Temperature)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>January</td>
<td>8.1</td>
</tr>
<tr>
<td>New York</td>
<td>January</td>
<td>9.1</td>
</tr>
<tr>
<td>Seattle</td>
<td>February</td>
<td>8.3</td>
</tr>
<tr>
<td>New York</td>
<td>February</td>
<td>4.4</td>
</tr>
<tr>
<td>Seattle</td>
<td>March</td>
<td>9.8</td>
</tr>
<tr>
<td>New York</td>
<td>March</td>
<td>4.4</td>
</tr>
<tr>
<td>Seattle</td>
<td>April</td>
<td>12.5</td>
</tr>
<tr>
<td>New York</td>
<td>April</td>
<td>9.4</td>
</tr>
<tr>
<td>Seattle</td>
<td>May</td>
<td>19.3</td>
</tr>
<tr>
<td>New York</td>
<td>May</td>
<td>15.9</td>
</tr>
<tr>
<td>Seattle</td>
<td>June</td>
<td>22.7</td>
</tr>
</tbody>
</table>
Building Blocks of Visualization

Data
- Input data to visualize
  - weather.csv

Transforms
- Filter, aggregation, binning, etc
  - aggregate temperature,
  - group by month of date and city

Scales
- Map data values to visual values
  - color: City → ["orange", "blue"]
  - x: Month → x-coordinate
  - y: Temperature → y-coordinate

<table>
<thead>
<tr>
<th>City</th>
<th>Month</th>
<th>MEAN(Temperature)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seattle</td>
<td>January</td>
<td>8.1</td>
</tr>
<tr>
<td>New York</td>
<td>January</td>
<td>9.1</td>
</tr>
<tr>
<td>Seattle</td>
<td>February</td>
<td>8.3</td>
</tr>
<tr>
<td>New York</td>
<td>February</td>
<td>4.4</td>
</tr>
<tr>
<td>Seattle</td>
<td>March</td>
<td>9.8</td>
</tr>
<tr>
<td>New York</td>
<td>March</td>
<td>4.4</td>
</tr>
<tr>
<td>Seattle</td>
<td>April</td>
<td>12.5</td>
</tr>
<tr>
<td>New York</td>
<td>April</td>
<td>9.4</td>
</tr>
<tr>
<td>Seattle</td>
<td>May</td>
<td>19.3</td>
</tr>
<tr>
<td>New York</td>
<td>May</td>
<td>15.9</td>
</tr>
<tr>
<td>Seattle</td>
<td>June</td>
<td>22.7</td>
</tr>
</tbody>
</table>
Building Blocks of Visualization

Data
Input data to visualize
weather.csv

Transforms
Filter, aggregation, binning, etc
aggregate temperature,
group by month of date and city

Scales
Map data values to visual values
color: City → ["orange", "blue"]
x: Month → x-coordinate
y: Temperature → y-coordinate

Guides
Axes & legends to visualize scales

Marks
Data-representative graphics

Building Blocks of Visualization

Data
Input data to visualize
weather.csv

Transforms
Filter, aggregation, binning, etc
aggregate temperature,
group by month of date and city

Scales
Map data values to visual values
color: City → ["orange", "blue"]
x: Month → x-coordinate
y: Temperature → y-coordinate

Guides
Axes & legends to visualize scales

Marks
Data-representative graphics
Vega-Lite
vega.github.io/vega-lite
Vega-Lite Encodings

data:
  url: weather.csv
Vega-Lite Encodings

data:
  url: weather.csv
mark: line
Vega-Lite Encodings

data:
    url: weather.csv
mark: line
encoding:
Vega-Lite Encodings

data:
  url: weather.csv
mark: line
encoding:
  x:
    field: date, type: temporal
timeUnit: monthdate
Vega-Lite Encodings

data:
  url: weather.csv
mark: line
encoding:
  x:
    field: date, type: temporal
timeUnit: monthdate
  y:
    field: temperature, type: quantitative
aggregate: mean
Vega-Lite Encodings

data:
  url: weather.csv
mark: line
encoding:
  x:
    field: date, type: temporal
timeUnit: monthdate
  y:
    field: temperature, type: quantitative
    aggregate: mean
color:
  field: city, 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.
Vega-Lite is an Expressive Language for Statistical Interactive Multi-View Graphics.
Vega-Lite is an Expressive Language for Statistical **Interactive** Multi-View 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 visualization 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 transformations including stacking and faceting.

Moreover, Vega-Lite specifications can be composed into layered and multi-view displays, and made interactive with selections.

Read our interactive documentation or browse our tutorials to learn more about the grammar of interactive graphics.

Examples

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

vega.github.io/vega-lite
Vega-Lite as a File Format

data:
  url: weather.csv
mark: line
encoding:
  x:
    field: date,
    type: temporal
timeUnit: monthdate
y:
  field: temperature
  type: quantitative
aggregate: mean
color:
  field: city
  type: nominal
Vega-Lite as a File Format

```json
{
  "data": {
    "url": "weather.csv"
  },
  "mark": "line",
  "encoding": {
    "x": {
      "field": "date",
      "type": "temporal",
      "timeUnit": "monthdate"
    },
    "y": {
      "field": "temperature",
      "type": "quantitative",
      "aggregate": "mean"
    },
    "color": {
      "field": "city",
      "type": "nominal"
    }
  }
}
```

Convenient JSON syntax
Native to the web and easy to generate

Started an ecosystem of tools
UI tools

**Voyager.** Wongsuphasawat, Moritz et al. Infovis 2015. *Invited to SIGGRAPH*
**Voyager 2.** Wongsuphasawat et al. CHI 2017.
More: https://vega.github.io/vega-lite/applications.html
Vega-Lite as a File Format

```json
{
    "data": {
        "url": "weather.csv"
    },
    "mark": "line",
    "encoding": {
        "x": {
            "field": "date",
            "type": "temporal",
            "timeUnit": "monthdate"
        },
        "y": {
            "field": "temperature",
            "type": "quantitative",
            "aggregate": "mean"
        },
        "color": {
            "field": "city",
            "type": "nominal"
        }
    }
}
```

Convenient JSON syntax
Native to the web and easy to generate

Started an ecosystem of tools
UI tools and bindings for programming languages

**Voyager**. Wongsuphasawat, Moritz et al. *Infovis* 2015. *Invited to SIGGRAPH*


More: https://vega.github.io/vega-lite/applications.html
Vega-Lite as a File Format

```json
{
  "data": {
    "url": "weather.csv"
  },
  "mark": "line",
  "encoding": {
    "x": {
      "field": "date",
      "type": "temporal",
      "timeUnit": "monthdate"
    },
    "y": {
      "field": "temperature",
      "type": "quantitative",
      "aggregate": "mean"
    },
    "color": {
      "field": "city",
      "type": "nominal"
    }
  }
}
```

Altair in Python

```python
import altair as alt

weather = alt.Data(url='weather.csv')

alt.Chart(weather)
.mark_line()
.encode(
    x=alt.X('date:T', timeUnit='monthdate'),
    y=alt.Y('temp_max:Q', aggregate='mean'),
    color='city:N'
)

go.gl/6ihGo2
```

Similar bindings exist for R, Julia, Elm, Scala, ...
Color andOpacity

The color encoding channel sets a mark’s color. The style of color encoding is highly dependent on the data type: nominal data will default to a multi-hued qualitative color scheme, whereas ordinal and quantitative data will use perceptually ordered color gradients.

Here, we encode the cluster field using the color channel and a nominal (N) data type, resulting in a distinct hue for each cluster value. Can you start to guess what the cluster field might indicate?

```
[13]: alt.Chart(data2000).mark_point().encode(
       alt.X('fertility:Q'),
       alt.Y('life_expect:Q'),
       alt.Size('pop0'), scale=alt.Scale(range=[0,1000]),
       alt.Color('cluster:N')
    )
```
Available as default plotting library in JupyterLab.

---

Data visualization tools drive interactivity and reproducibility in online publishing
Vega-Lite as a File Format

```json
{
  "data": {
    "url": "weather.csv"
  },
  "mark": "line",
  "encoding": {
    "x": {
      "field": "date",
      "type": "temporal",
      "timeUnit": "monthdate"
    },
    "y": {
      "field": "temperature",
      "type": "quantitative",
      "aggregate": "mean"
    },
    "color": {
      "field": "city",
      "type": "nominal"
    }
  }
}
```

```python
import altair as alt

weather = alt.Data(url='weather.csv')

alt
  .Chart(weather)
  .mark_line()
  .encode(
    x=alt.X('date:T', timeUnit='monthdate'),
    y=alt.Y('temp_max:Q', aggregate='mean'),
    color='city:N')
```

goo.gl/6ihGo2
Available as default plotting library in JupyterLab.
Available as default plotting library in JupyterLab.

200K downloads/month on NPM. 1M hits/month on CDN.
Available as default plotting library in JupyterLab.

200K downloads/month on NPM. 1M hits/month on CDN.

Research Projects at UW, Stanford, MIT, Georgia Tech, Maryland, City London, Northwestern...
Available as **default plotting library** in JupyterLab.

Used to teach visualization at UW, MIT, Michigan, Georgia Tech, ...

**Featured by Nature.**

Used by the LA Times

Research Projects at UW, Stanford, MIT, Georgia Tech, Maryland, City London, Northwestern...

Data visualization tools drive interactivity and

**Used by Apple, Google, Uber, Netflix, Intel, ...**

Wikipedia integration in Progress.
"Vega-Lite is perhaps the best existing candidate for a principled lingua franca of data visualization"

Brian Granger
Lead developer of Project Jupyter
Vega-Lite

*IEEE Infovis 2016. Best Paper Award*

Easy to use for people
Concise specifications
Reusable designs
Facilitates rapid authoring for fast iterations

Designed for programmatic generation
Declarative: specification decoupled from execution
High-level, domain-specific abstractions
Composable building blocks
Draco's goal:

Provide a formal model of design knowledge for automated reasoning in tools that provide guidance.
Draco's goal:

Provide a formal model of design knowledge for automated reasoning in tools that provide guidance.

Enable computational reasoning.
Automated design and critique.
Improve our ability to create perceptually effective charts.

Foster research. Sharing of concrete, testable models of design knowledge.
Draco
IEEE Infovis 2018. Best Paper Award

Formal model of visual encodings as sets of facts.

Design knowledge as constraints.
Draco

IEEE Infovis 2018. **Best Paper Award**

- Formal model of visual encodings as sets of facts.
- Design knowledge as constraints.
Draco: Encodings as Logical Facts

data:
  url: weather.csv
mark: line
encoding:
  x:
    timeUnit: month
    field: date
    type: temporal
  y:
    aggregate: mean
    field: temperature
    type: quantitative
  color:
    field: city
    type: nominal
Draco: Encodings as Logical Facts

Vega-Lite

data:
  url: weather.csv
mark: line
encoding:
x:
  timeUnit: month
  field: date
  type: temporal
y:
  aggregate: mean
  field: temperature
  type: quantitative

color:
  field: city
  type: nominal

Draco

data("weather.csv").
mark(line).
encoding(e0).
channel(e0,x).
timeUnit(e0,month).
field(e0,date).
type(e0,t).

encoding(e1).
channel(e1,y).
aggregate(e1,mean).
field(e1,temperature).
type(e1,q).

encoding(e2).
channel(e2,color).
field(e2,city).
type(e2,n).
Draco: Encodings as Logical Facts

data("weather.csv").
mark(line).
encoding(e0).
channel(e0,x).
timeUnit(e0,month).
field(e0,date).
type(e0,t).

encoding(e1).
channel(e1,y).
aggregate(e1,mean).
field(e1,temperature).
type(e1,q).

encoding(e2).
channel(e2,color).
field(e2,city).
type(e2,n).
What are the properties of the fields?

Weather

Data

Select Data Fields

City Date Temperature

Transform Data

MEAN(Temperature) By Month of Date, City

Design Encoding

Visualization
What are the properties of the fields?

Weather

Select Data Fields

City
Date
Temperature

Transform Data

MEAN(Temperature)
By Month of Date, City

Design Encoding

Data

Data Fields

Transformed Data

Visualization

65
Visualization Always has a Context

What are the properties of the fields?

Seattle is gray and cold all the time, right?

Weather

Select Data Fields

City Date Temperature

Transform Data

Mean(Temperature) by Month of Date, City

Design Encoding

Visualization

Data

Data Fields

Transformed Data

Visualization
Draco: Encodings as Logical Facts

Draco extends Vega-Lite to capture the context of user task and data properties.

data("weather.csv").
mark(line).
encoding(e0).
channel(e0,x).
timeUnit(e0,month).
field(e0,date).
type(e0,t).

encoding(e1).
channel(e1,y).
aggregate(e1,mean).
field(e1,temperature).
type(e1,q).

encoding(e2).
channel(e2,color).
field(e2,city).
type(e2,n).

task(value).
field(month).
dataType(month, date).
cardinality(month, 48).

field(temperature).
dataType(temperature, float).
cardinality(temperature, 3786).
entropy(temperature, 11).
extent(temperature, -13, 38).

field(city).
dataType(city, string).
cardinality(city, 6).
entropy(city, 1).
...

...
How do we make the computer reason for us?
Draco

IEEE Infovis 2018. Best Paper Award

Formal model of visual encodings as sets of facts.

Design knowledge as constraints.
Draco
IEEE Infovis 2018. Best Paper Award

Formal model of visual encodings as sets of facts.

Design knowledge as constraints.

- Attribute domains
- Integrity
- Preferences
Describe the domain of attributes.

e.g. mark type

“The mark of a chart should be one of bar, line, area or point.”

\[ \text{marktype}(\text{bar};\text{line};\text{area};\text{point}). \{ \text{mark}(\text{M}) : \text{marktype}(\text{M}) \} = 1. \]
Draco: Design Knowledge as Constraints

Describe the domain of attributes.

e.g. mark type, encoding type, aggregate, channels, binning, data types, tasks...
Draco: Design Knowledge as Constraints

Constrain to valid visualizations that satisfy rules of visual design.

Hard constraints.

e.g. “Only continuous fields can be aggregated.”

“A bar mark needs at least one continuous x or y.”

“The shape channel requires point marks.”

... 

total of ~70 hard constraints
Draco: Design Knowledge as Constraints

Describe *preferences* within the space of valid encodings as soft constraints.

“Prefer specifications with fewer encodings.”

“Prefer not to use aggregation.”

“Prevent overlapping marks.”

total ~150 soft constraints
Draco: Design Knowledge as Constraints

Describe **preferences** within the space of valid encodings as soft constraints.

Each violation incurs a **cost**.

1. “Prefer specifications with fewer encodings.”
2. “Prefer not to use aggregation.”
3. “Prevent overlapping marks.”

**total ~150 soft constraints**
Visualization Recommendation with Draco

💡 Formulate visualization recommendation as finding optimal completions.

"I want a visualization of the temperature."
Visualization Recommendation with Draco

💡 Formulate visualization recommendation as finding optimal completions.

"I want a visualization of the temperature."

data("weather.csv").
encoding(e0).
field(e0, temperature).

+ constraint solver

attributedomains(e0).
mark(tick).
encoding(e0).
field(e0, temperature).
channel(e0, x).
type(e0, q).
Visualization Recommendation with Draco

💡 Formulate visualization recommendation as finding optimal completions.

"I want a visualization of the temperature."

```
data("weather.csv").
encoding(e0).
field(e0,temperature).
```

```
data("weather.csv").
mark(tick).
encoding(e0).
field(e0,temperature).
channel(e0,x).
type(e0,q).
```

Attribute domains
Integrity
Preferences
constraint solver

How much data?

Temperature

0 10 20 30
Visualization Recommendation with Draco

"I want a visualization of the binned temperature."

data("weather.csv").
encoding(e0).
field(e0, temperature).
bin(e0).

Attribute domains
Integrity.
Preferences

Overlapping marks

Binned Temperature

data("weather.csv").
mark(tick).
encoding(e0).
field(e0, temperature).
channel(e0, x).
type(e0, q).
bin(e0).
"I want a visualization of the binned temperature."

"Prefer specifications with fewer encodings."
"Prefer not to use aggregation."
"Prevent overlapping marks."

\[2 + 3 < 6\]

- data("weather.csv")
- encoding(e0)
- field(e0, temperature)
- bin(e0)

- Attribute domains
- Integrity
- Preferences

- constraint solver
- data("weather.csv")
- mark(tick)
- encoding(e0)
- field(e0, temperature)
- channel(e0, x)
- type(e0, q)
- bin(e0)
"I want a visualization of the binned temperature."

- Prefer specifications with fewer encodings.
- Prefer not to use aggregation.
- Prevent overlapping marks.

\[2 + 3 < 6\]

data("weather.csv").
encoding(e0).
field(e0, temperature).
bin(e0).

Attribute domains
Integration
Preferences

Constraint solver

data("weather.csv").
mark(bar).
encoding(e0).
field(e0, temperature).
channel(e0, x).
type(e0, q).
bin(e0).
encoding(e1).
type(e1, q).
aggregate(e1, count).
“Prefer specifications with fewer encodings.”

“Prefer not to use aggregation.”

“Prevent overlapping marks.”

Where do these weights come from?

"Graduate Student Descent"

Machine Learning
Draco

IEEE Infovis 2018. Best Paper Award

Design knowledge as constraints.

- Attribute domains
- Integrity
- Preferences

Methods to learn trade-offs from experimental data.

Formal model of visual encodings as sets of facts.
Data for Learning Trade-Offs

Recommendation with Draco: **partial input → complete specification**
But: little data to learn from

(Visualization → Score) → Pairs where score is significantly different

Score: 0.5 (bad)  <  Score: 1.7 (good)

Score: 1.1 (okay)  <  Score: 2.5 (great)

Pairs are Ordinal
We can combine the results of multiple experiments with different measures.
Draco: Learn Trade-Offs from Data

Training Data

Pairs of
Ranked Visualizations
Draco: Learn Trade-Offs from Data

Training Data
Pairs of Ranked Visualizations

Features
Violations of Soft Constraints
Draco: Learn Trade-Offs from Data

Training Data
Pairs of Ranked Visualizations

Features
Violations of Soft Constraints

positive example
Feature Vector $[u_1, u_2, \ldots, u_k]$

negative example
Feature Vector $[v_1, v_2, \ldots, v_k]$

$v_j$: the number of violations of constraint $i$
Draco: Learn Trade-Offs from Data

**Training Data**
Pairs of Ranked Visualizations

**Features**
Violations of Soft Constraints

**Learning Algorithm**
Learning to Rank with Linear SVM

### Positive Example

- Feature Vector: $[u_1, u_2, \ldots, u_k]$
- $w$ is the weight vector of the soft constraints
- $\text{arg max}_w \sum_{i \in 0 \ldots k} w_i (u_i - v_i)$

### Negative Example

- Feature Vector: $[v_1, v_2, \ldots, v_k]$
- $v_i$: the number of violations of constraint $i$
Draco at the Core of Visualization Research

Theory
Describe formal models.
Systematically improve models.

Systems
Develop automated design tools.

Empirical
Inform perceptual studies.

Infovis 2018. Best Paper

Research → Practice
How do I create the next generation of visualization systems where users can rapidly create good designs regardless of the scale of their data?
How can we visualize and interact with billion+ record datasets in real-time?
How can we visualize and interact with billion+ record datasets in real-time?
How to Visualize a Billion+ Records

Data

Sampling

Binned Aggregation
How to **Visualize** a Billion+ Records

Decouple the visual complexity from the raw data through aggregation.
Formal Models of Visualization

**Vega-Lite** *Infovis 2016. Best Paper*
High-Level grammar for interactive multi-view graphics
Designed for programmatic generation

**Draco** *Infovis 2018. Best Paper*
Formal reasoning for visualization design

Scalable Visualization

**Falcon** *CHI 2019.*
Real-time linked interactions with billions of records

**Optimistic Visualization** *CHI 2017.*
Fast and reliable approximations for data exploration
How do we interact with billion+record datasets in real-time?
How do we interact with billion+record datasets in real-time?

Delays reduce engagement and lead to fewer observations.


Delays may bias analysts towards convenient data.
Falcon

uwdata.github.io/falcon
How can Falcon be real-time?
Falcon Interaction Log

- Arrival Delay in Minutes
- Departure Time
- Distance in Miles

5x speedup
Falcon Interaction Log

Arrival Delay in Minutes

Departure Time

Distance in Miles

5x speedup
Brushing is more common and people are sensitive to latencies. Prioritize brushing latency over view switching latency.
Key Idea:

User-centered prefetching and indexing to support all brushing interactions with one view. Re-compute if the user switches the view.
brushes in the precomputed view
brushes in the precomputed view

serves requests from a data cube

brushes in the precomputed view

serves requests from a data cube

**Data Cube.** Gray et al. 1997.

interacts with a new view

computes new data cubes
brushes in the precomputed view

serves requests from a data cube


interacts with a new view

computes new data cubes
brushes in the precomputed view
serves requests from a data cube
interacts with a new view
computes new data cubes
Constant data & time. Client only.

- brushes in the precomputed view
- serves requests from a data cube

- interacts with a new view
- computes new data cubes
Constant data & time. Client only.

\[ \text{brushes in the precomputed view} \]

\[ \text{serves requests from a data cube} \]

\[ \text{Data Cube. Gray et al. 1997} \]

💡 Aggregation decouples interactions from queries over the raw data.

\[ \text{interacts with a new view} \]

\[ \text{computes new data cubes} \]
Constant data & time. Client only.

灯具 in the precomputed view

serves requests from a data cube


Aggregation decouples interactions from queries over the raw data.

💡 Requires one pass over the data.

interacts with a new view

computes new data cubes

💡 View switches are rare and users are not as latency sensitive with them.
Visualization Systems that Leverage Data Cubes

Problem: The full data cube has size $\prod_{i} b_i$, where $b_i$ is the number of bins in dimension $i$.

**Nanocubes.** Lins et al. *Infovis* 2013.
Specialized hierarchical data structure for sparse cubes.
Cubes are still too large for the browser. Hours of build time.

Dense cube. Decomposed into overlapping cubes.
One cube per pairwise interactions. One brush. Brushing at bin resolution. Hours of build time.

**Falcon.** Moritz et al. *CHI* 2019.
Small cubes for single active view.
Small cubes are built on the fly. View switches require new cube.
1.7 B stars.
1.2 TB of data.
Visualizations running in my browser.
Data stored in OmniSci database.
"With Falcon it feels like I'm really interacting with my data."

Data Platform Engineer at Stitch Fix
What if data too large to even query it in a reasonable time?
Latencies reduce engagement and lead to fewer observations.

Approximation.  Accuracy $\rightarrow$ Speed

Approximate query processing (AQP)
Uncertainty estimation in statistics
Uncertainty visualization
Probabilistic programming
Approximate hardware
Approximation. Accuracy → Speed

Approximate query processing (AQP)
Uncertainty estimation in statistics
Uncertainty visualization
Probabilistic programming
Approximate hardware
Approximation. Accuracy → Speed

Approximate query processing (AQP)
Uncertainty estimation in statistics
Uncertainty visualization
Probabilistic programming
Approximate hardware
Users had to choose:
1. Trust the approximation, or
2. Wait for everything to complete.
Optimistic Visualization

Trust but Verify
What if we think of the issues with approximation as user-experience problems?
Optimistic Visualization

1. Analysts uses initial estimates.

1. Analysts uses initial estimates.
2. Precise queries run in the background.
1. Analysts use initial estimates.
2. Precise queries run in the background.

Optimistic Visualization

1. Analysts uses initial estimates.
2. Precise queries run in the background.

Analysts can use approximations and also trust them.
Pangloss Implements Optimistic Visualization
Pangloss Visualizes Uncertainty
Pangloss shows a History of Previous Charts
In Pangloss, Analysts can Confirm results

The visualization is read only because you're looking at the history. Return to the working view or make a copy of the current chart.
Evaluation

Case studies with teams at Microsoft who brought in *their own data*.

**Approximation works**

“seeing something right away at first glimpse is really great”

**Need for guarantees**

“[with a competitor] I was willing to wait 70-80 seconds. It wasn’t ideally interactive, but it meant I was looking at all the data.”

**Optimism works**

“I was thinking what to do next— and I saw that it had loaded, so I went back and checked it . . . [the passive update is] very nice for not interrupting your workflow.”
Formal Models of Visualization

Vega-Lite *Infovis 2016. Best Paper*
High-Level grammar for interactive multi-view graphics
Designed for programmatic generation

Scalable Visualization

Falcon *CHI 2019.*
Real-time linked interactions with billions of records

Optimistic Visualization *CHI 2017.*
Fast and reliable approximations for data exploration

Draco *Infovis 2018. Best Paper*
Formal reasoning for visualization design
My Mission:

Develop **tools for data analysis and communication** that richly integrate the strengths of both **people** and **machines**.
Vega-Lite
High-level visualization language

Authoring by people

Programmatic generation

New tools like Altair
Draco
Model of visualization design

Draco formalizes visualization design.

The model can inform our understanding of visualization.
Falcon + Pangloss
Scalable visualization systems

Understanding how 👤 interact with visualizations enabled new 🤖 optimizations.
Challenge for the Future:

Reasoning about user task and system concerns are largely not available in end-user tools.
Challenge for the Future:

Reasoning about user task and system concerns are largely not available in end-user tools. We need end-to-end integration into analysis workflows.
Vega-Lite's Research Frontier

Reduce redundant computation.
Vega-Lite's Research Frontier

Reduce redundant computation.
Push expensive computation into scalable backends.  (UW, Google, OmniSci)
Draco's Research Frontier

UI Tools to browse, update, and compare Draco knowledge bases.
Evaluate impact of new perceptual models
(UW, Apple)

uwdata.github.io/draco-tuner
Draco's Research Frontier

**UI Tools** to browse, update, and compare Draco knowledge bases.

Integrate Draco into tools (e.g. Altair) to collect feedback.

(UW)
Draco's Research Frontier

**UI Tools** to browse, update, and compare Draco knowledge bases.

Integrate Draco into tools (e.g. Altair) to collect feedback.

**Domain-specific models** for multi-view graphics, interactions, big data, uncertainty visualization, education...

(UW, MIT, Northwestern)
Draco's Research Frontier

**UI Tools** to browse, update, and compare Draco knowledge bases.

Integrate Draco into tools (e.g. Altair) to collect feedback.

**Domain-specific models** for multi-view graphics, interactions, big data, uncertainty visualization, education...

**Modelling Tasks**
(NYU, Northwestern)
Runtime Engine for Visualization

Interactive analysis regardless of scale.
Automatically apply Falcon's optimizations. Combined with approximation.

💡 and 🤖 aware optimizations.

e.g. Suggest aggregation for large data. Approximation for very large data.

e.g. A mobile user has different constraints compared to a desktop user.
Visualization

People

Systems
Describe the analysis process
Separate specification from execution

Reason about analysis
End-to-end optimizations
Improved understanding

Data Analysis

Visualization

Data Cleaning
Machine Learning
Uncertainty
Provenance
Reproducibility
Sharing

Uncertainty

People
Systems
Vega-Lite
High-level grammar for interactive multi-view graphics.
Designed for people and systems.
*Vega-Lite. Infovis 2016. Best Paper*

Falcon
Real-time interactions for big data.
Leverages holistic system optimization.
*Falcon. CHI 2019.*

Draco
Formalized design knowledge.
Enables holistic reasoning and optimization.
*Draco. Infovis 2018. Best Paper*

Optimistic Visualization
Provide guarantees for approximations.
Treats DB problem as a UX problem.
*Trust but Verify. CHI 2017.*

@domoritz
www.domoritz.de