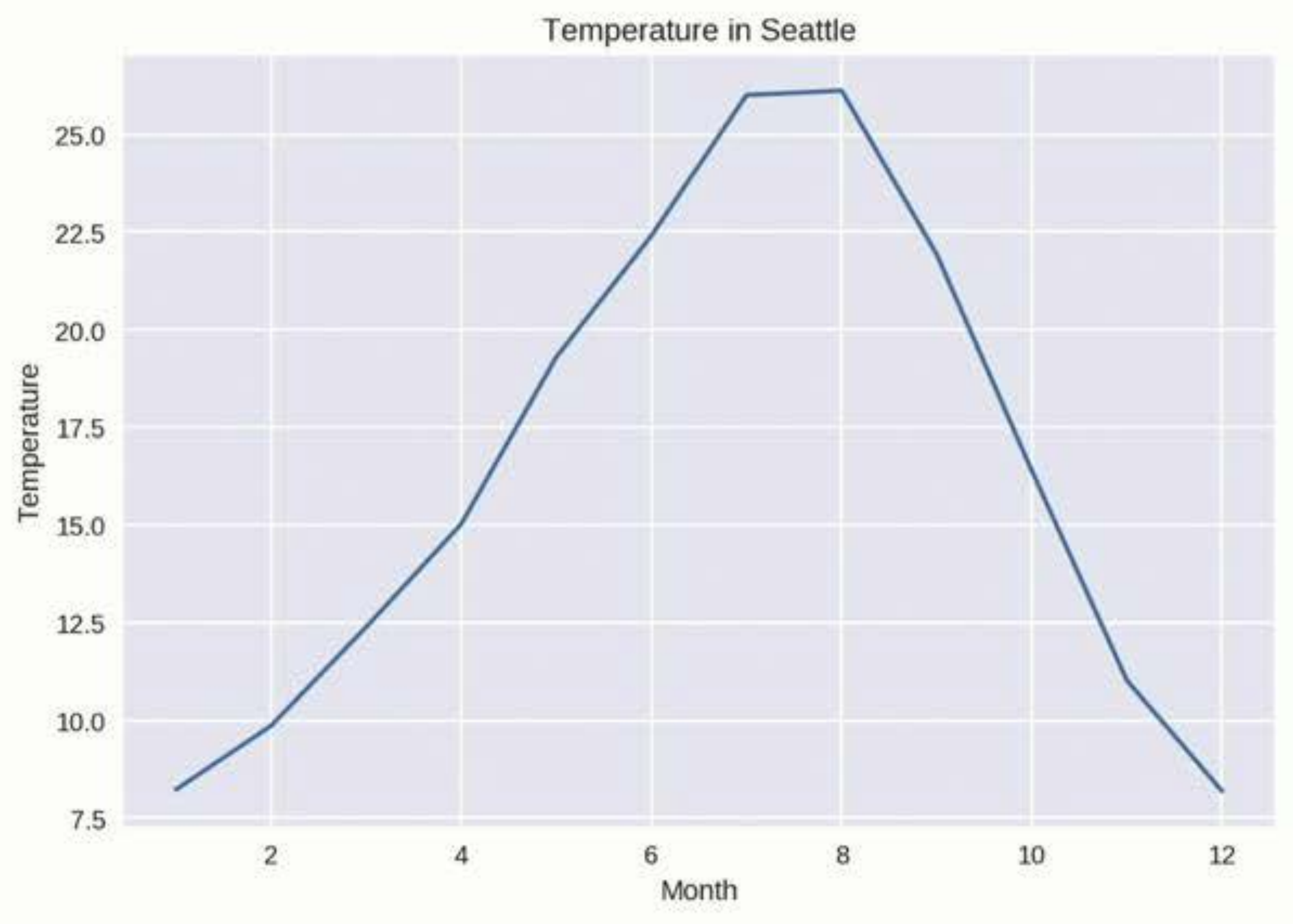
Visualization for People + Systems

Dominik Moritz @domoritz
University of Washington

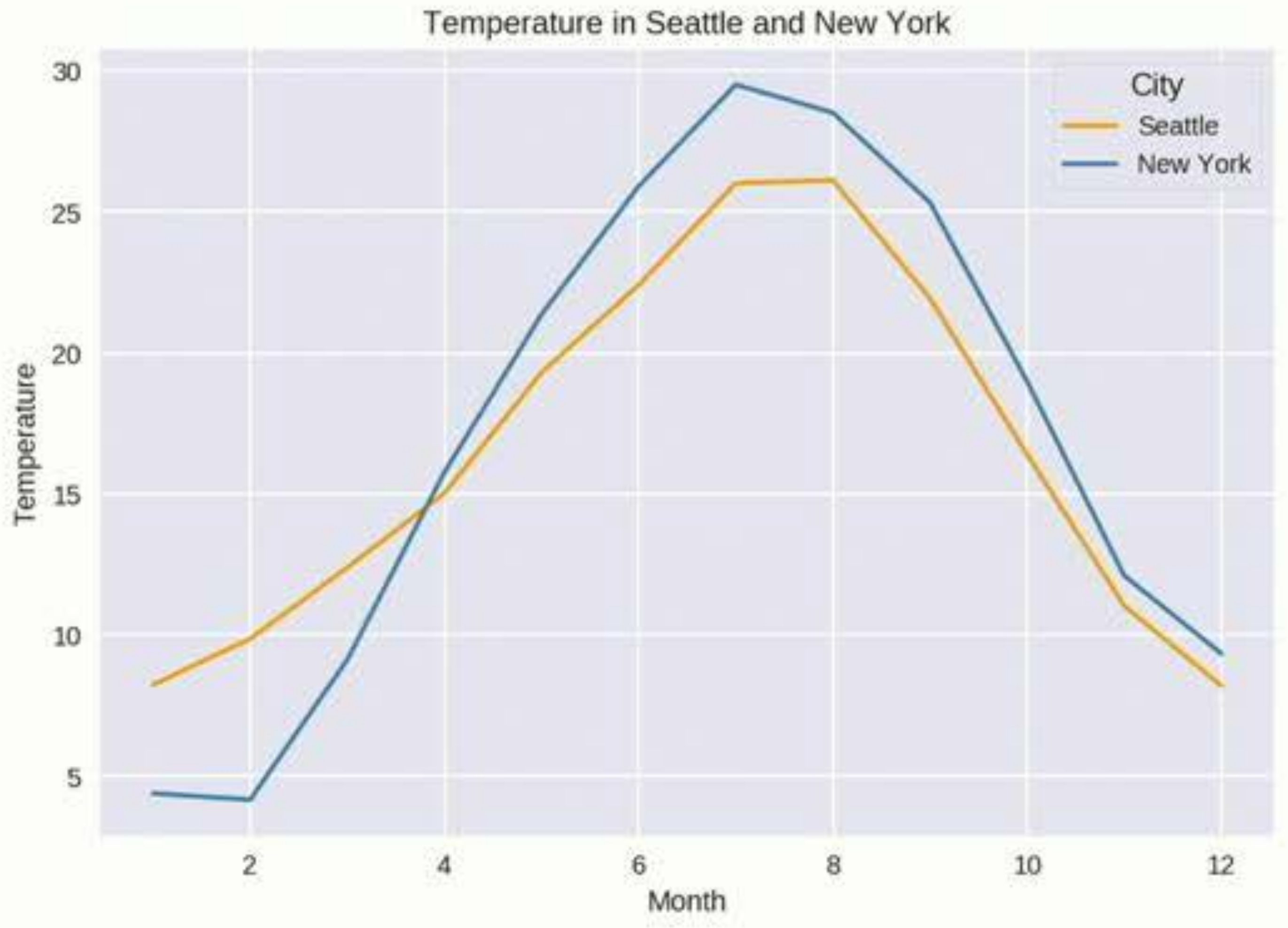
Visualize Weather Data for Seattle

```
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()
```



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

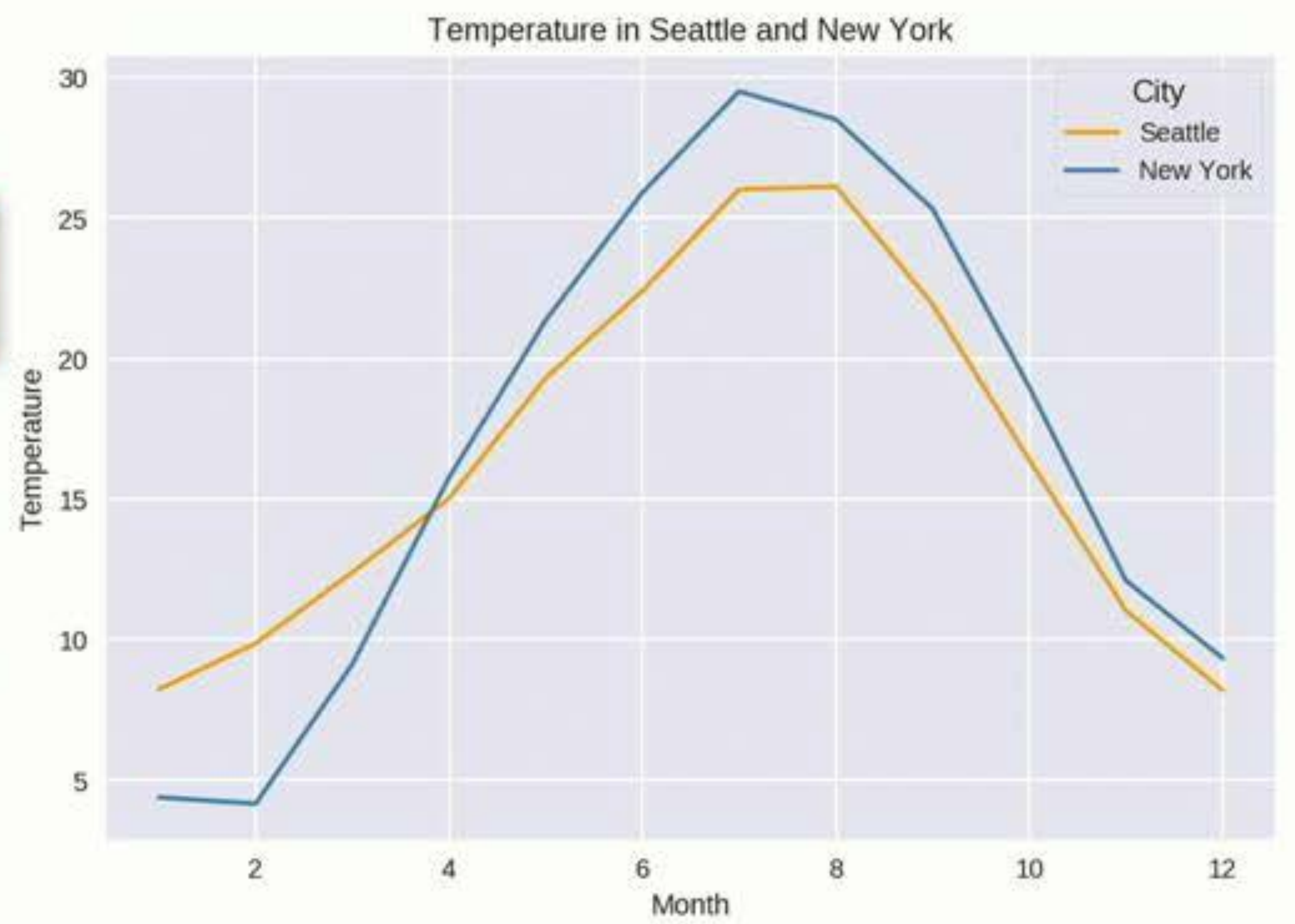
```
import matplotlib.pyplot as plt
import pandas as pd

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

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

# 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
ax.legend(frameon=True, title='City')
ax.set_ylabel('Temperature')
ax.set_xlabel('Month')

fig.show()
```

Group by location

Create color map

Draw a line for each city

Don't forget the legend!



Visualize Weather Data for Seattle and New York

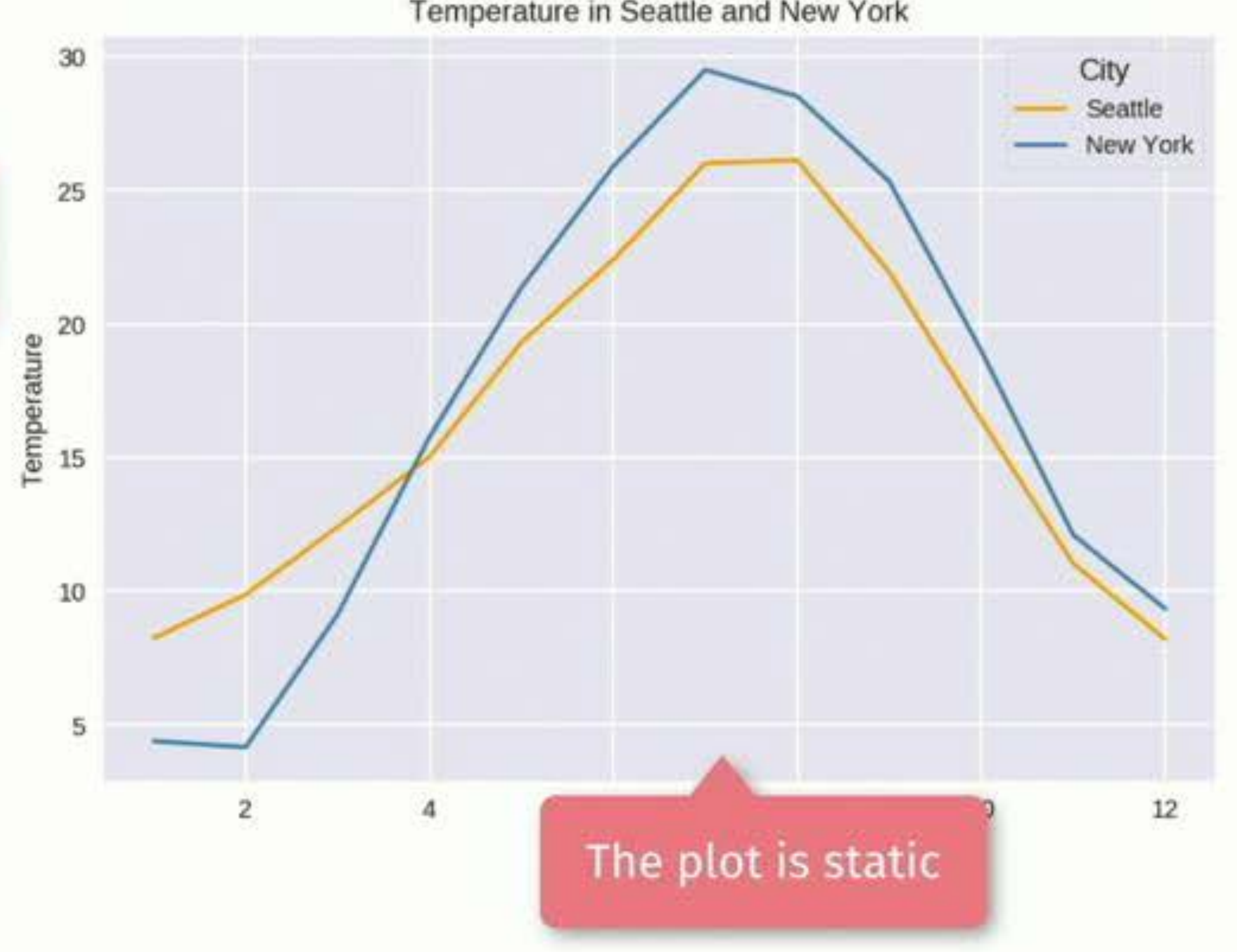
```
import matplotlib.pyplot as plt
import pandas as pd

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()
```

Group by location

Create color map

Draw a line for each city

Don't forget the legend!



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()
```

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

Group by location

Create color map

Draw a line for each city

Don't forget the legend!

Temperature in Seattle and New York

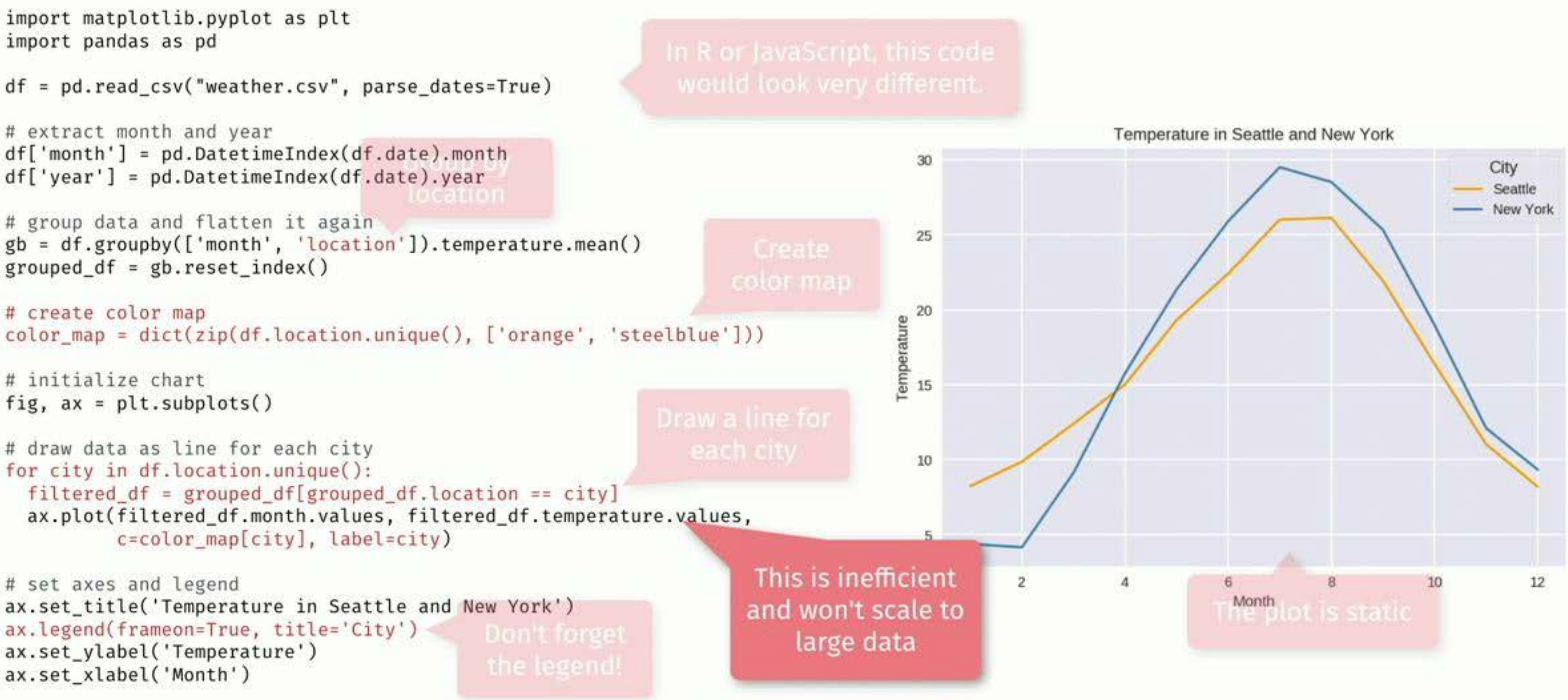
City
Seattle
New York

Temperature
30
25
20
15
10
5

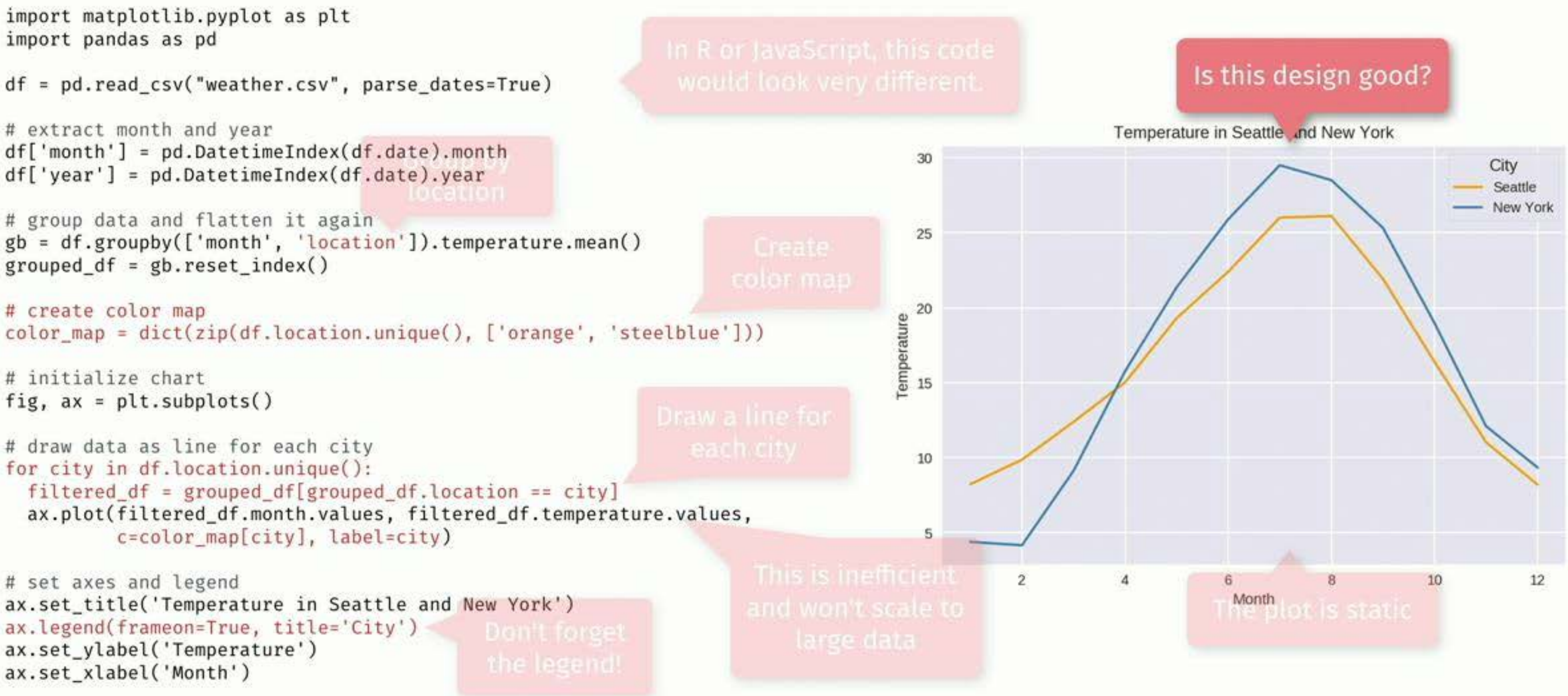
Month
2
4
6
8
10
12

The plot is static

Visualize Weather Data for Seattle and New York



Visualize Weather Data for Seattle and New York



For any good design...

…there are many poor designs.

Missing legend

…there are many poor designs.

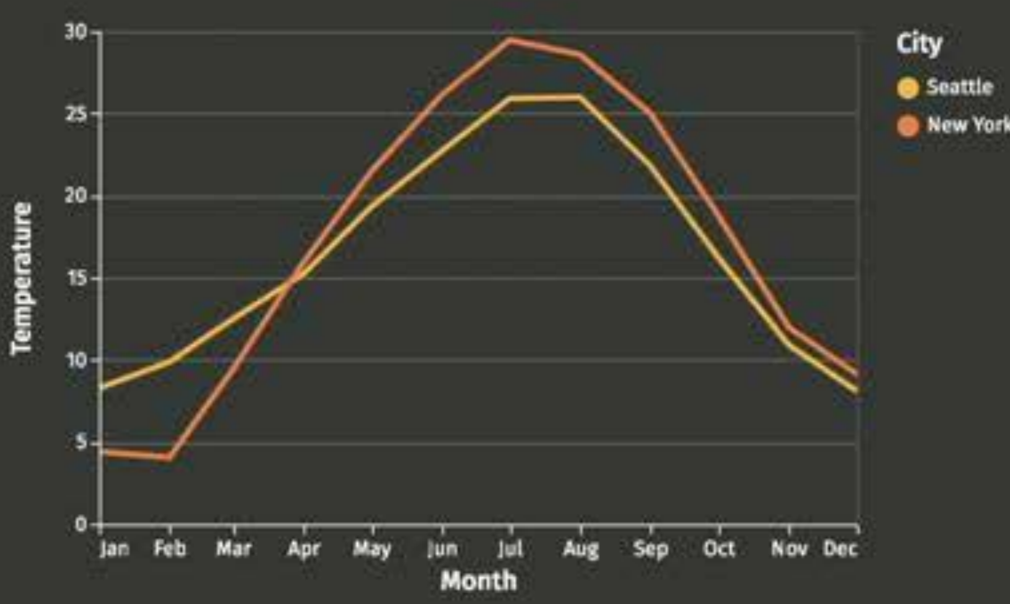
Poor color choice

…there are many poor designs.

Non-optimal mark

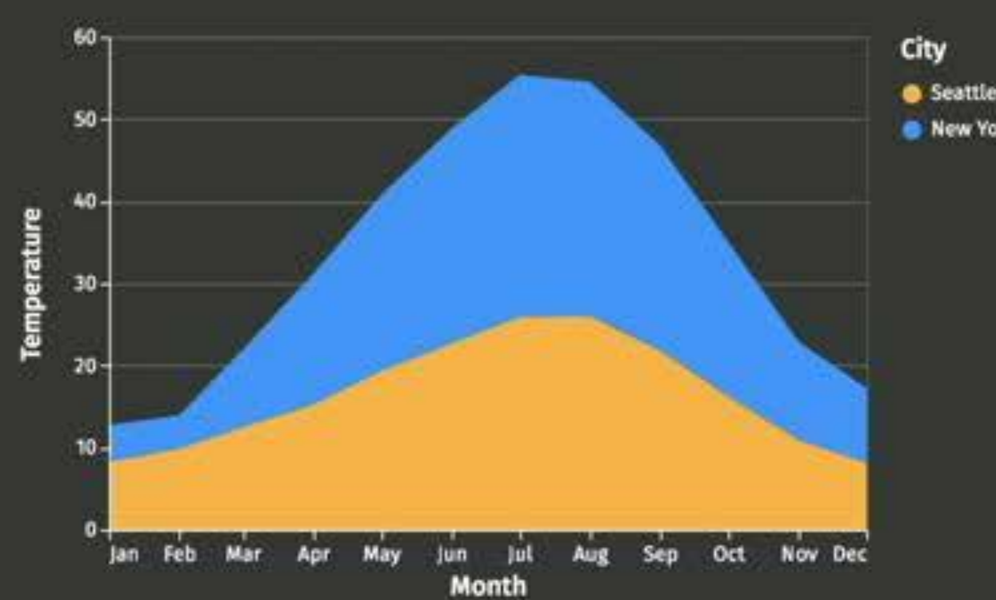
...there are many poor designs.

Missing legend



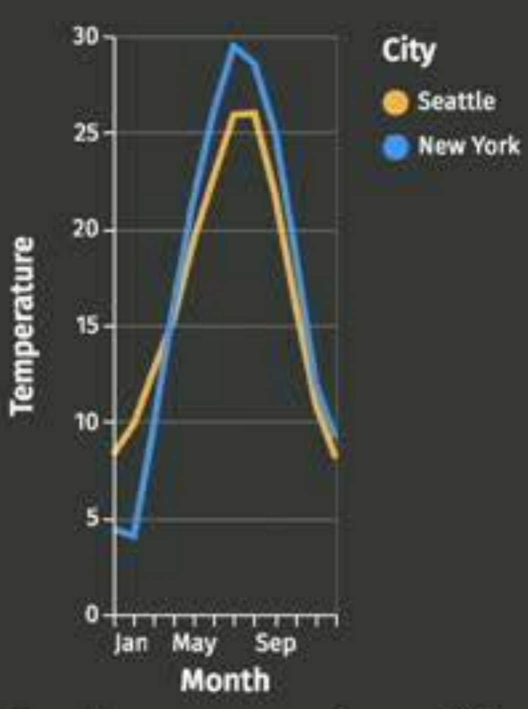
Poor color choice

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**?

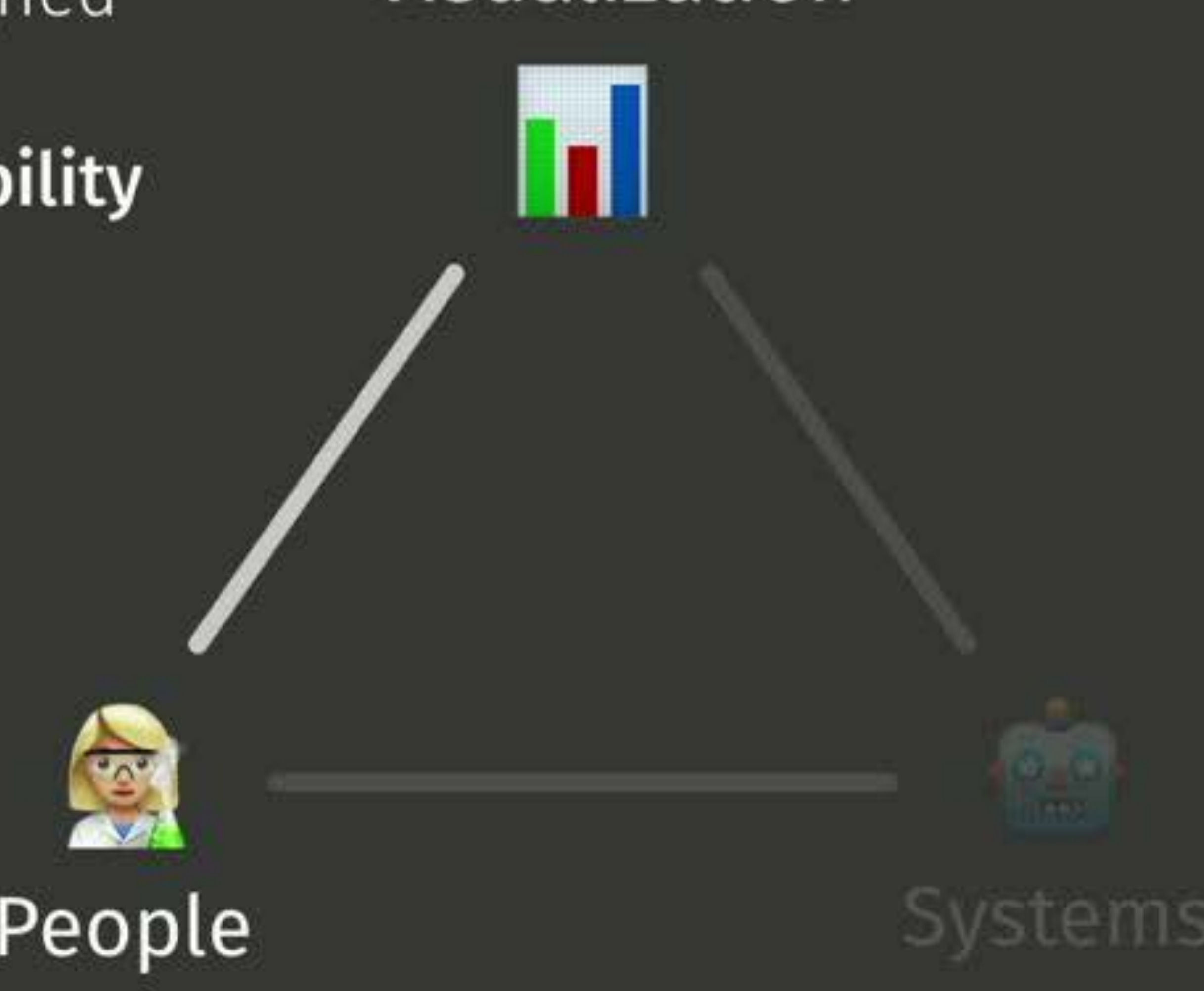
How do I create the next generation of visualization systems where users can **rapidly create good designs**?

How do I create the next generation of visualization systems where users can **rapidly create good designs** regardless of the **scale of their data**?

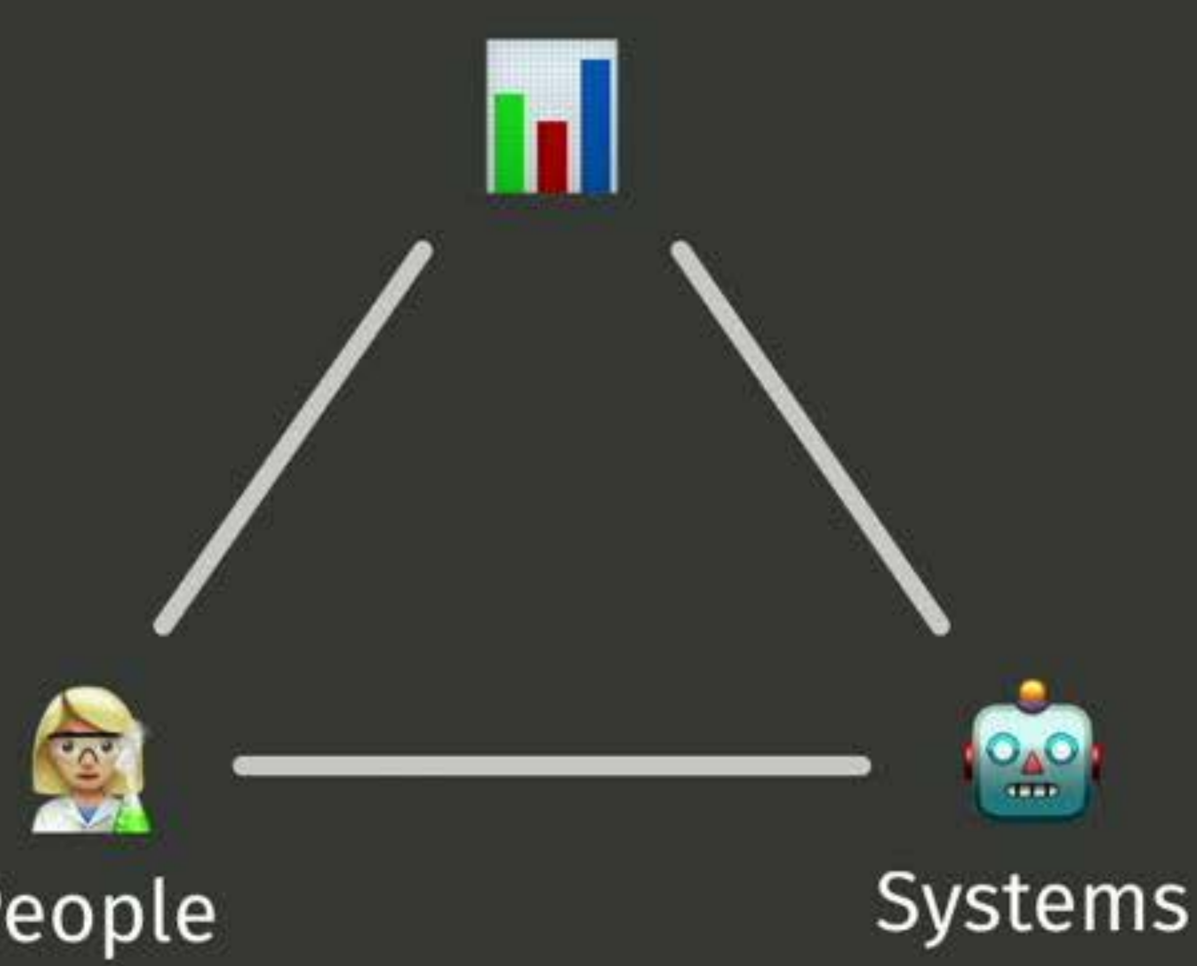
How do I create the next generation of visualization systems where users can **rapidly create good designs** regardless of the **scale of their data**?

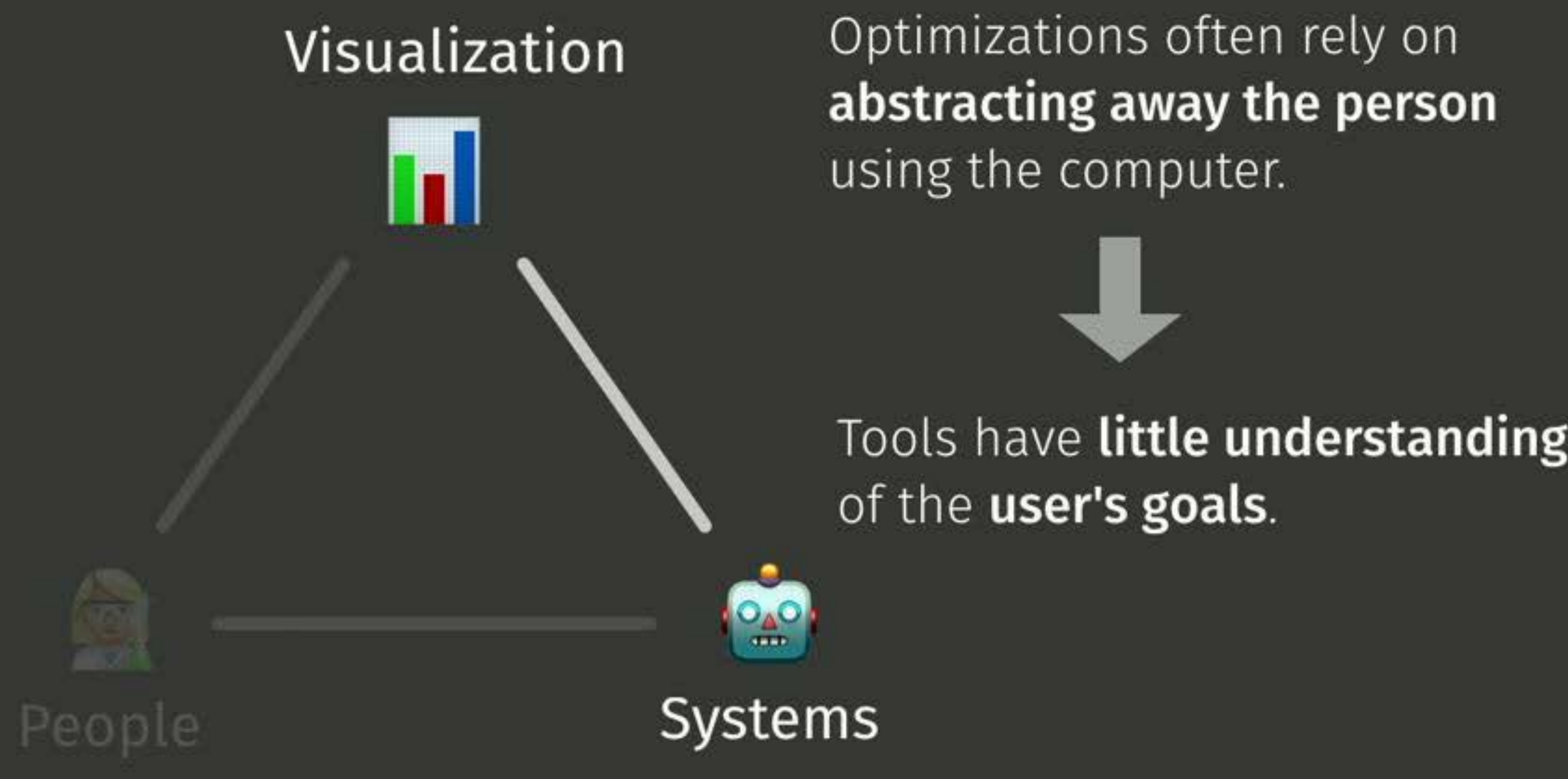
Programming tools are designed for **manual authoring**. Good design is the **responsibility of the human designer**.

Tools do not provide computational guidance.

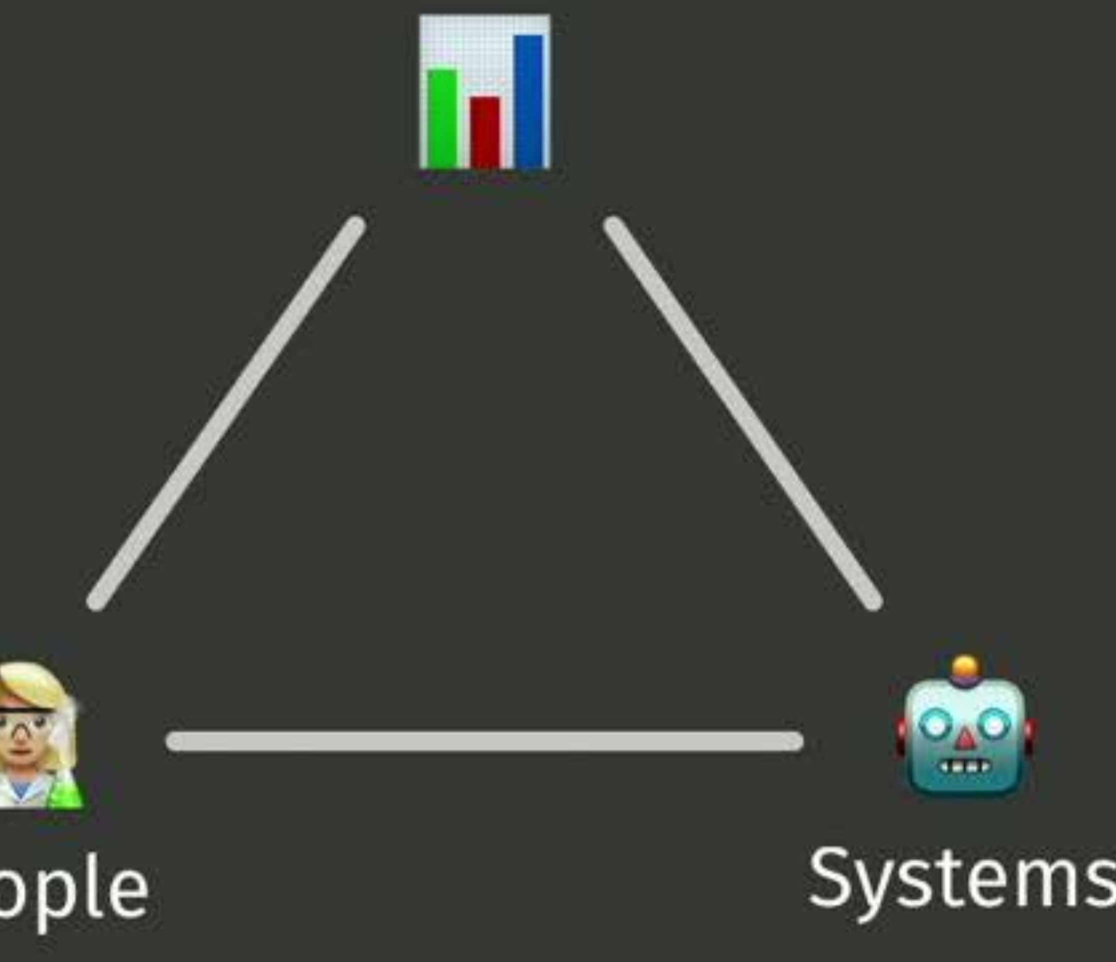


I design domain-specific languages where **people** and **systems** can meaningfully participate in the **visualization** process.



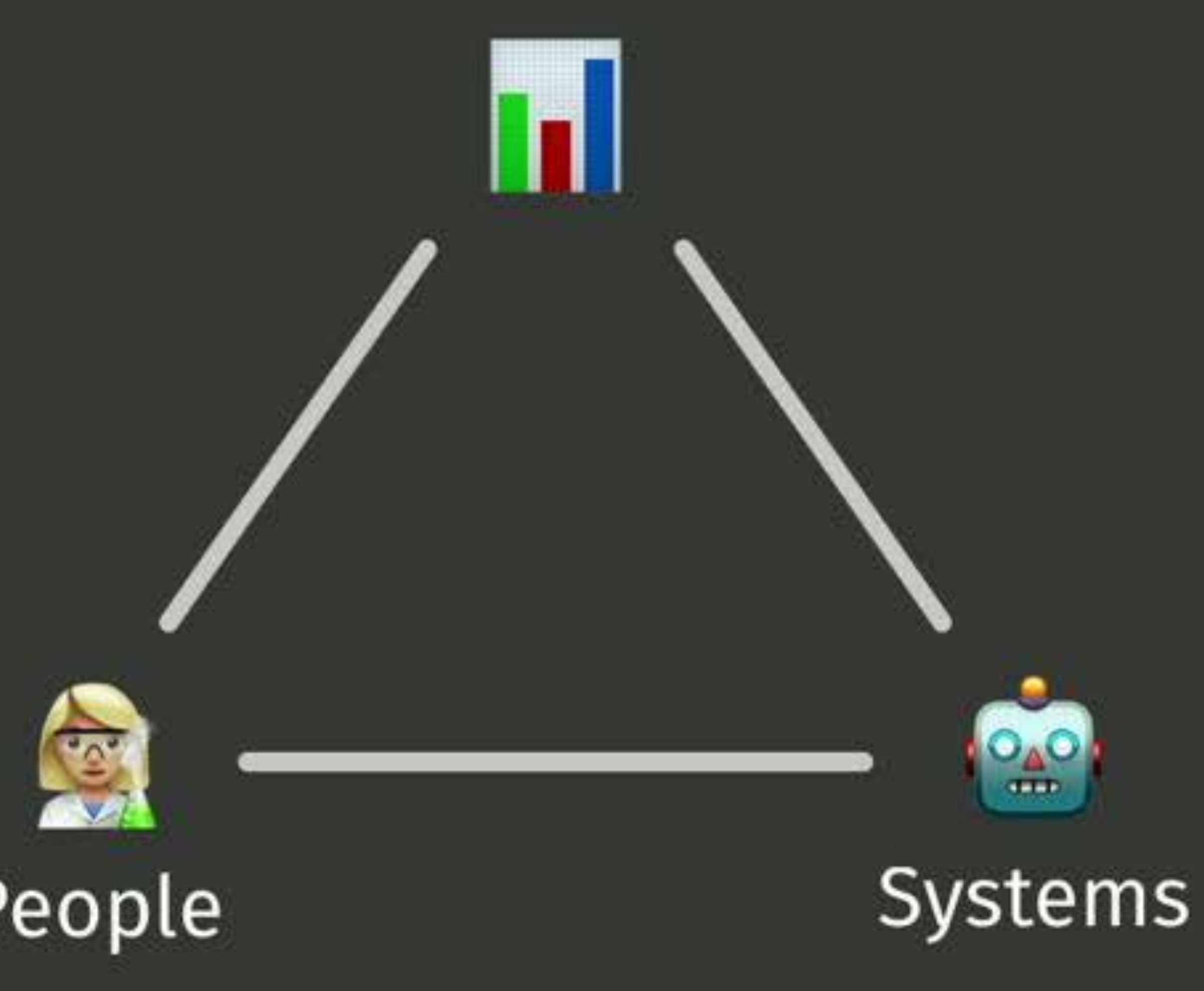

Visualization
People
Systems
Optimizations often rely on **abstracting away the person** using the computer.
Tools have **little understanding** of the **user's goals**.

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

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

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**
	Voyager 2. Wongsuphasawat et al. *CHI 2017.*
	Learning Design. Saket, Moritz et al. VisGuides *2018.*
Data science	**Altair.** VanderPlas et al. *JOSS 2018.*
	SQLShare. Jain, Moritz et al. *SIGMOD 2016.*
	Deep Learning for Text Detection. Moritz. *JOSS 2017.*
	High Variety. Jain, Moritz et al. *ICDE 2016.*
	Voronoi. Schmechel, Moritz et al. *IVAPP 2014.*
Big data (visualization) systems	**Falcon.** Moritz et al. *CHI 2019.*
	Trust but Verify. Moritz et al. *CHI 2017.*
	Myria. Wang et al. *CIDR 2017.*
	Myria. Halperin et al. *SIGMOD Demo 2014.*
	Dynamic Client-Server Optimization. Moritz et al. *DSIA 2015.*
	Million Time Series. Moritz, Fisher. *arXiv 2018.*
	Advances in Visualization of Big Data. Battle, Moritz, Fisher, Heer. MIT *2019.*
Uncertainty	**VSUP.** Correll, Moritz, Heer. *CHI 2018.*
	Uncertainty for Users. Moritz Fisher. *HILDA 2017.*
	Lessons from Pangloss. Moritz Fisher. *Uncertainty 2017.*
Debugging systems	**Perfopticon.** Moritz, et al. *Eurovis 2015.*
Searching genomes	**Spacegraphcats.** Brown et al. bioRxiv 2018.

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

Representation

Guidance

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

Draco

Guidance

Automated reasoning

Formal model of design knowledge

Programmatic generation
- Declarative
- High-level

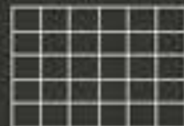
Representation

Vega-Lite

How do we make visualizations?

Visualizing Data

Weather



Data

City	Date	Precipitation	Temperature	Wind	Weather
Seattle	January 1, 2012	0.0	12.8	4.7	drizzle
New York	January 1, 2012	5.1	10.0	4.5	rain
Seattle	January 2, 2012	10.9	10.6	4.5	rain
New York	January 2, 2012	0.0	10.1	8.7	sun
Seattle	January 3, 2012	0.8	11.7	2.3	rain
New York	January 3, 2012	0.0	0.6	8.2	sun
Seattle	January 4, 2012	20.3	12.2	4.7	rain

Visualizing Data

Data **Data Fields**

City	Date	Precipitation	Temperature	Wind	Weather
Seattle	January 1, 2012	0.0	12.8	4.7	drizzle
New York	January 1, 2012	5.1	10.0	4.5	rain
Seattle	January 2, 2012	10.9	10.6	4.5	rain
New York	January 2, 2012	0.0	10.1	8.7	sun
Seattle	January 3, 2012	0.8	11.7	2.3	rain
New York	January 3, 2012	0.0	0.6	8.2	sun
Seattle	January 4, 2012	20.3	12.2	4.7	rain

Visualizing Data

Data

Data Fields

Transformed Data

City	Month	MEAN(Temperature)
Seattle	January	8.1
New York	January	9.1
Seattle	February	8.3
New York	February	4.4
Seattle	March	9.8
New York	March	4.4

Visualizing Data

Building Blocks of Visualization

Grammar of Graphics. Wilkinson. *2015.*

Data

Input data to visualize
`weather.csv`

City	Date	Temp.	Prec.	Wind	Weather
Seattle	Jan 1	12.8	0.0	4.7	drizzle
New York	Jan 1	10.0	5.1	4.5	rain
Seattle	Jan 2	10.6	10.9	4.5	rain
New York	Jan 2	10.1	0.0	8.7	sun
Seattle	Jan 3	11.7	0.8	2.3	rain
New York	Jan 3	0.6	0.0	8.2	sun
Seattle	Jan 4	12.2	20.3	4.7	rain
New York	Jan 4	-1.7	0.0	5.5	sun
Seattle	Jan 5	8.9	1.3	6.1	rain
New York	Jan 5	5.6	0.0	5.4	sun
Seattle	Jan 6	4.4	2.5	2.2	rain
New York	Jan 6	12.2	2.5	4.6	sun
Seattle	Jan 7	7.2	0.0	2.3	rain
New York	Jan 7	16.1	0.0	4.7	sun
Seattle	Jan 8	10	0.0	2.0	rain
New York	Jan 8	[illegible]	0.0	[illegible]	[illegible]

Building Blocks of Visualization

Grammar of Graphics. Wilkinson. *2015.*

Data Input data to visualize

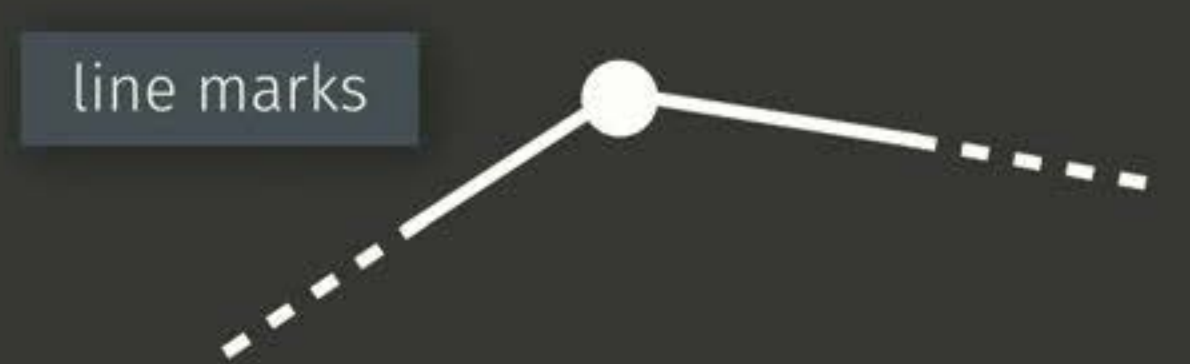
```
weather.csv
```

Transforms Filter, aggregation, binning, etc

```
aggregate temperature,
group by month of date and city
```

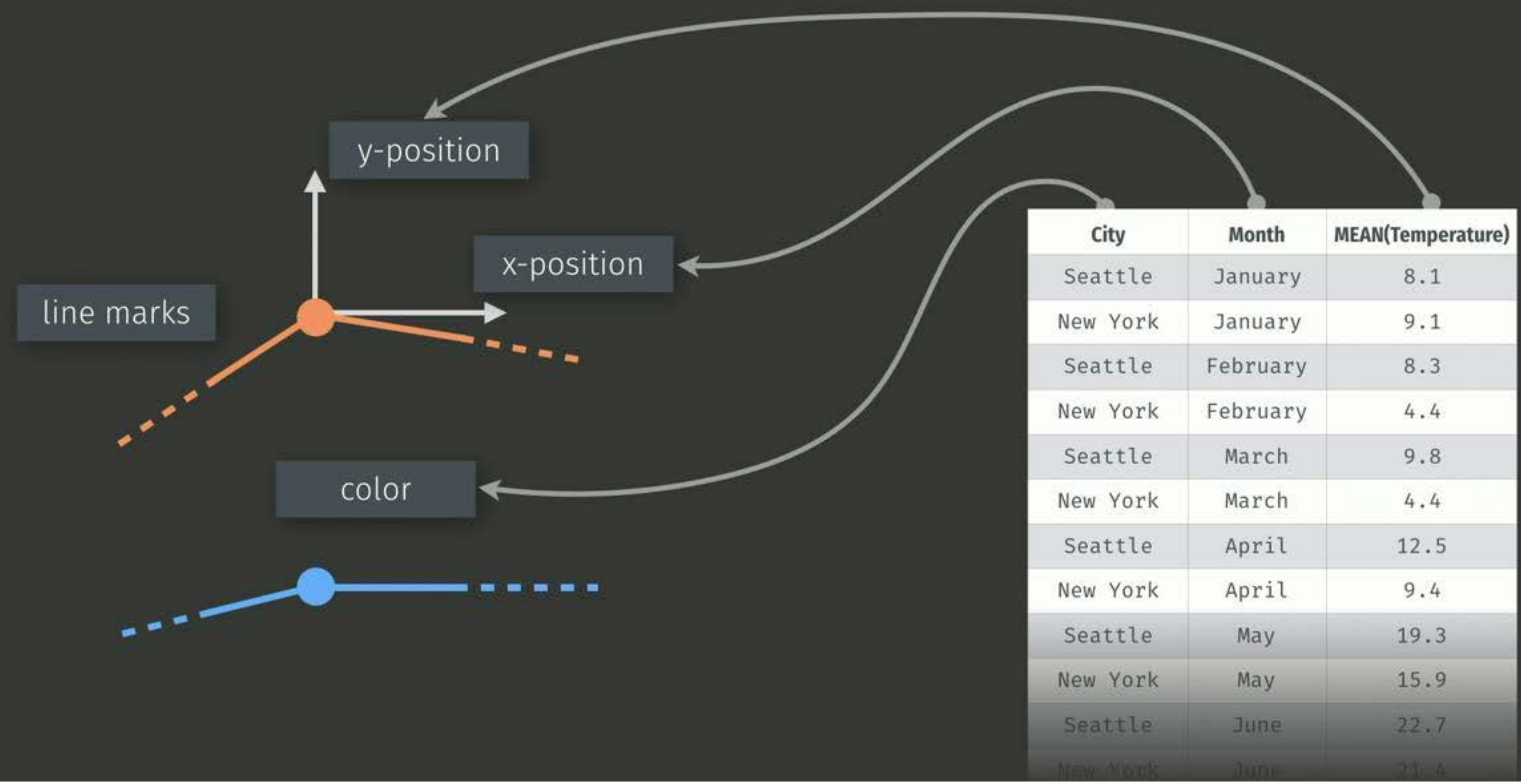
City	Month	MEAN(Temperature)
Seattle	January	8.1
New York	January	9.1
Seattle	February	8.3
New York	February	4.4
Seattle	March	9.8
New York	March	4.4
Seattle	April	12.5
New York	April	9.4
Seattle	May	19.3
New York	May	15.9
Seattle	June	22.7
New York	June	[illegible]

Visualizations Encode Data as Visual Properties



City	Month	MEAN(Temperature)
Seattle	January	8.1
New York	January	9.1
Seattle	February	8.3
New York	February	4.4
Seattle	March	9.8
New York	March	4.4
Seattle	April	12.5
New York	April	9.4
Seattle	May	19.3
New York	May	15.9
Seattle	June	22.7
New York	June	21.4

Visualizations Encode Data as Visual Properties



City	Month	MEAN(Temperature)
Seattle	January	8.1
New York	January	9.1
Seattle	February	8.3
New York	February	4.4
Seattle	March	9.8
New York	March	4.4
Seattle	April	12.5
New York	April	9.4
Seattle	May	19.3
New York	May	15.9
Seattle	June	22.7
New York	June	[illegible]

Building Blocks of Visualization

Grammar of Graphics. Wilkinson. *2015.*

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`

City	Month	MEAN(Temperature)
Seattle	January	8.1
New York	January	9.1
Seattle	February	8.3
New York	February	4.4
Seattle	March	9.8
New York	March	4.4
Seattle	April	12.5
New York	April	9.4
Seattle	May	19.3
New York	May	15.9
Seattle	June	22.7
New York	June	21.4

Building Blocks of Visualization

Grammar of Graphics. Wilkinson. *2015.*

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

Symbol

Rect

Line

Area

Text

Building Blocks of Visualization

Grammar of Graphics. Wilkinson. *2015.*

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

Symbol

Rect

Line

Area

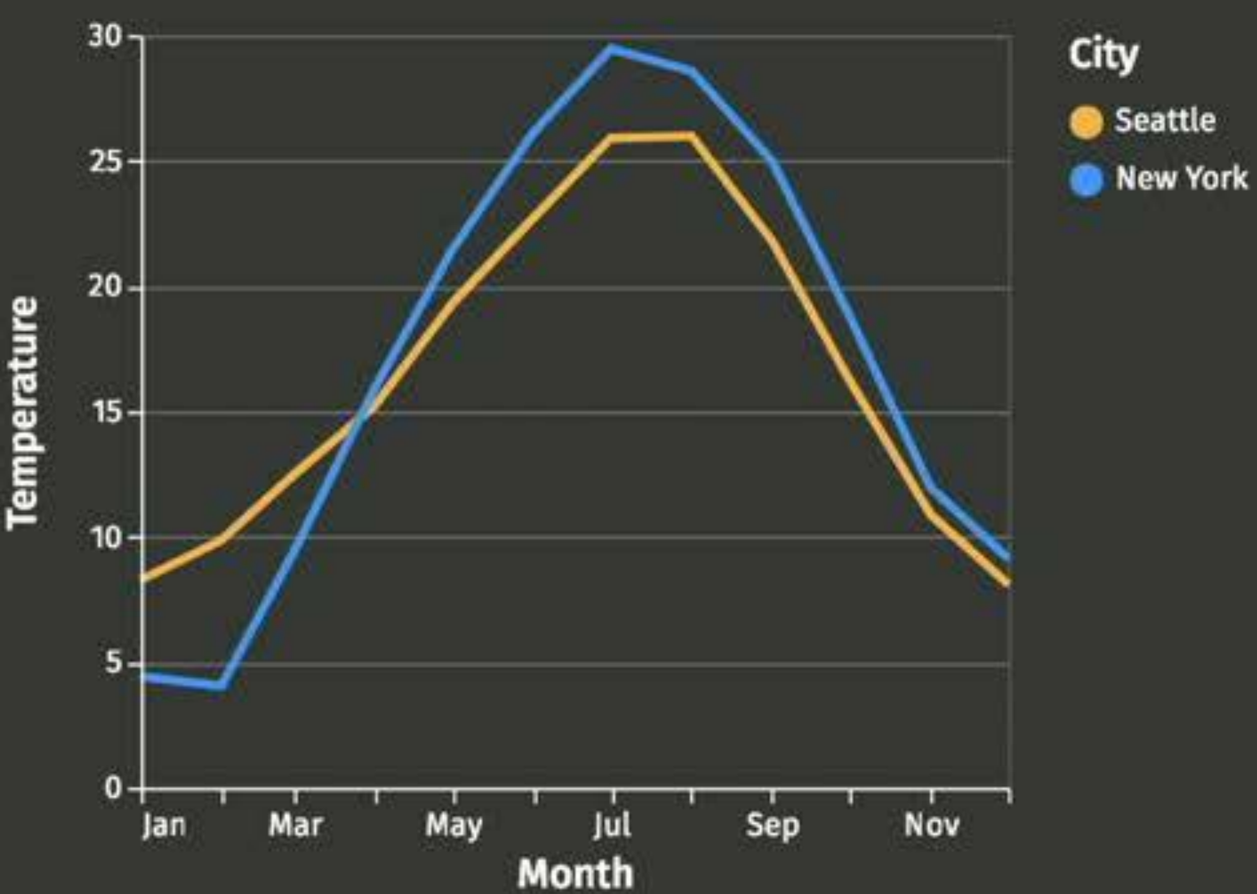
Text

Vega-Lite

vega.github.io/vega-lite

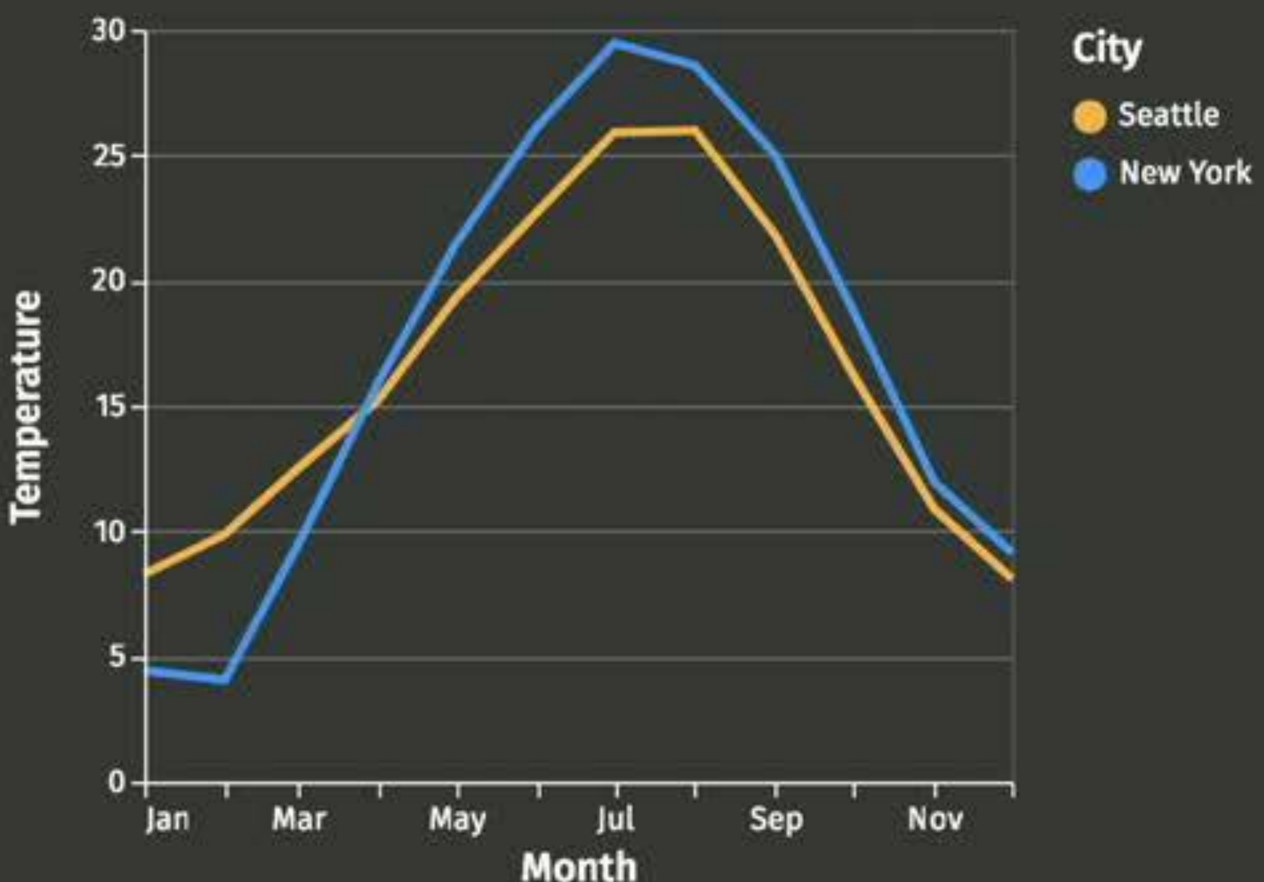
Vega-Lite Encodings

```yaml
data:
  url: weather.csv
```



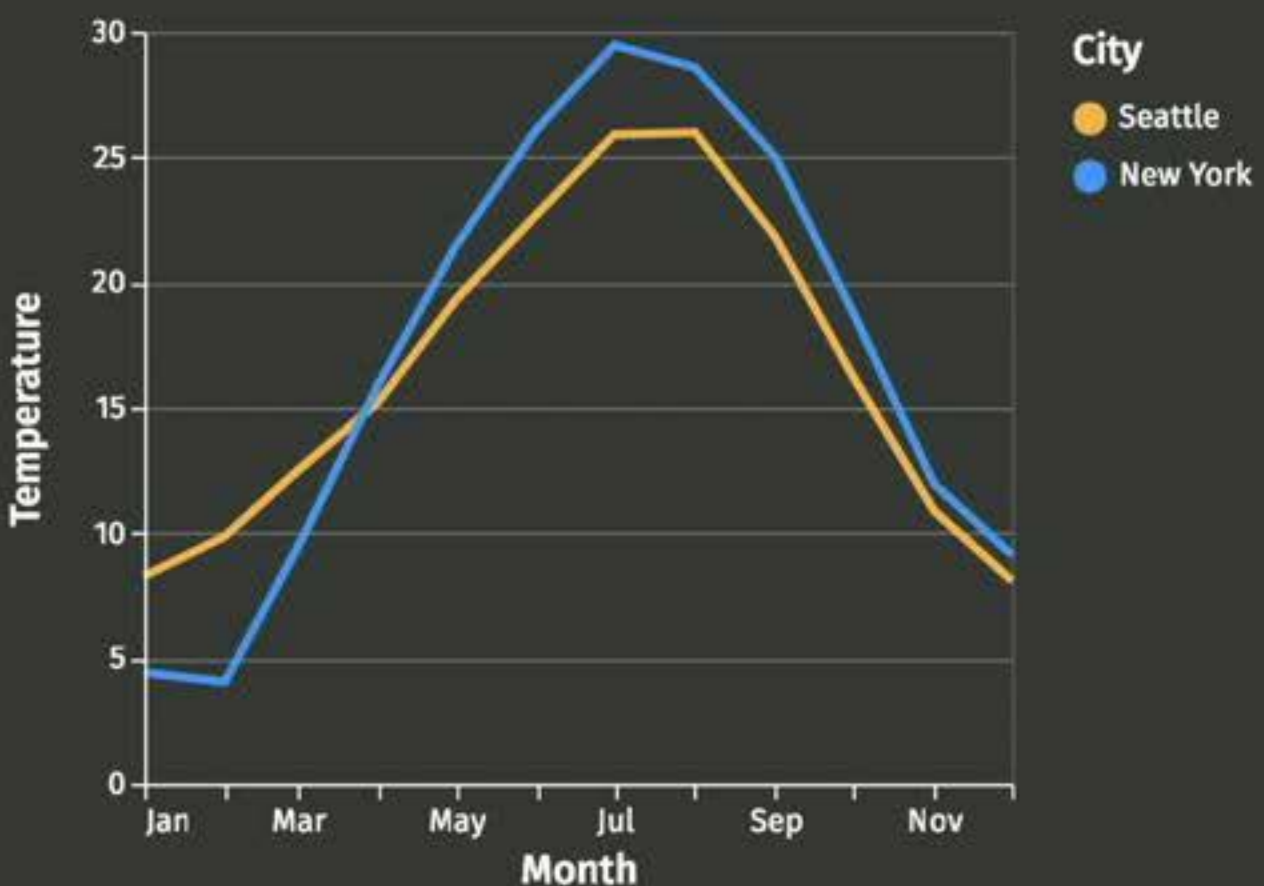
Vega-Lite Encodings

```yaml
data:
  url: weather.csv
mark: line
```



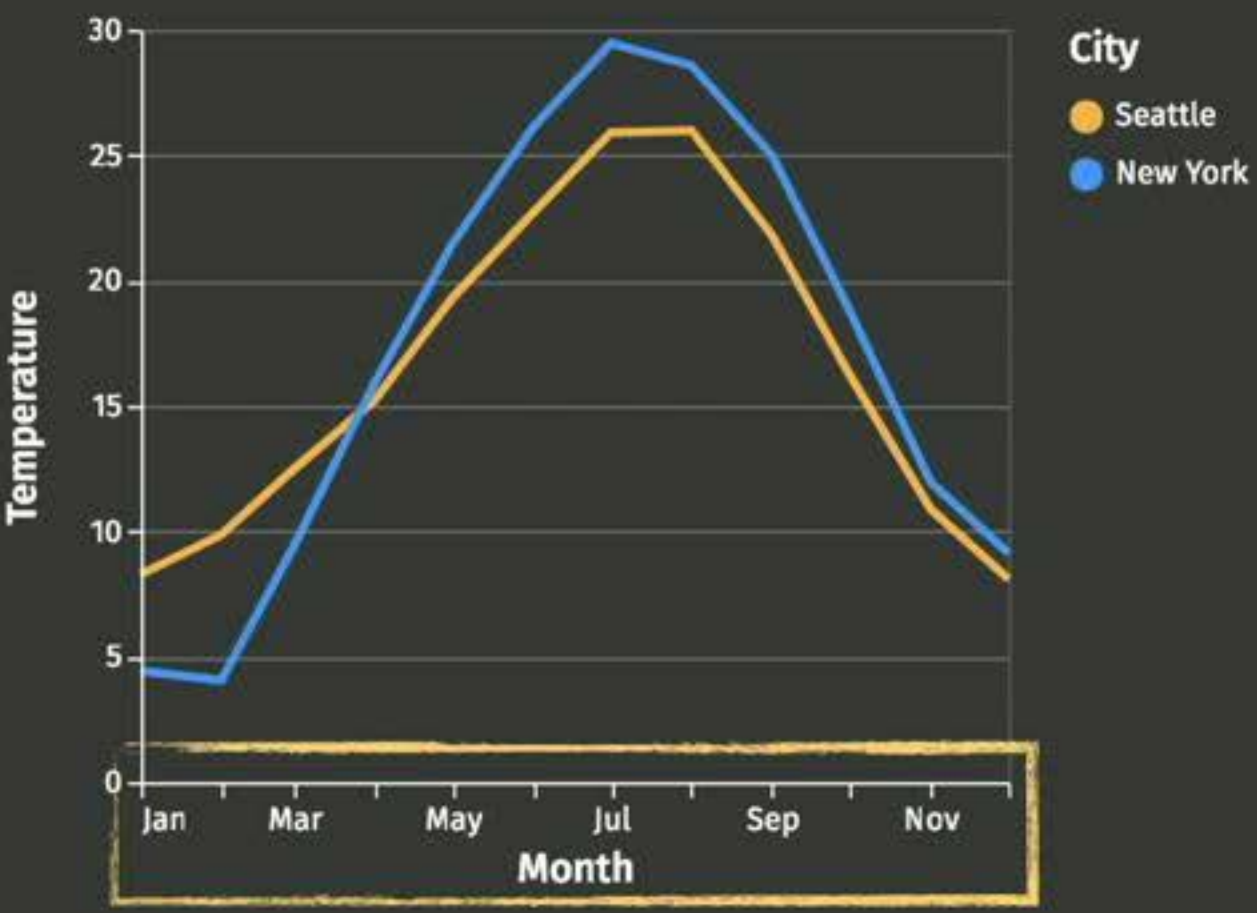
Vega-Lite Encodings

```
data:
  url: weather.csv
mark: line
encoding:
```



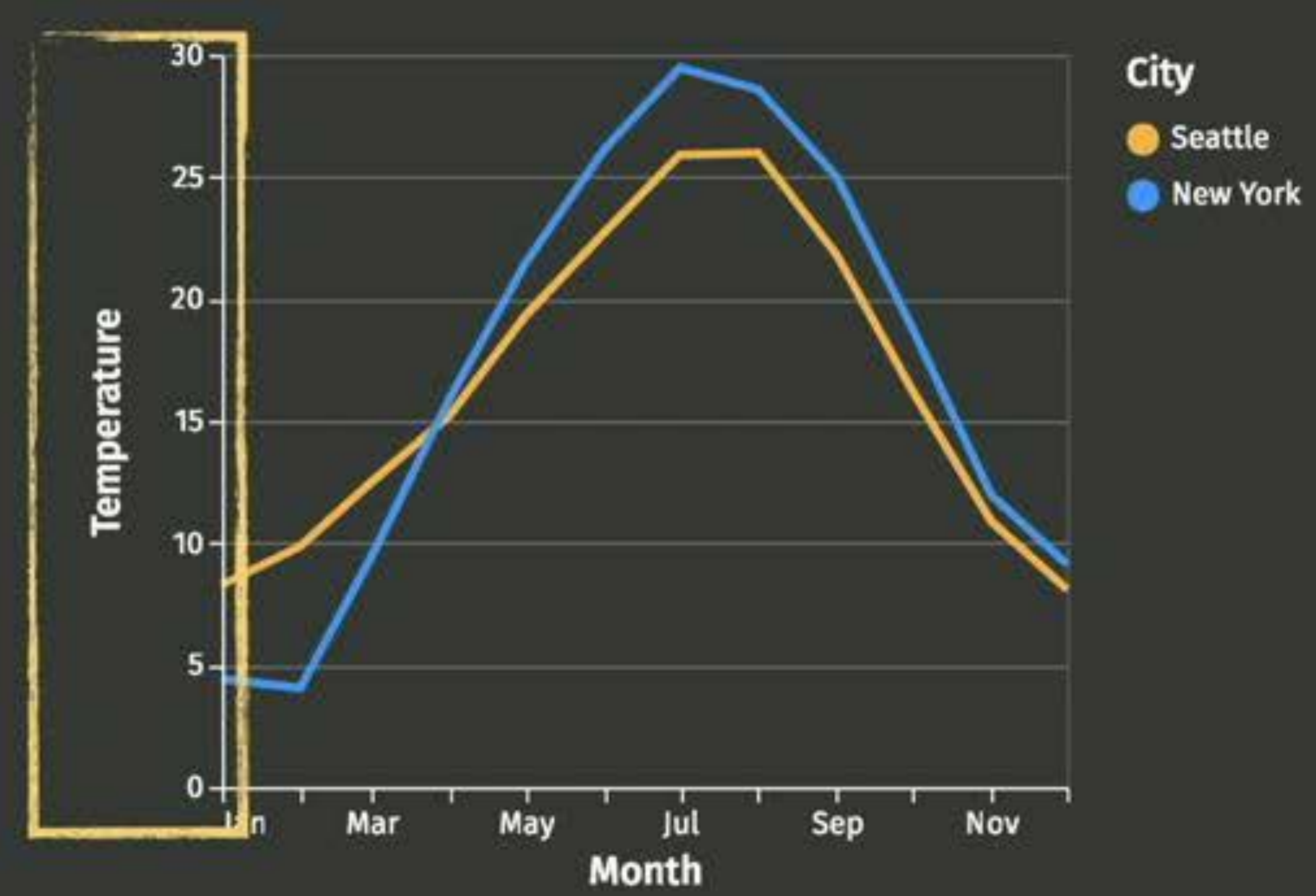
Vega-Lite Encodings

```
data:
  url: weather.csv
mark: line
encoding:
  x:
    field: date, type: temporal
    timeUnit: monthdate
```



Vega-Lite Encodings

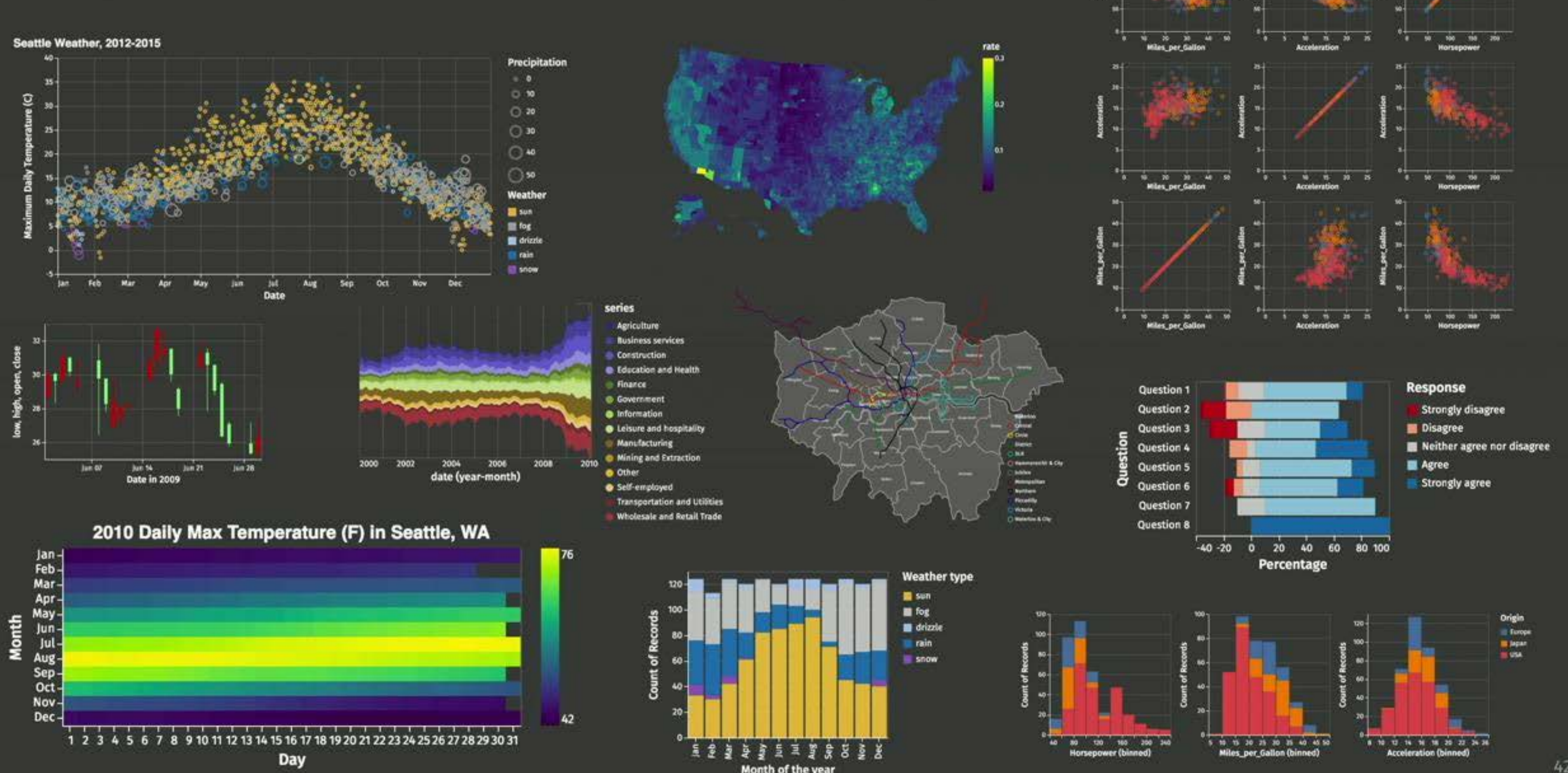
```
data:
  url: weather.csv
mark: line
encoding:
  x:
    field: date, type: temporal
    timeUnit: monthdate
  y:
    field: temperature, type: quantitative
    aggregate: mean
```



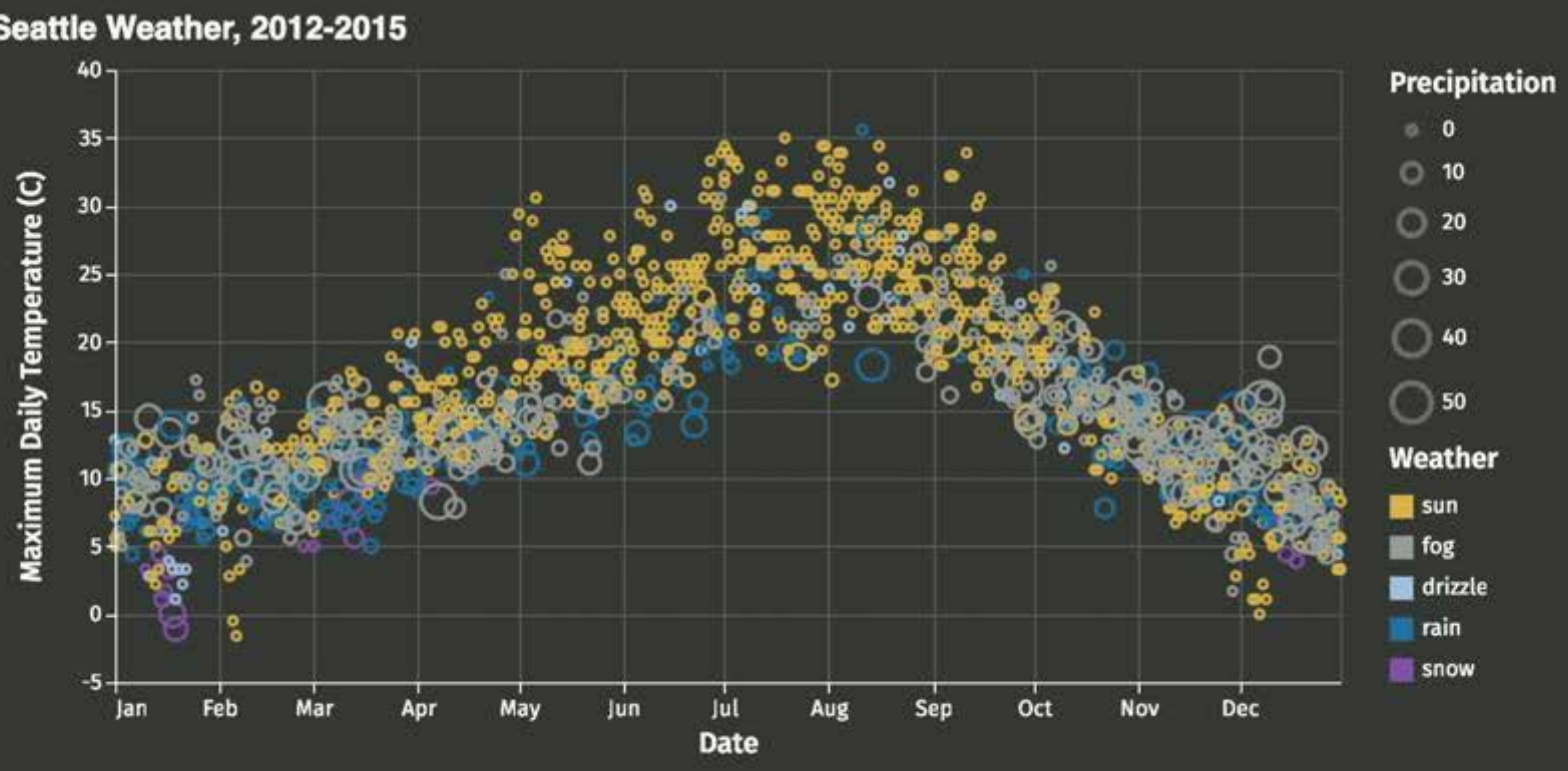
Vega-Lite Encodings

```
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
```

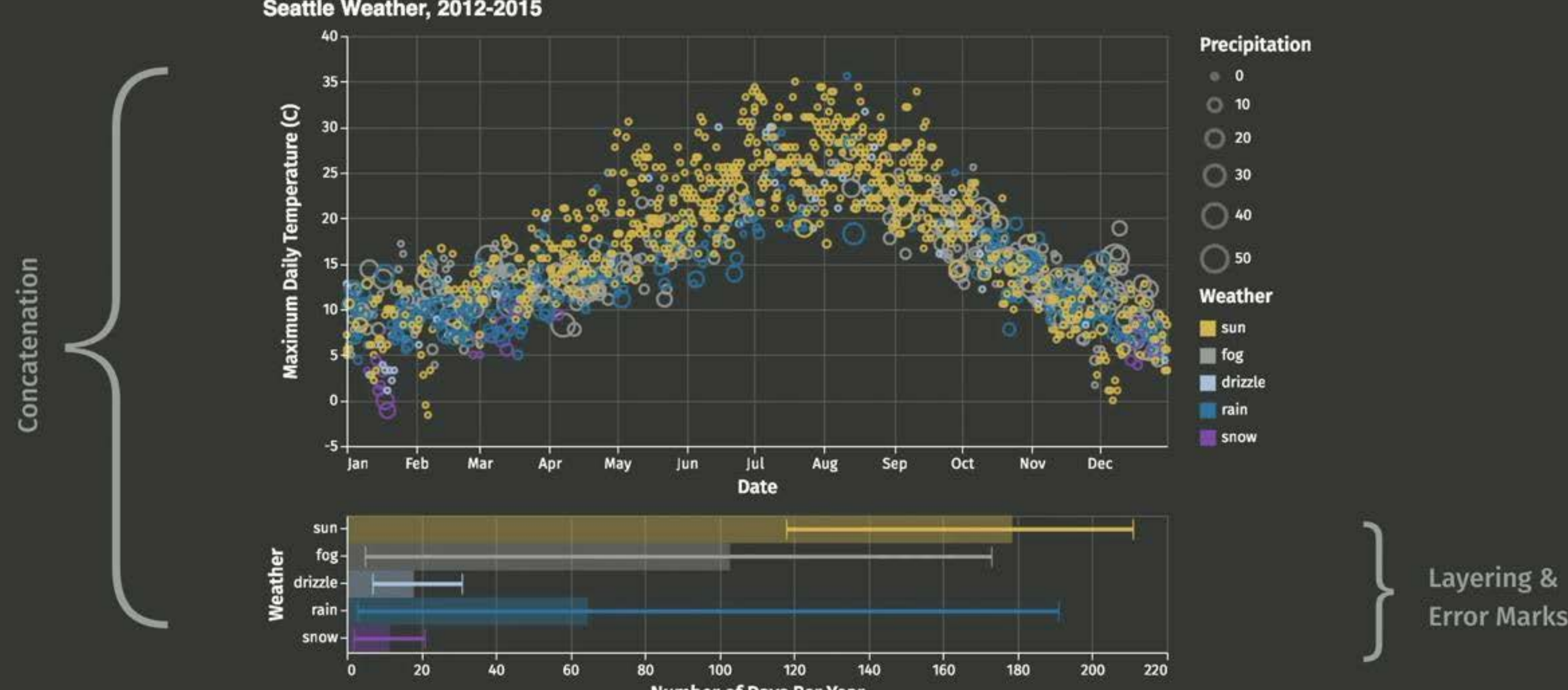
Vega-Lite is an Expressive Language.



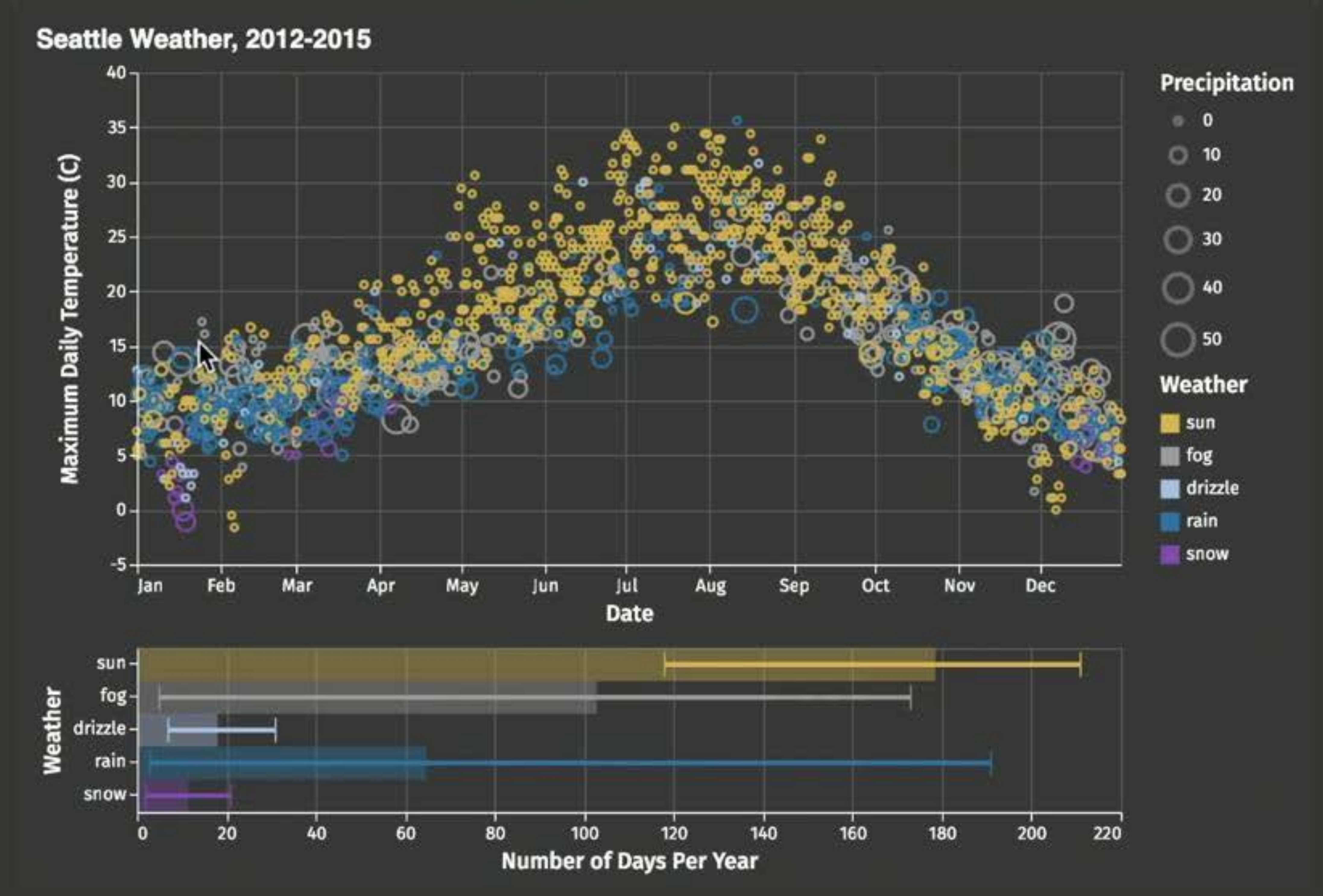
Vega-Lite is an Expressive Language for **Statistical** Graphics.



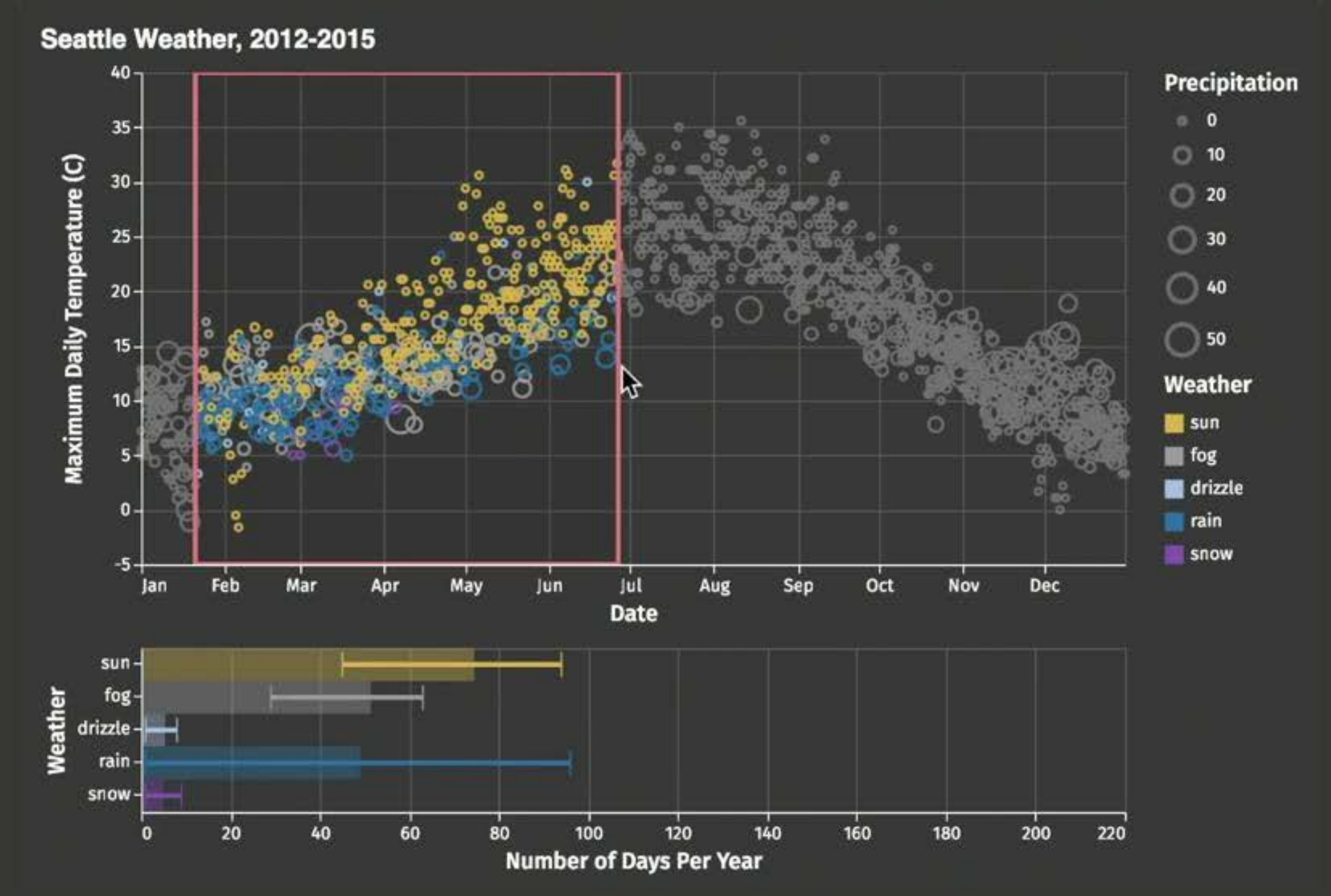
Vega-Lite is an Expressive Language for Statistical **Multi-View** Graphics.



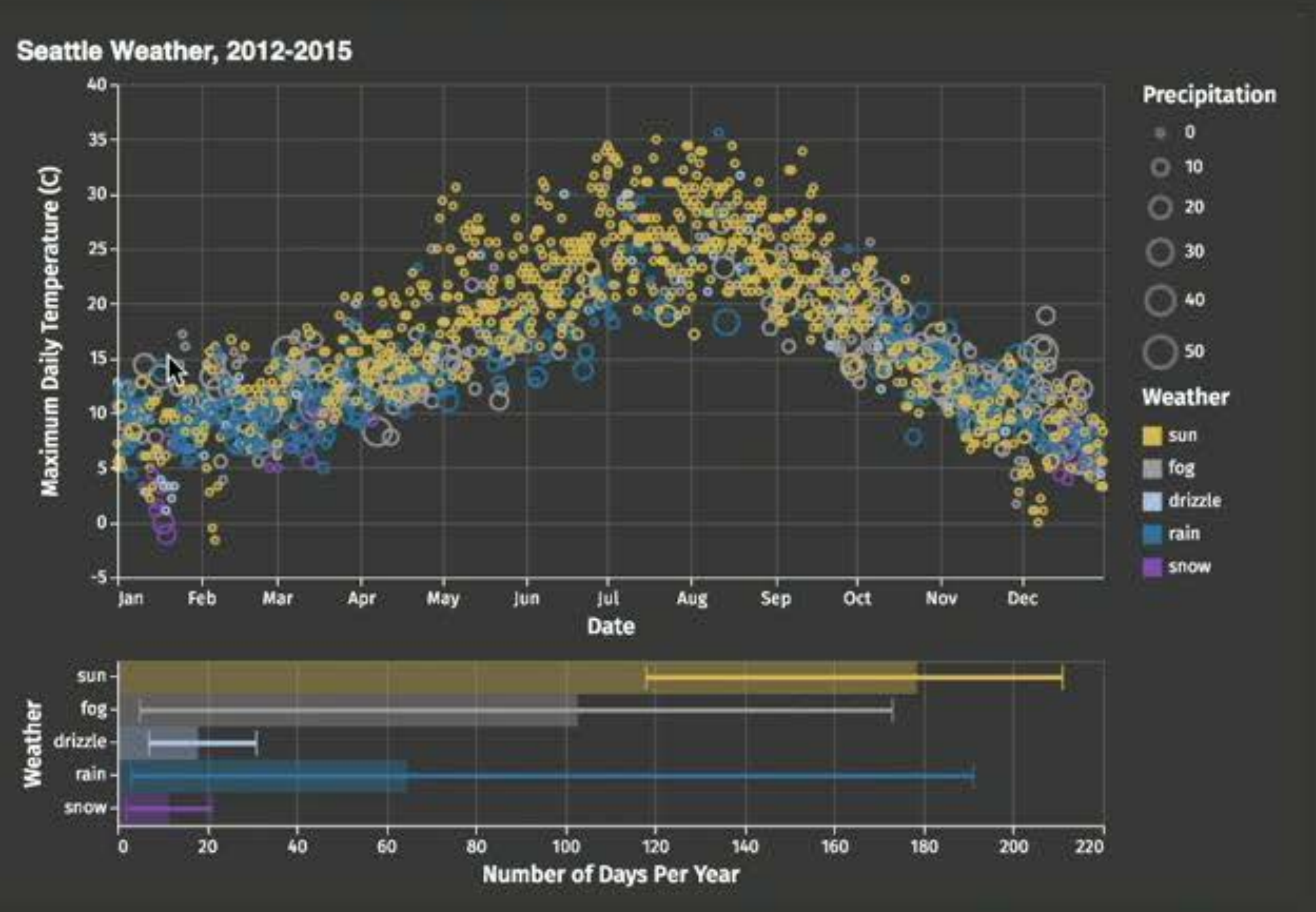
Vega-Lite is an Expressive Language for Statistical **Interactive** Multi-View Graphics.



Vega-Lite is an Expressive Language for Statistical **Interactive** Multi-View Graphics.

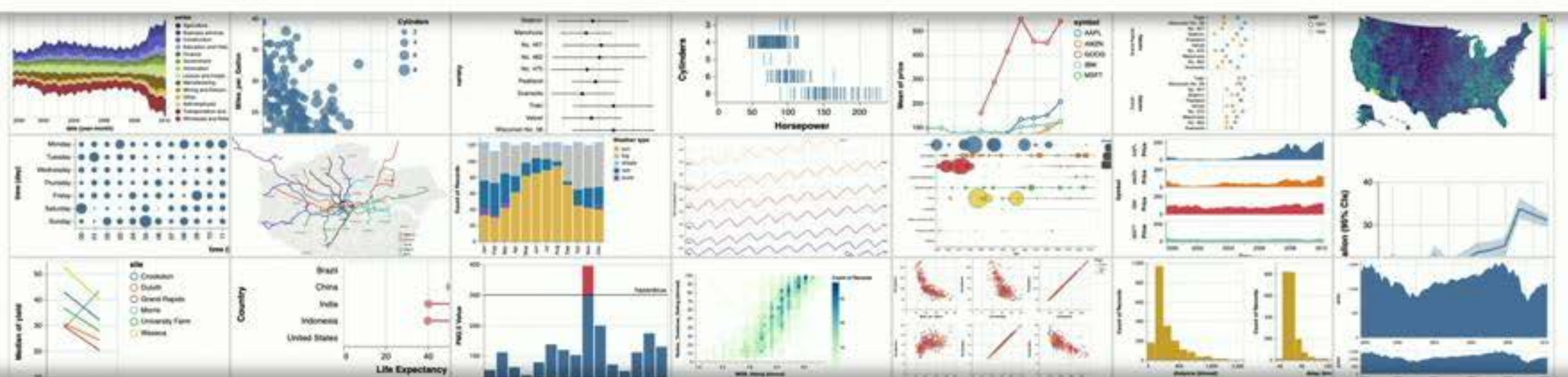


```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v3.json",
  "title": "Seattle Weather, 2012-2015",
  "data": {
    "url": "data/seattle-weather.csv"
  },
  "vconcat": [
    {
      "width": 600,
      "height": 300,
      "mark": "point",
      "selection": {"brush": {"encodings": ["x"], "type": "interval"}},
      "transform": [{"filter": {"selection": "click"}}],
      "encoding": {
        "color": {
          "condition": {
            "field": "weather",
            "selection": "brush",
            "type": "nominal"
          },
          "value": "lightgray"
        },
        "size": {
          "field": "precipitation",
          "scale": {"domain": [-1, 50]},
          "type": "quantitative"
        },
        "x": {
          "axis": {"title": "Date", "format": "%b"},
          "field": "date",
          "timeUnit": "monthdate",
          "type": "temporal"
        },
        "y": {
          "field": "temp_max",
          "scale": {"domain": [-5, 40]},
          "type": "quantitative"
        }
      }
    },
    {
      "width": 600,
      "mark": "bar",
      "selection": {"click": {"encodings": ["color"], "type": "multi"}},
      "transform": [{"filter": {"selection": "brush"}}],
      "encoding": {
        "color": {
          "condition": {
            "field": "weather",
            "selection": "click",
            "type": "nominal"
          },
          "value": "lightgray"
        },
        "x": {"aggregate": "count", "type": "quantitative"},
        "y": {"field": "weather", "type": "nominal"}
      }
    }
  ]
}
```



Vega-Lite Examples Tutorials Documentation Usage Applications GitHub Try Online

Vega-Lite – A Grammar of Interactive Graphics



{:.lead} **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**.

Get started
Latest Version: 3.0.0-rc12

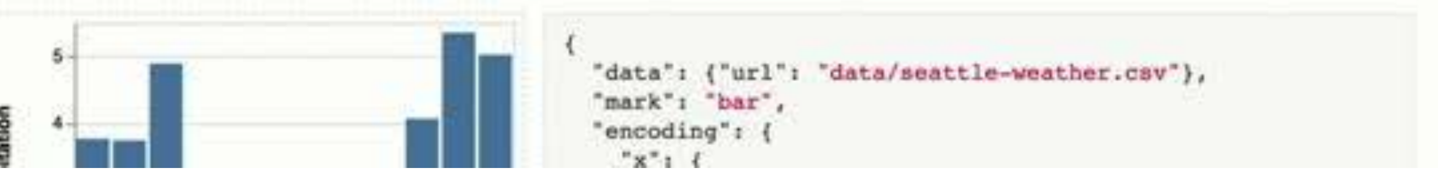
Try online

Read our intr … out the documentatio…

vega.github.io/vega-lite

Exampl…

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. Next step



```
{
  "data": {"url": "data/seattle-weather.csv"},
  "mark": "bar",
  "encoding": {
    "x": {
```

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

```
{
  "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**

Voyager 2. Wongsuphasawat et al. *CHI 2017.*

More: https://vega.github.io/vega-lite/applications.html

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"
    }
  }
}
```

Altair in Python

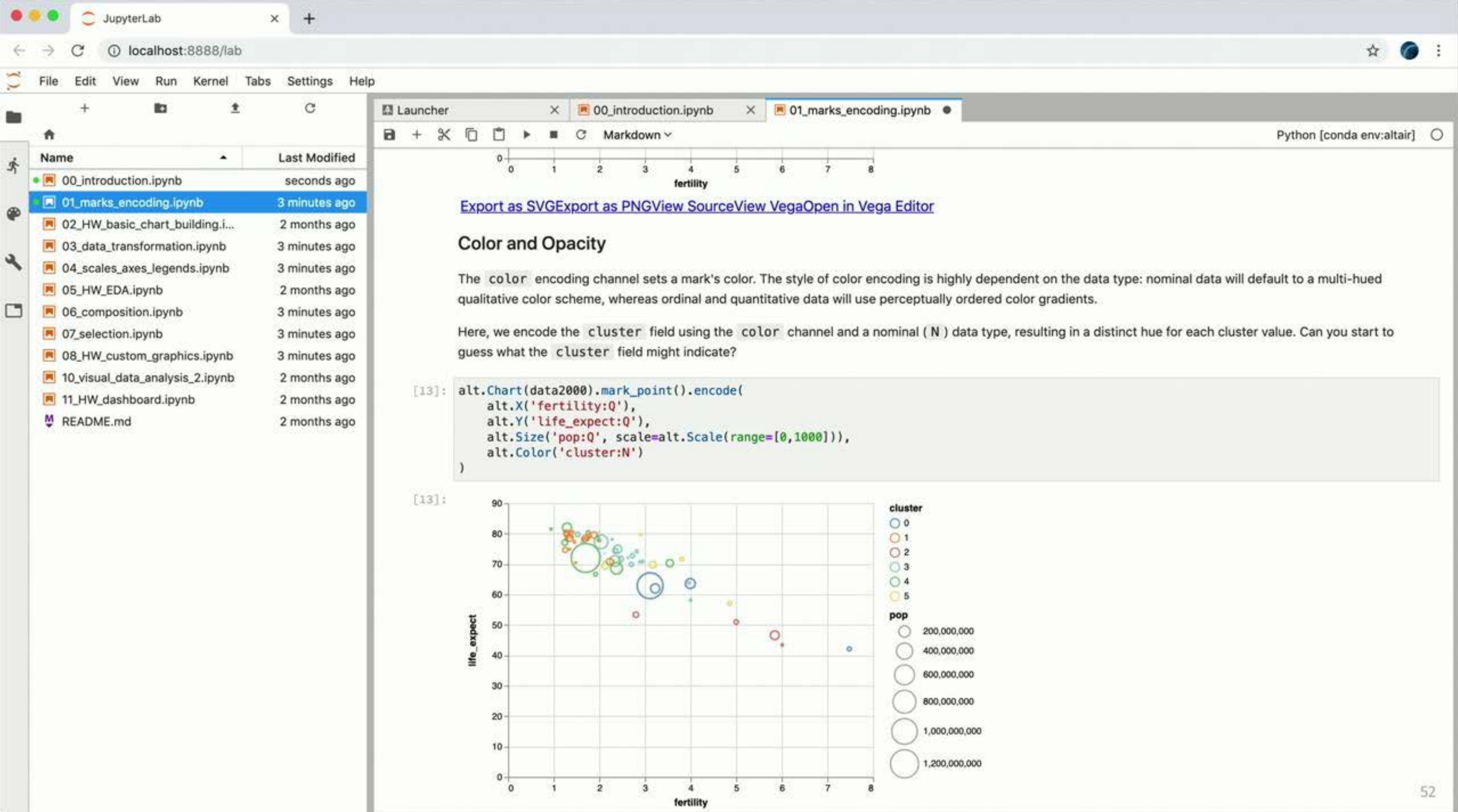
Altair. VanderPlas et al. *JOSS 2018.*

```
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')
```

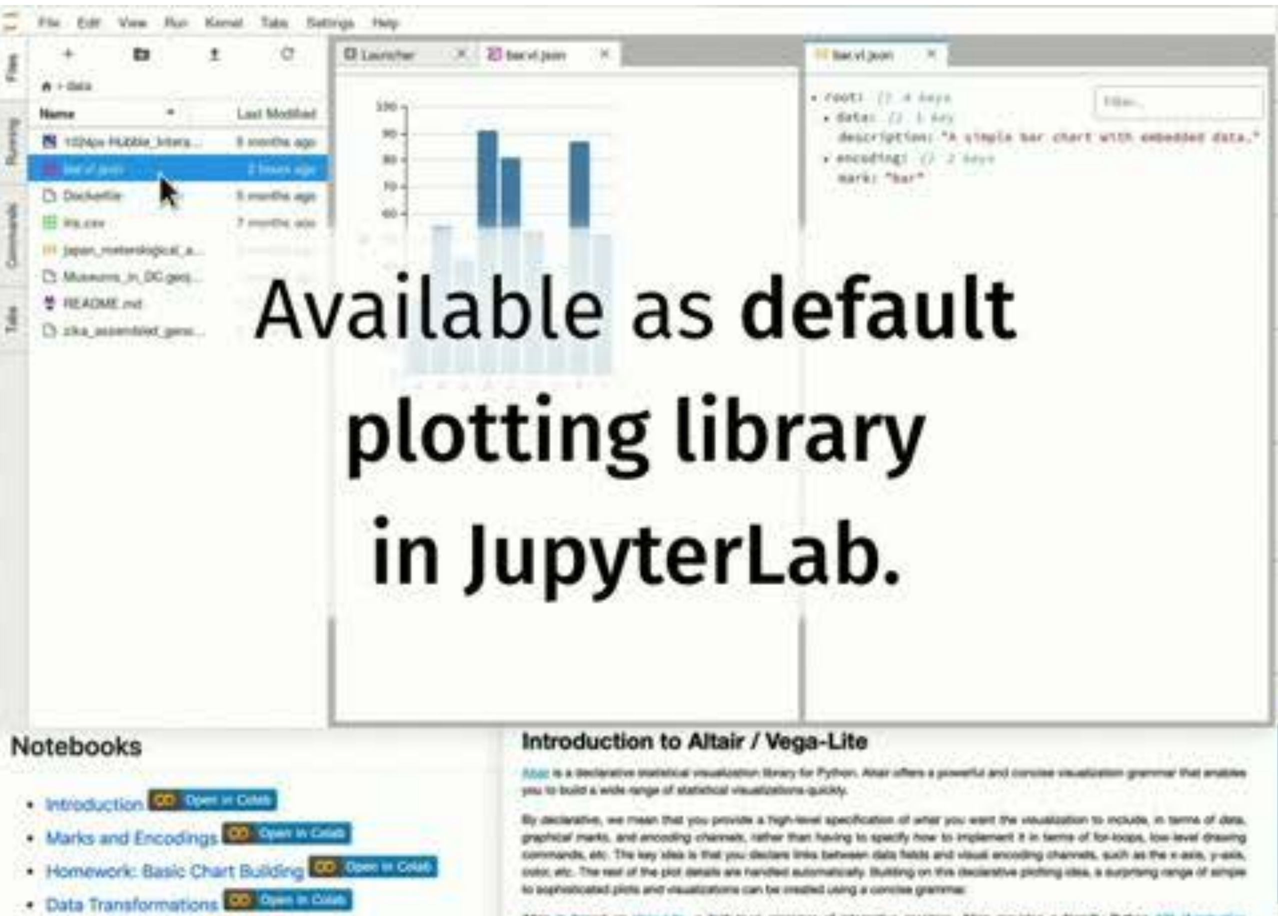
goo.gl/6ihGo2

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

jupyter



JupyterLab
localhost:8888/lab
File Edit View Run Kernel Tabs Settings Help
Name
Last Modified
00_introduction.ipynb seconds ago
01_marks_encoding.ipynb 3 minutes ago
02_HW_basic_chart_building.i... 2 months ago
03_data_transformation.ipynb 3 minutes ago
04_scales_axes_legends.ipynb 3 minutes ago
05_HW_EDA.ipynb 2 months ago
06_composition.ipynb 3 minutes ago
07_selection.ipynb 3 minutes ago
08_HW_custom_graphics.ipynb 3 minutes ago
10_visual_data_analysis_2.ipynb 2 months ago
11_HW_dashboard.ipynb 2 months ago
README.md 2 months ago
Launcher
00_introduction.ipynb
01_marks_encoding.ipynb
Markdown
Python [conda env:altair]
fertility
Export as SVGExport as PNGView SourceView VegaOpen in Vega Editor
Color and Opacity
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('pop:Q', scale=alt.Scale(range=[0,1000])),
    alt.Color('cluster:N')
)
[13]:
life_expect
fertility
cluster
0
1
2
3
4
5
pop
200,000,000
400,000,000
600,000,000
800,000,000
1,000,000,000
1,200,000,000
52

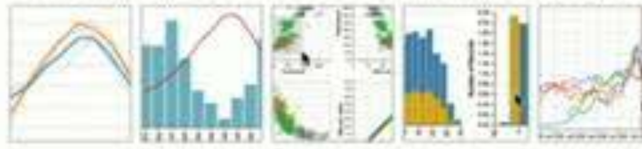


NPM Downloads for Vega-Lite per Month

Dec 2017, Jan 2018, Feb 2018, Mar 2018, Apr 2018, May 2018, Jun 2018, Jul 2018, Aug 2018, Sep 2018, Oct 2018, Nov 2018, Dec 2018

Vega-Lite: A Grammar of Interactive Graphics

Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer



Notebooks

- Introduction
- Marks and Encodings
- Homework: Basic Chart Building
- Data Transformations
- Visual Data Analysis
- Homework: EDA
- Scales, Axes, Legends
- Layering and Composition
- Homework: Multi-View Composition
- Selections
- Visual Data Analysis 2
- Homework: Interactive Dashboards

Introduction to Altair / Vega-Lite

Imports

```
import pandas as pd
import altair as alt
```

Altair column sort example

By Ben Welsh

An example of how to sort the columns in a bar chart created by the Altair data visualization library. Created in response to a question from Joe Germuska.

```
In [1]: import pandas as pd
        import altair as alt
```

Read in the U.S. Census data file provided by Joe, median household income by county

```
In [2]: df = pd.read_csv("input/mhhi_by_county.csv")
In [3]: df.head()
```

Out[3]:

	Unnamed: 0	name	geoid	b19013001
0	0	Anchorage Municipality, Alaska	05000US02020	85634.0
1	1	Fairbanks North Star Borough, Alaska	05000US02090	77328.0
2	2	Matanuska-Susitna Borough, Alaska	05000US02170	69332.0
3	3	Baldwin County, Alabama	05000US01003	56732.0
4	4	Calhoun County, Alabama	05000US01015	41687.0

Make the chart.

Data visualization tools drive interactivity and reproducibility in online publishing

New tools for building interactive figures and software make scientific data more accessible, and reproducible.

Vega-Lite as a File Format

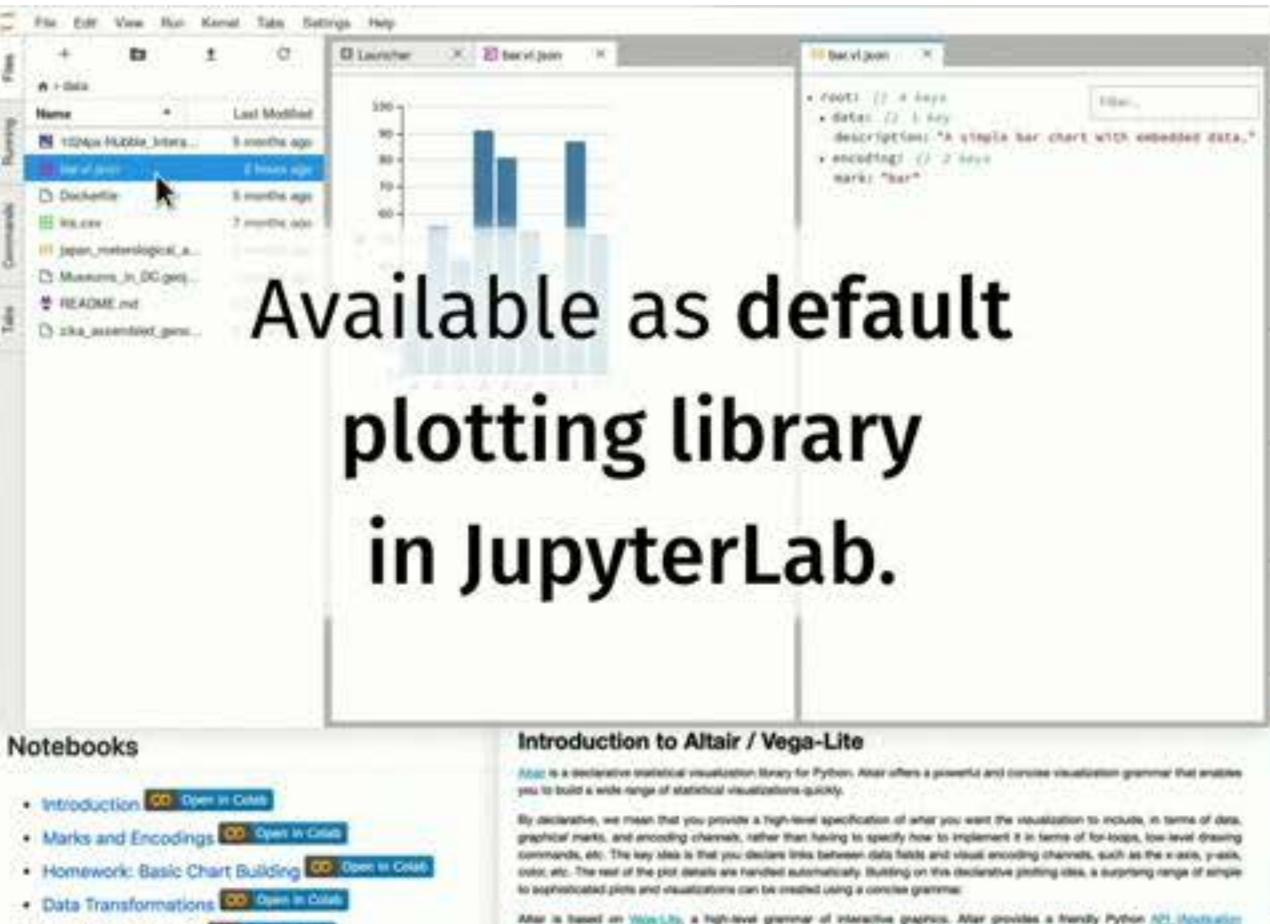
```
{
  "data": {
    "url": "weather.csv"
  },
  "mark": "line",
  "encoding": {
    "x": {
      "field": "date",
      "type": "temporal",
      "timeUnit": "monthdate"
    },
    "y": {
      "field": "temperature",
      "type": "quantitative",
      "aggregate": "mean"
    },
    "color": {
  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

Altair in Python

Altair. VanderPlas et al. *JOSS 2018.*

```
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

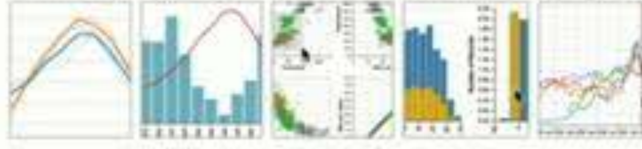


NPM Downloads for Vega-Lite per Month

Dec 2017
Jan 2018
Feb 2018
Mar 2018
Apr 2018
May 2018
Jun 2018
Jul 2018
Aug 2018
Sep 2018
Oct 2018
Nov 2018
Dec 2018

Vega-Lite: A Grammar of Interactive Graphics

Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer



Notebooks

- Introduction Open in Colab
- Marks and Encodings Open in Colab
- Homework: Basic Chart Building Open in Colab
- Data Transformations Open in Colab
- Visual Data Analysis Open in Colab
- Homework: EDA Open in Colab
- Scales, Axes, Legends Open in Colab
- Layering and Composition Open in Colab
- Homework: Multi-View Composition Open in Colab
- Selections Open in Colab
- Visual Data Analysis 2 Open in Colab
- Homework: Interactive Dashboards Open in Colab

Introduction to Altair / Vega-Lite

Imports

To start, we must import the necessary libraries: Pandas for data frames and Altair for visualization.

```
import pandas as pd
import altair as alt
```

Altair column sort example

By Ben Welsh

An example of how to sort the columns in a bar chart created by the Altair data visualization library. Created in response to a question from Joe Germuska.

```
In [1]: import pandas as pd
        import altair as alt
```

Read in the U.S. Census data file provided by Joe, median household income by county

```
In [2]: df = pd.read_csv("input/mhhi_by_county.csv")
```

```
In [3]: df.head()
```

Out[3]:

	Unnamed: 0	name	geoid	b19013001
0	0	Anchorage Municipality, Alaska	05000US02020	85634.0
1	1	Fairbanks North Star Borough, Alaska	05000US02090	77328.0
2	2	Matanuska-Susitna Borough, Alaska	05000US02170	69332.0
3	3	Baldwin County, Alabama	05000US01003	56732.0
4	4	Calhoun County, Alabama	05000US01015	41687.0

Make the chart.

Data visualization tools drive interactivity and reproducibility in online publishing

New tools for building interactive figures and software make scientific data more accessible, and reproducible.

Available as **default plotting library in JupyterLab.**

NPM Downloads for Vega-Lite per Month

Dec 2017, Jan 2018, Feb 2018, Mar 2018, Apr 2018, May 2018, Jun 2018, Jul 2018, Aug 2018, Sep 2018, Oct 2018, Nov 2018, Dec 2018

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

Vega-Lite: A Grammar of Interactive Graphics

Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer

Notebooks

- Introduction Open in Colab
- Marks and Encodings Open in Colab
- Homework: Basic Chart Building Open in Colab
- Data Transformations Open in Colab
- Visual Data Analysis Open in Colab
- Homework: EDA Open in Colab
- Scales, Axes, Legends Open in Colab
- Layering and Composition Open in Colab
- Homework: Multi-View Composition Open in Colab
- Selections Open in Colab
- Visual Data Analysis 2 Open in Colab
- Homework: Interactive Dashboards Open in Colab

Introduction to Altair / Vega-Lite

Imports

```
import pandas as pd
import altair as alt
```

Data visualization tools drive interactivity and reproducibility in online publishing

New tools for building interactive figures and software make scientific data more accessible, and reproducible.

Google

Altair column sort example

By Ben Welsh

An example of how to sort the columns in a bar chart created by the Altair data visualization library. Created in response to a question from Joe Germuska.

```
In [1]: import pandas as pd
        import altair as alt
```

Read in the U.S. Census data file provided by Joe, median household income by county

```
In [2]: df = pd.read_csv("input/mhhi_by_county.csv")
In [3]: df.head()
```

Out[3]:

	Unnamed: 0	name	geoid	b19013001
0	0	Anchorage Municipality, Alaska	05000US02020	85634.0
1	1	Fairbanks North Star Borough, Alaska	05000US02090	77328.0
2	2	Matanuska-Susitna Borough, Alaska	05000US02170	69332.0
3	3	Baldwin County, Alabama	05000US01003	56732.0
4	4	Calhoun County, Alabama	05000US01015	41687.0

Make the chart.

U.S. Airports, 2008

Available as **default plotting library in JupyterLab.**

NPM Downloads for Vega-Lite per Month

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

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

Notebooks

Introduction to Altair / Vega-Lite

Altair column sort example

Data visualization tools drive interactivity and reproducibility in online publishing

Google

U.S. Airports, 2008

Flight Routes in USA

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...

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

Featured by Nature.

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

Used by the LA Times

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

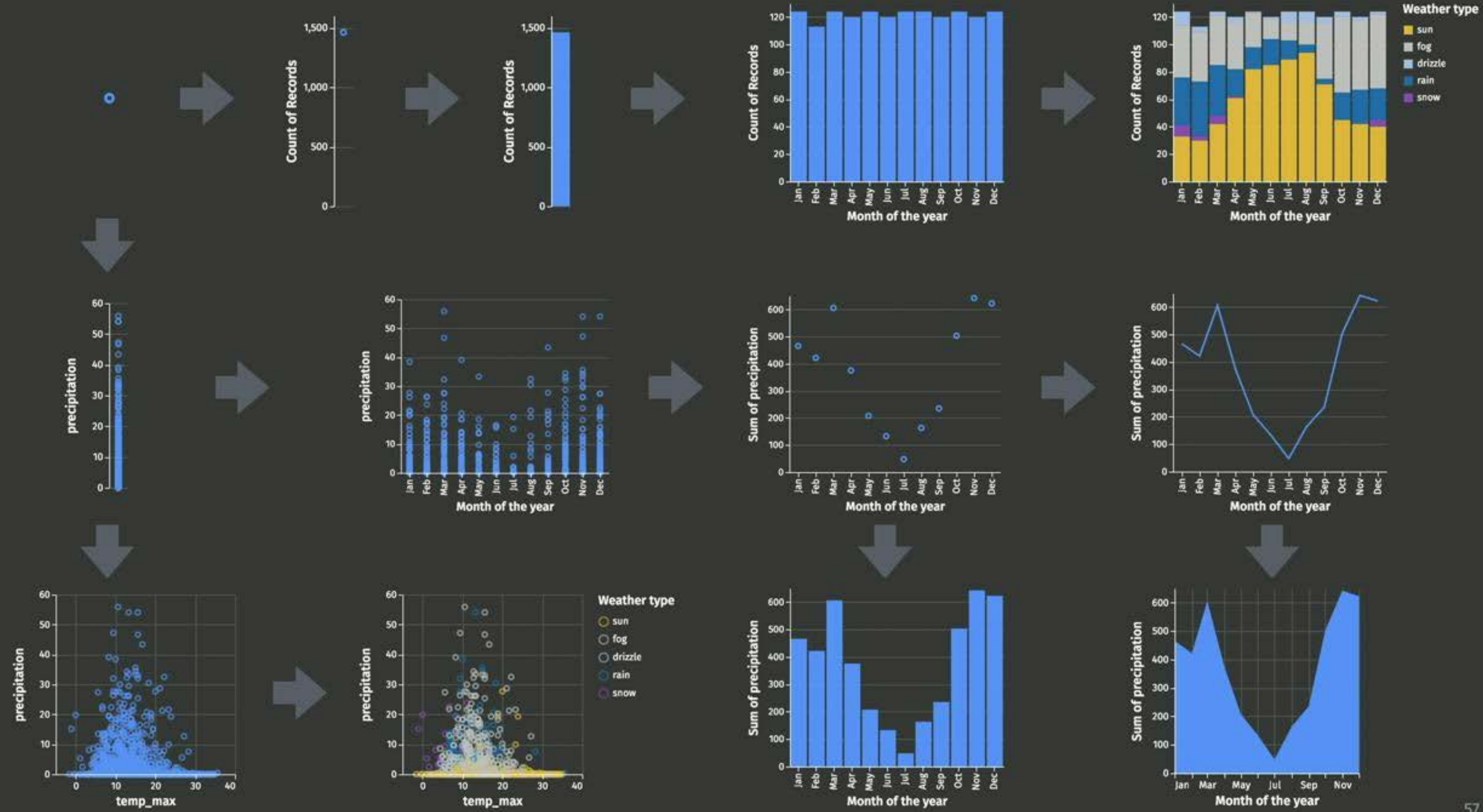
Facilitates rapid authoring for fast iterations

Designed for programmatic generation

Declarative: specification decoupled from execution

High-level, domain-specific abstractions

Composable building blocks



Count of Records
Month of the year
Weather type
sun
fog
drizzle
rain
snow
precipitation
Sum of precipitation
temp_max
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec

Draco

uwdata.github.io/draco

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.

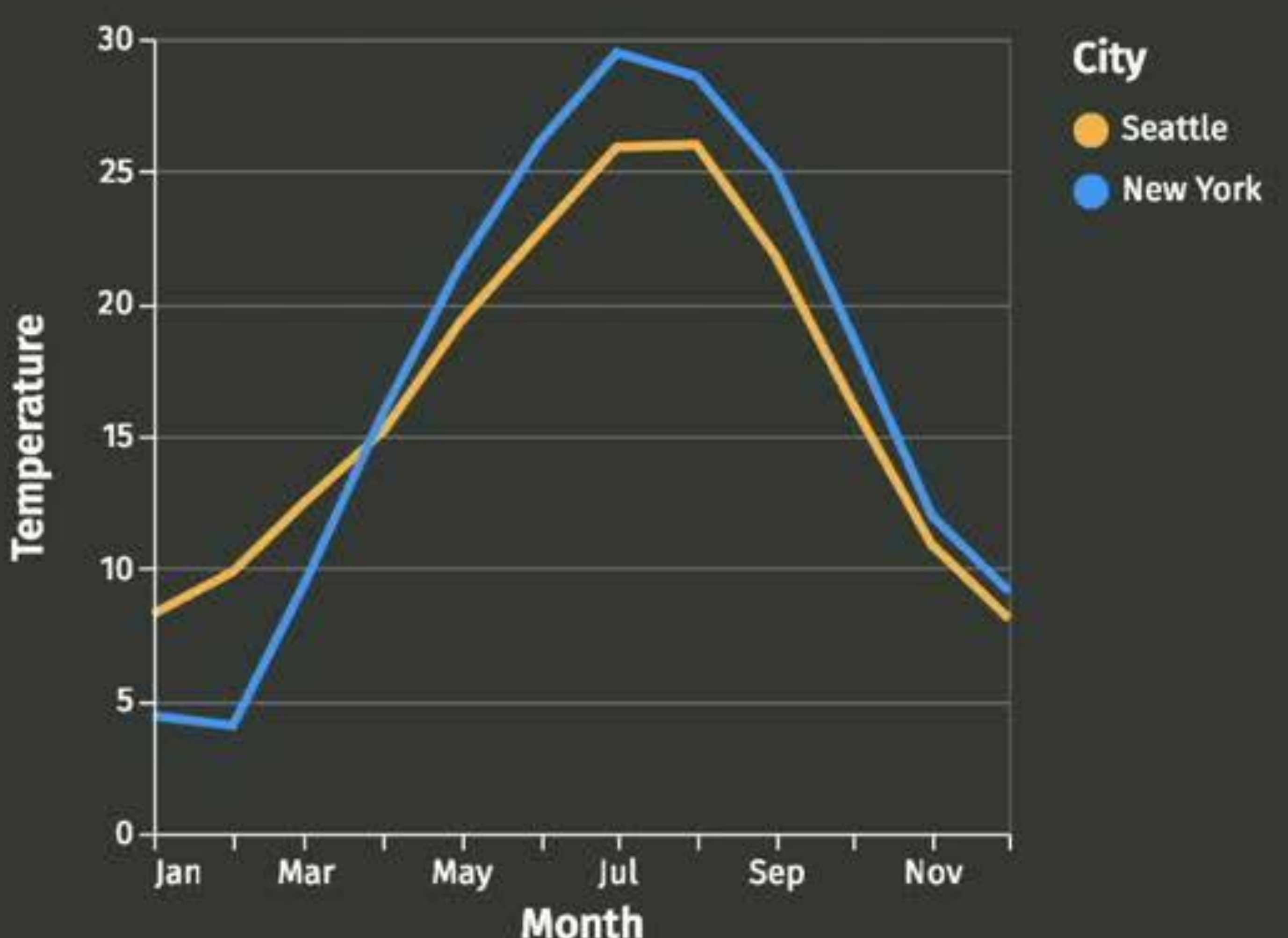
Draco

IEEE Infovis 2018. **Best Paper Award**

Formal model of visual encodings as sets of facts.

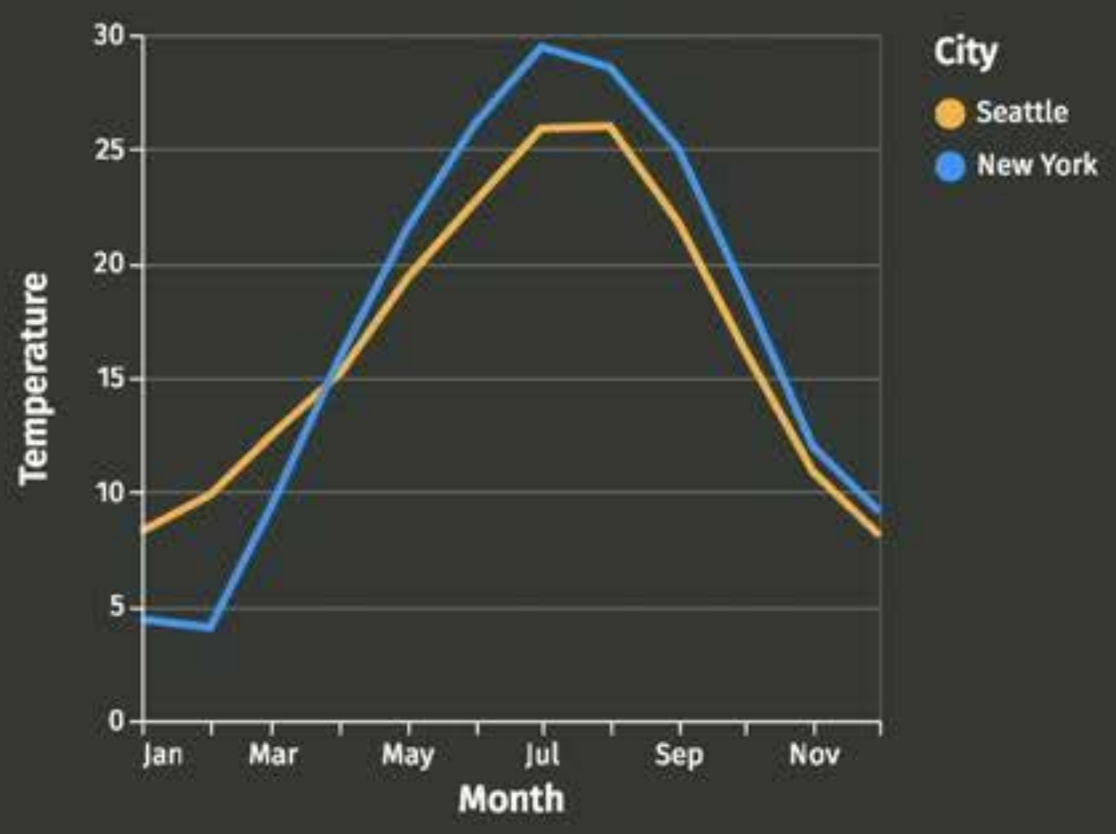
Design knowledge as constraints.

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



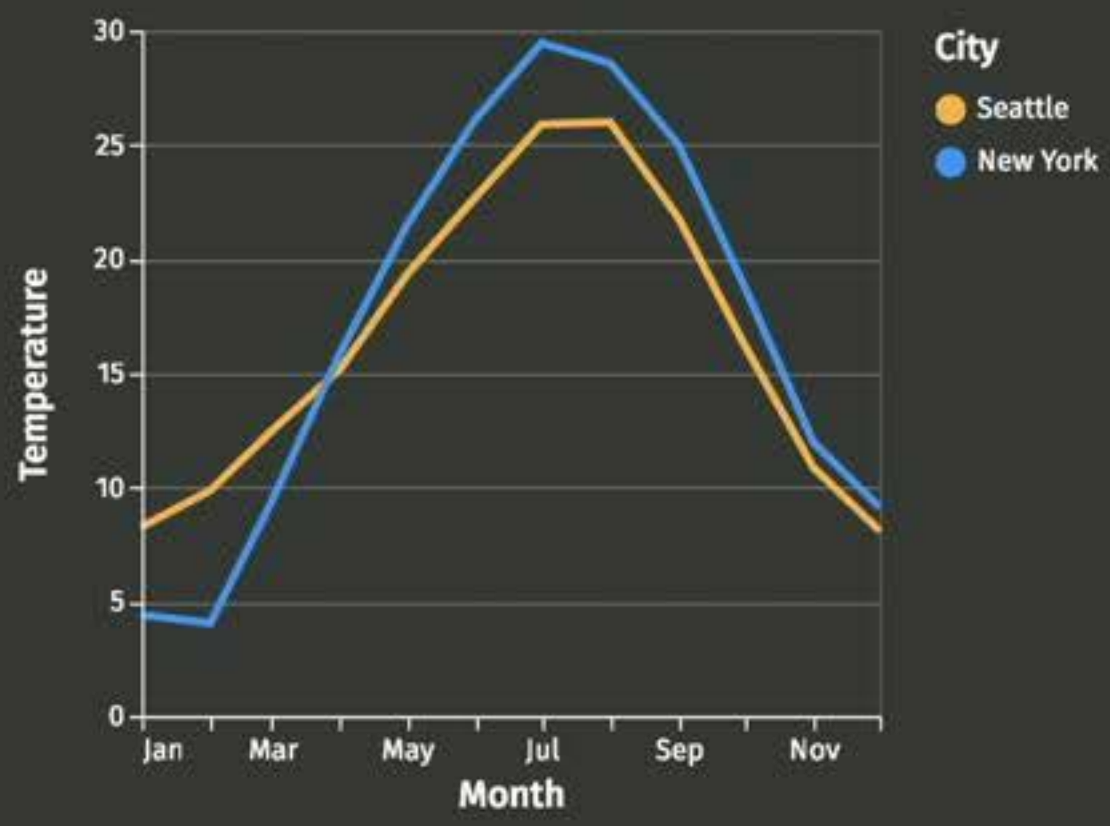
Vega-Lite

```
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

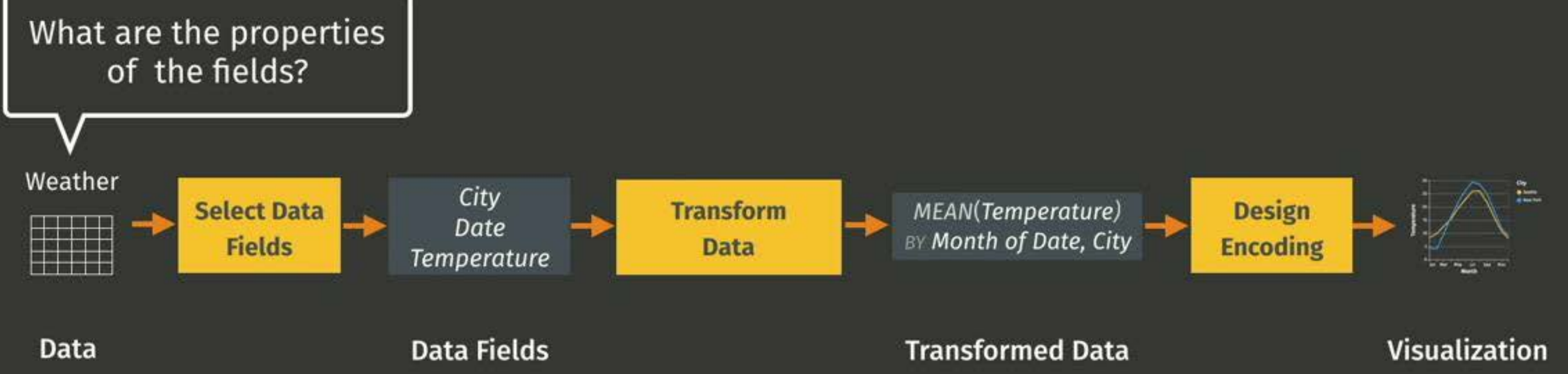
```
data("weather.csv").

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).
```

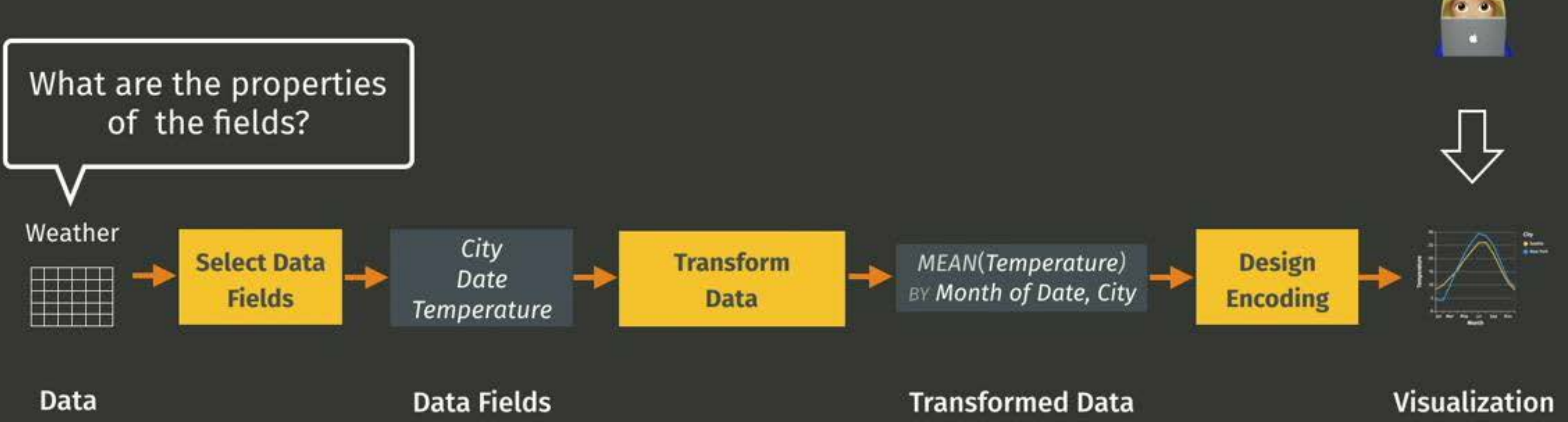
Draco: Encodings as Logical Facts




```
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).
```

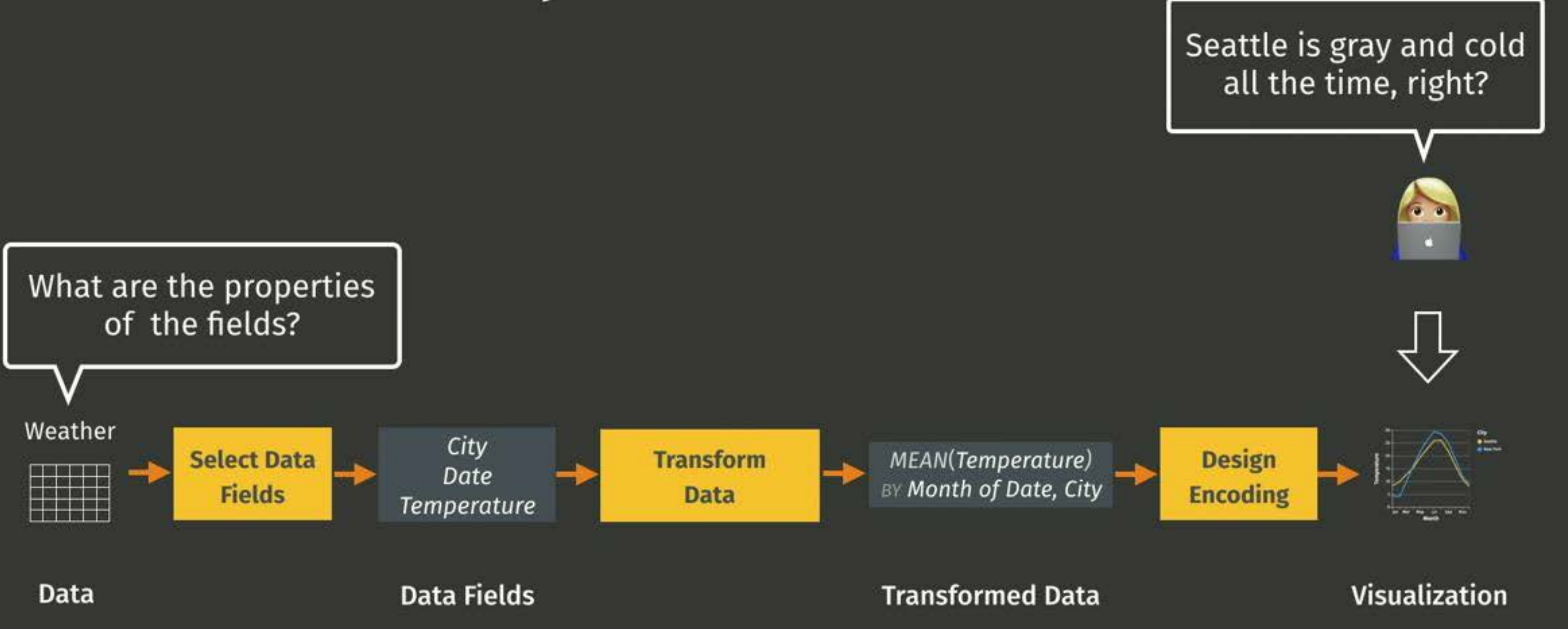
Weather
Select Data Fields
City
Date
Temperature
Transform Data
MEAN(Temperature)
BY Month of Date, City
Design Encoding
Data
Data Fields
Transformed Data
Visualization


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



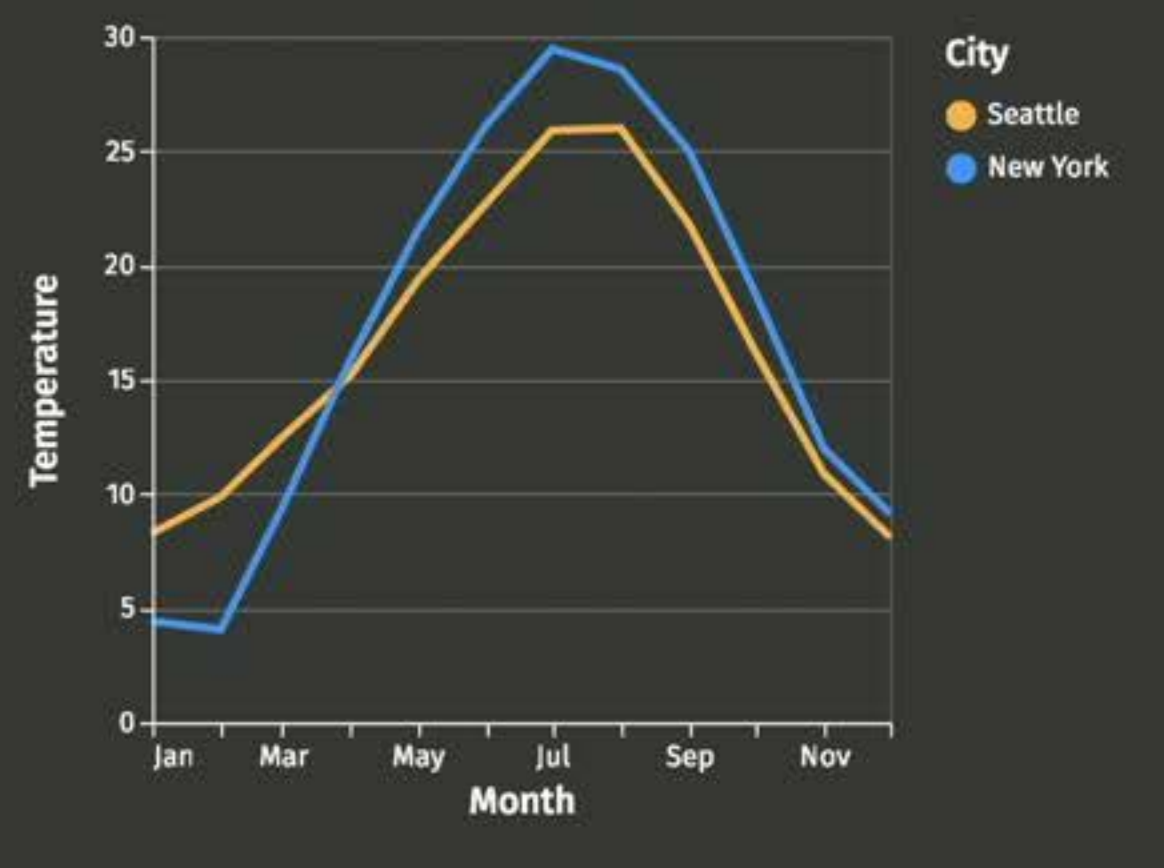
What are the properties of the fields?
Weather
Select Data Fields
City
Date
Temperature
Transform Data
MEAN(Temperature)
BY Month of Date, City
Design Encoding
Data
Data Fields
Transformed Data
Visualization

Visualization Always has a Context



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**

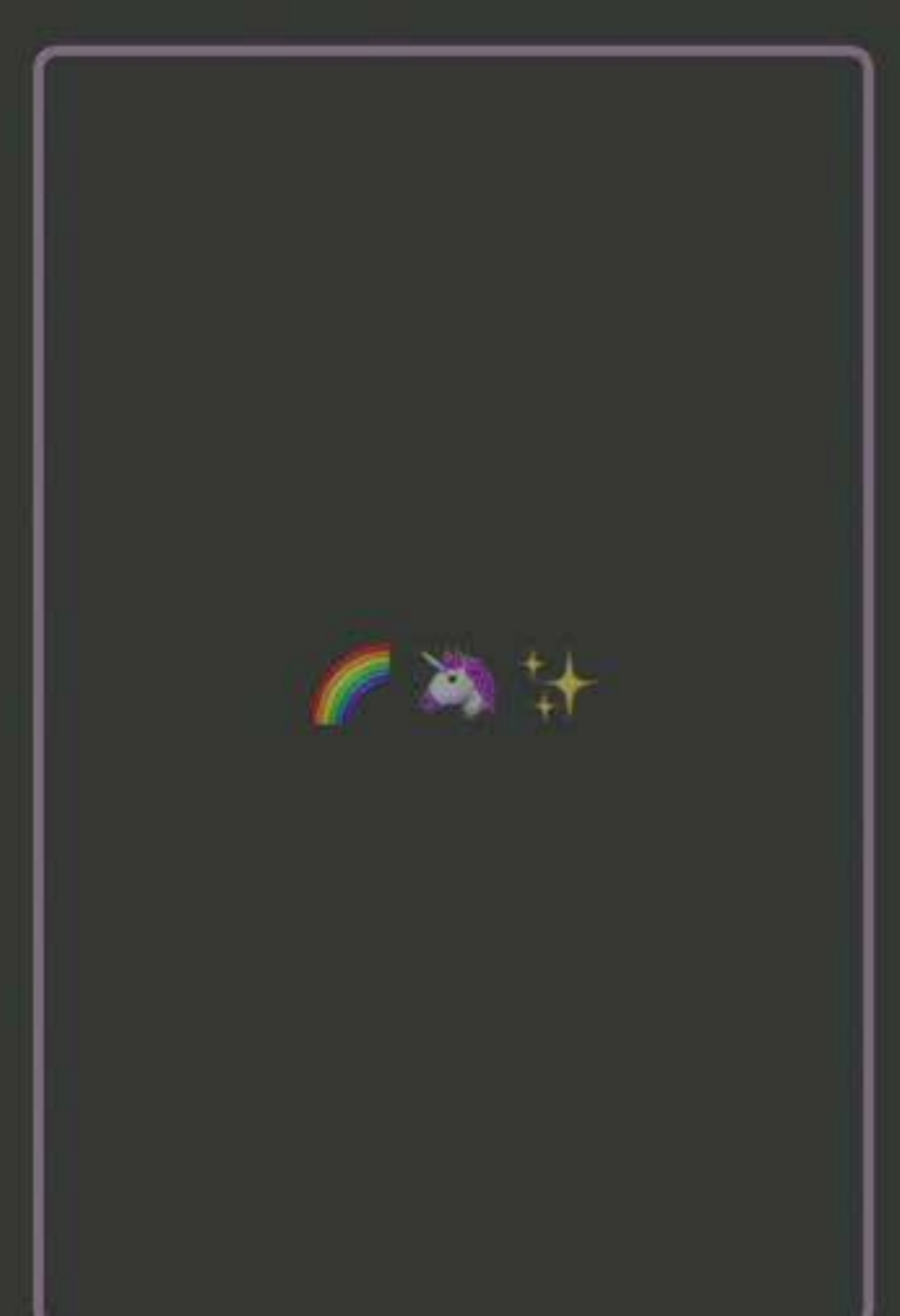
Formal model of visual encodings as sets of facts.

Design knowledge as constraints.

Draco

IEEE Infovis 2018. **Best Paper Award**

Formal model of visual encodings as sets of facts.



Draco: Design Knowledge as Constraints

Describe the domain of attributes.

e.g. mark type

"The ***mark*** of a chart should be one of ***bar***, ***line***, ***area*** or ***point***."

```
marktype(bar;line;area;point).
{ mark(M) : marktype(M) } = 1.
```

Attribute domains

Integrity

Preferences

Draco: Design Knowledge as Constraints

Describe the domain of attributes.

e.g. mark type, encoding type, aggregate, channels, binning, data types, tasks...

Attribute domains

Integrity

Preferences

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

Attribute domains

Integrity

Preferences

Draco: Design Knowledge as Constraints

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

Attribute domains

Integrity

Preferences

"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**.

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

total ~150 soft constraints

Attribute domains

Integrity

Preferences

Visualization Recommendation with Draco

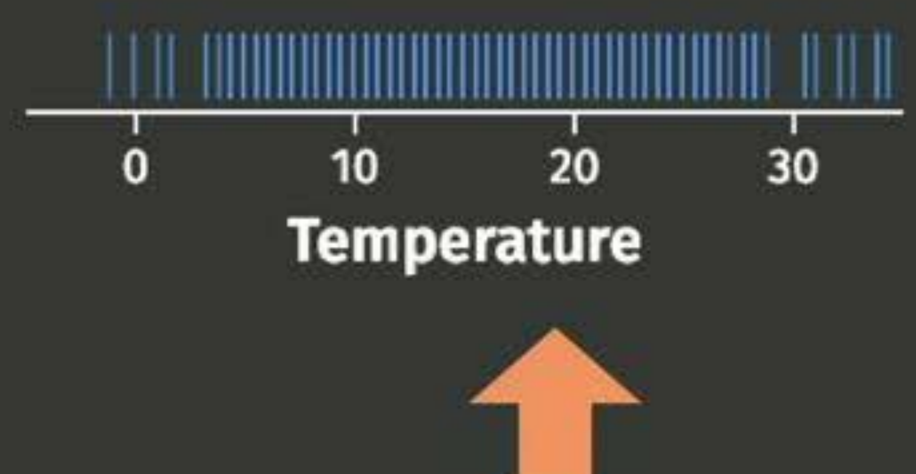
💡 Formulate visualization recommendation as finding optimal completions.

"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`."



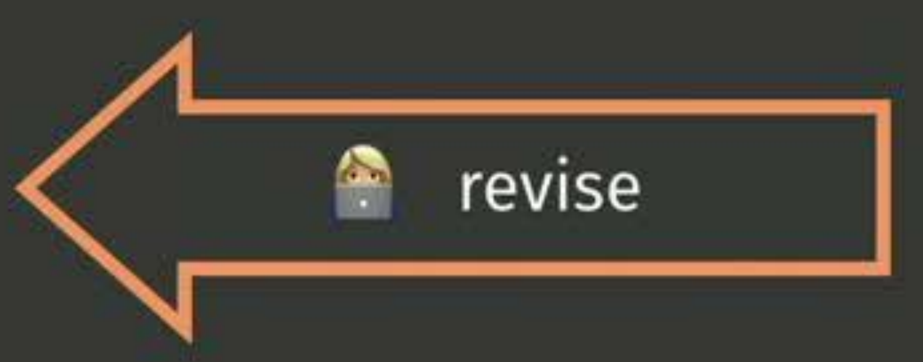
data("weather.csv").
encoding(e0).
field(e0,`temperature`).

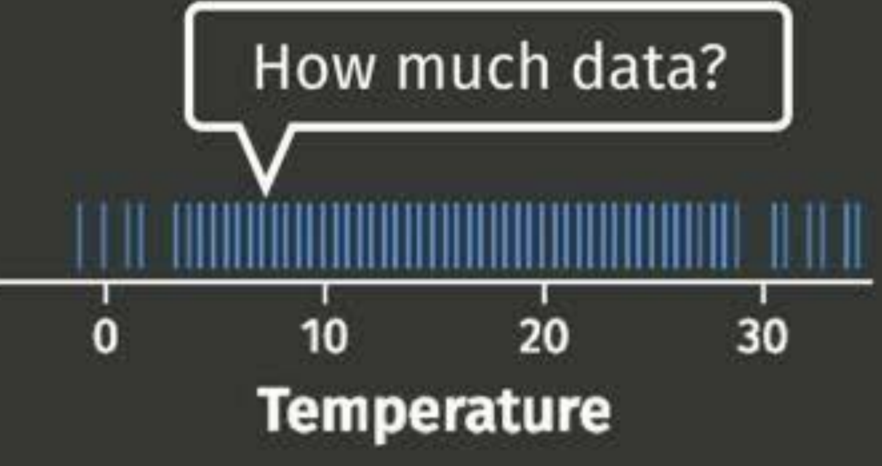
data("weather.csv").
mark(tick).
encoding(e0).
field(e0,`temperature`).
channel(e0,x).
type(e0,q).

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).
```

Attribute domains

Integrity

Preferences

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

Visualization Recommendation with Draco

"I want a visualization of the binned temperature."

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

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

⚠ Overlapping marks

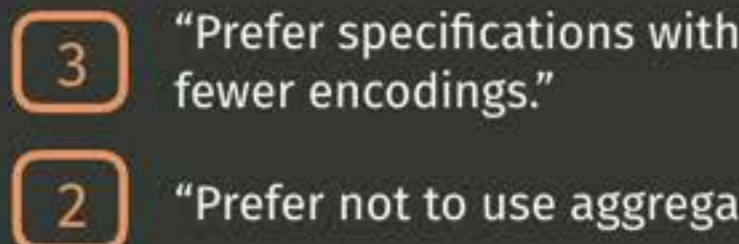
-5 0 5 10 15 20 25 30 35

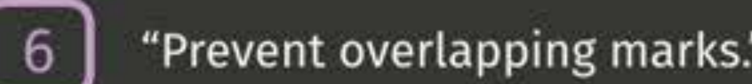
Binned Temperature

Visualization Recommendation with Draco

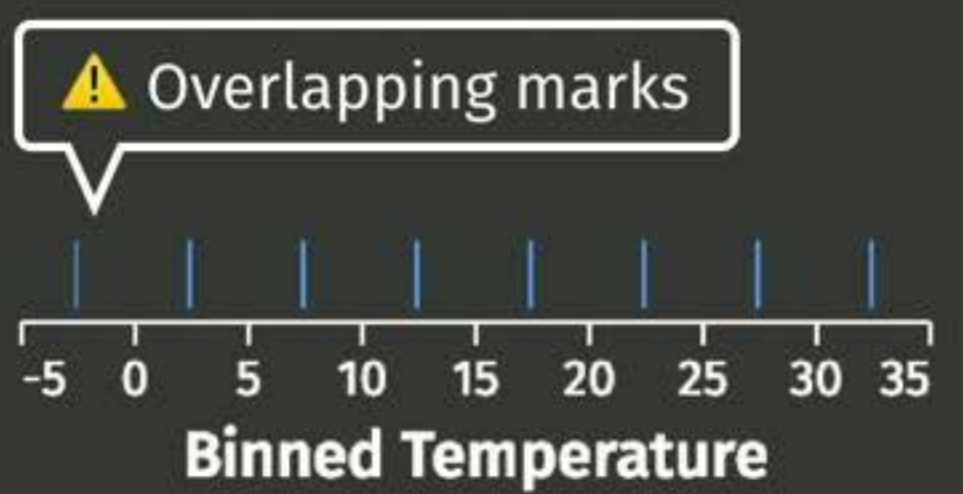
"I want a visualization of the binned temperature."

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





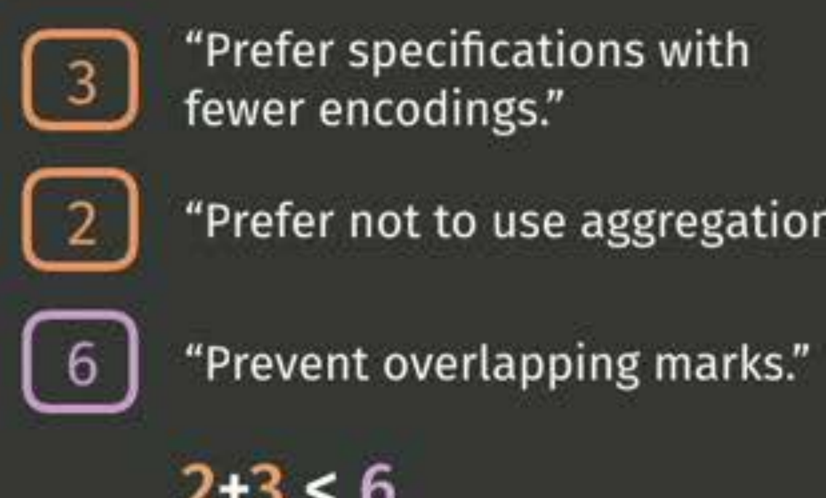
```
data("weather.csv").
mark(tick).
encoding(e0).
field(e0,temperature).
channel(e0,x).
type(e0,q).
bin(e0).
```



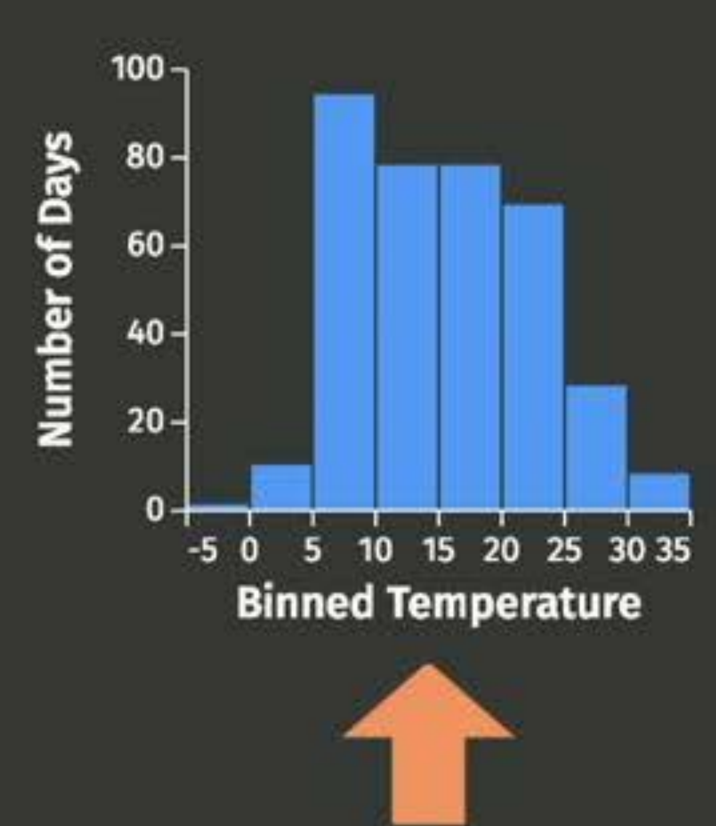
Visualization Recommendation with Draco

"I want a visualization of the binned temperature."

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



```
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).
```



3 "Prefer specifications with fewer encodings."

2 "Prefer not to use aggregation."

6 "Prevent overlapping marks."

Where do these weights come from?

"Graduate Student Descent"

Machine Learning

Draco

IEEE Infovis 2018. **Best Paper Award**

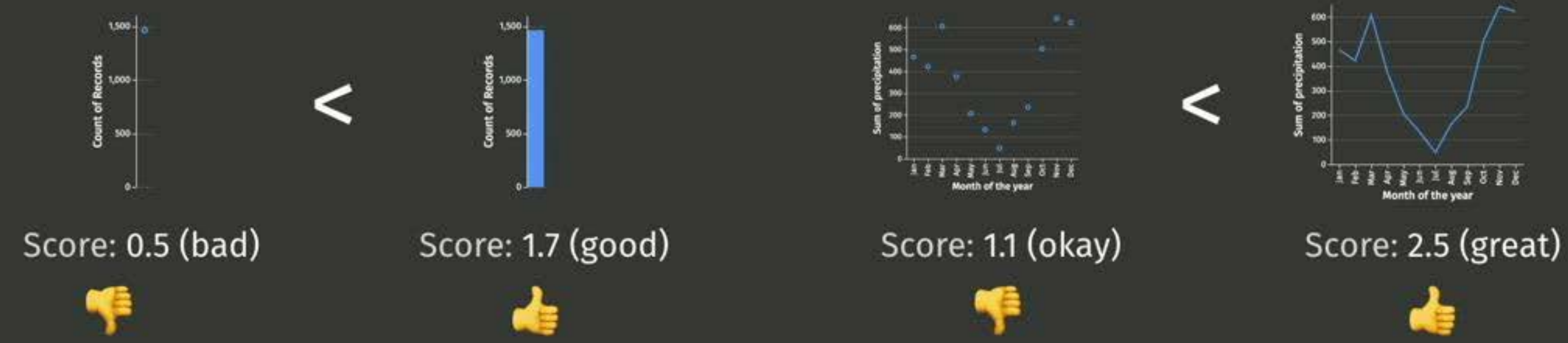
Formal model of visual encodings as sets of facts.

Methods to learn trade-offs from experimental data.

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



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

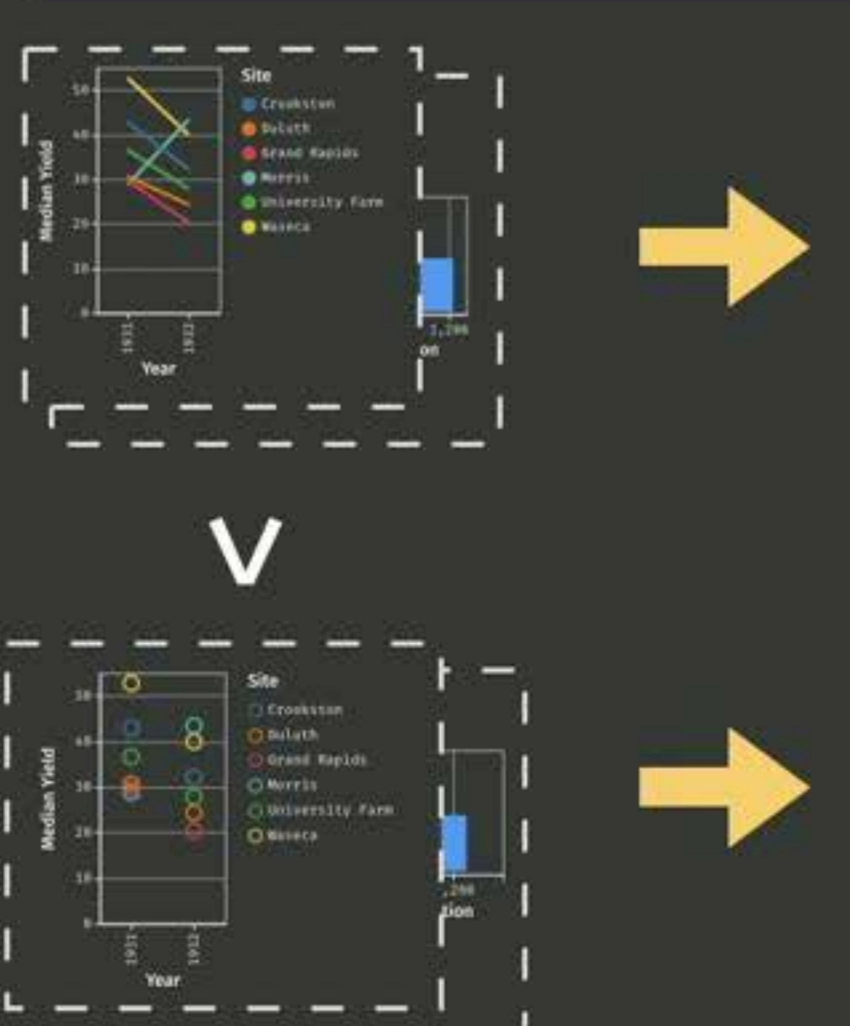
Draco: Learn Trade-Offs from Data

Training Data

Pairs of
Ranked Visualizations

Features

Violations of
Soft Constraints



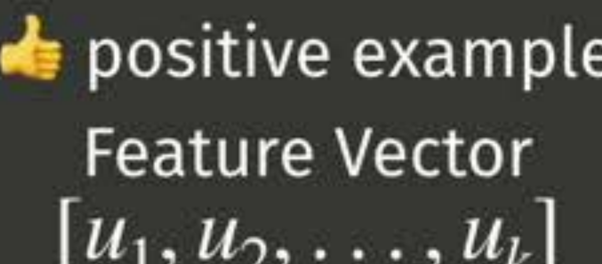
Draco: Learn Trade-Offs from Data

Training Data

Pairs of
Ranked Visualizations

Features

Violations of
Soft Constraints

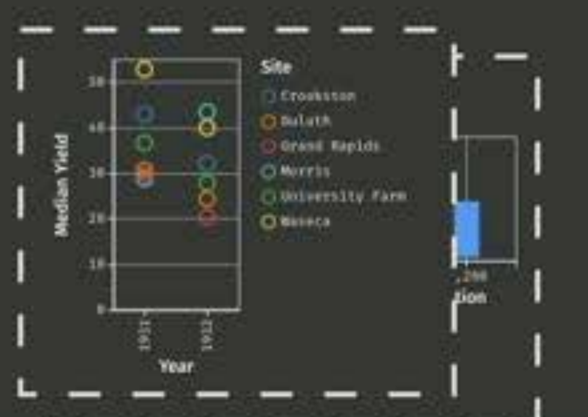


👍 positive example

Feature Vector

$$[u_1, u_2, \ldots, u_k]$$

>



👎 negative example

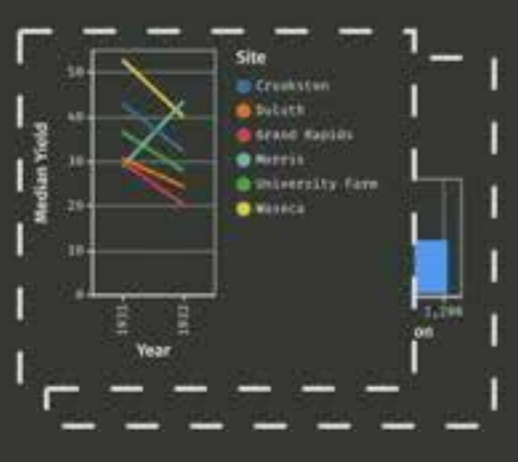
Feature Vector

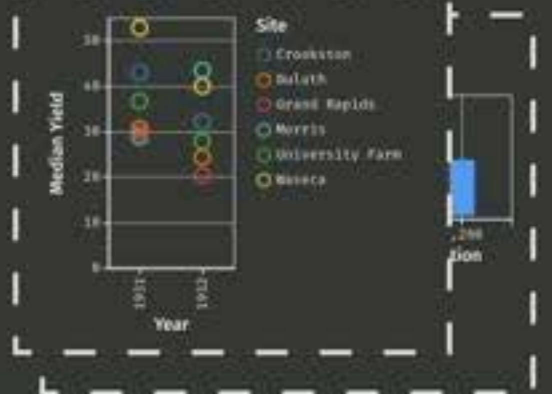
$$[v_1, v_2, \ldots, v_k]$$

v_i : the number of violations of constraint i

Draco: Learn Trade-Offs from Data

Training Data	Features	Learning Algorithm
Pairs of Ranked Visualizations	Violations of Soft Constraints	Learning to Rank with Linear SVM





👍 positive example

Feature Vector

$[u_1, u_2, \ldots, u_k]$

👎 negative example

Feature Vector

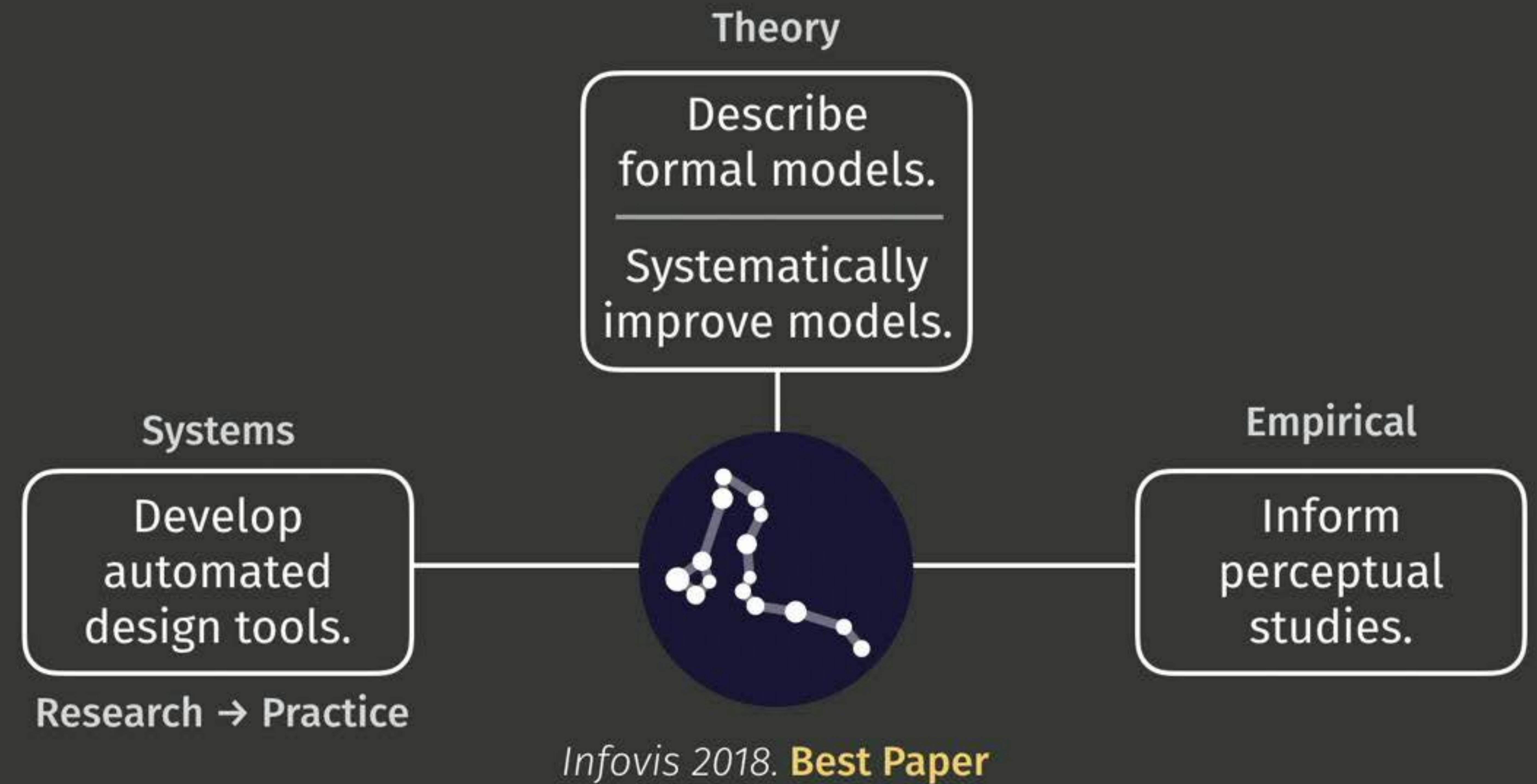
$[v_1, v_2, \ldots, v_k]$

v_i : the number of violations of constraint i

$$\arg\max_w \sum_{i \in 0 \ldots k} w_i (u_i - v_i)$$

w is the weight vector of the soft constraints

Draco at the Core of Visualization Research

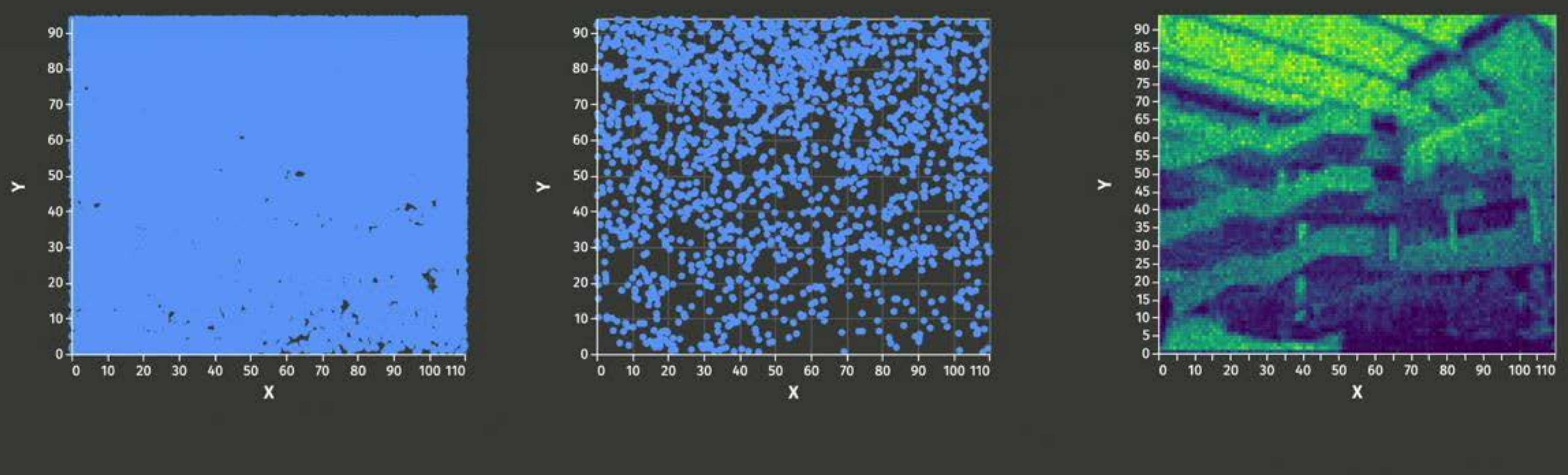


How do I create the next generation of visualization systems where users can **rapidly create good designs** regardless of the **scale of their data**?

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

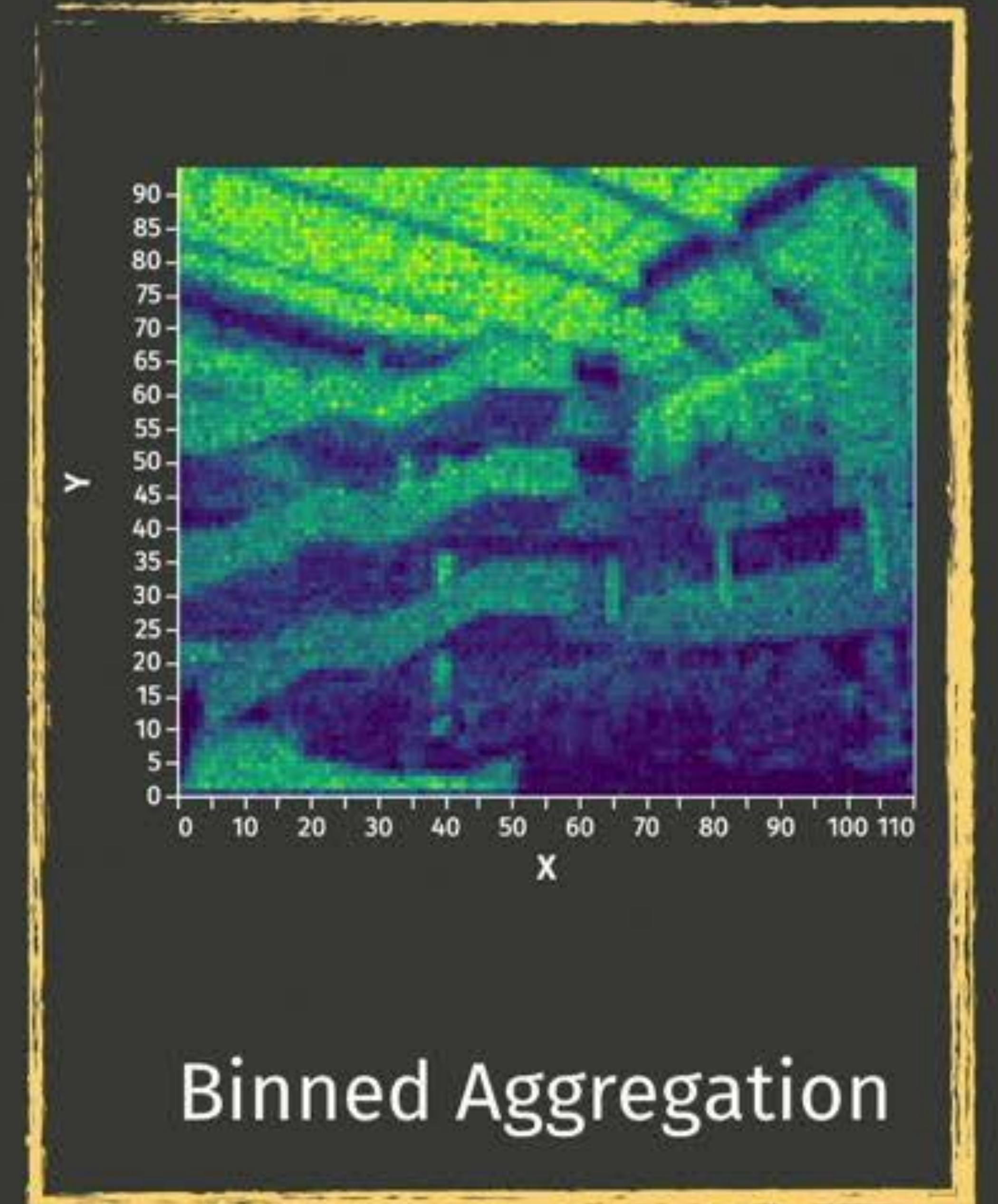
Sampling

Binned Aggregation

How to **Visualize** a Billion+ Records

Data

Sampling



Binned Aggregation

Decouple the visual complexity from the raw data through aggregation.

Arrival Delay in Minutes

Count

40,000,000
20,000,000
0

-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180

Departure Time

Count

10,000,000
5,000,000
0

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Distance in Miles

Count

40,000,000
20,000,000
0

0 400 800 1,200 1,600 2,000 2,400 2,800 3,200 3,600 4,000 4,400 4,800

Arrival Delay in Minutes

Count

40,000,000
20,000,000
0

-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180

Departure Time

Count

10,000,000
5,000,000
0

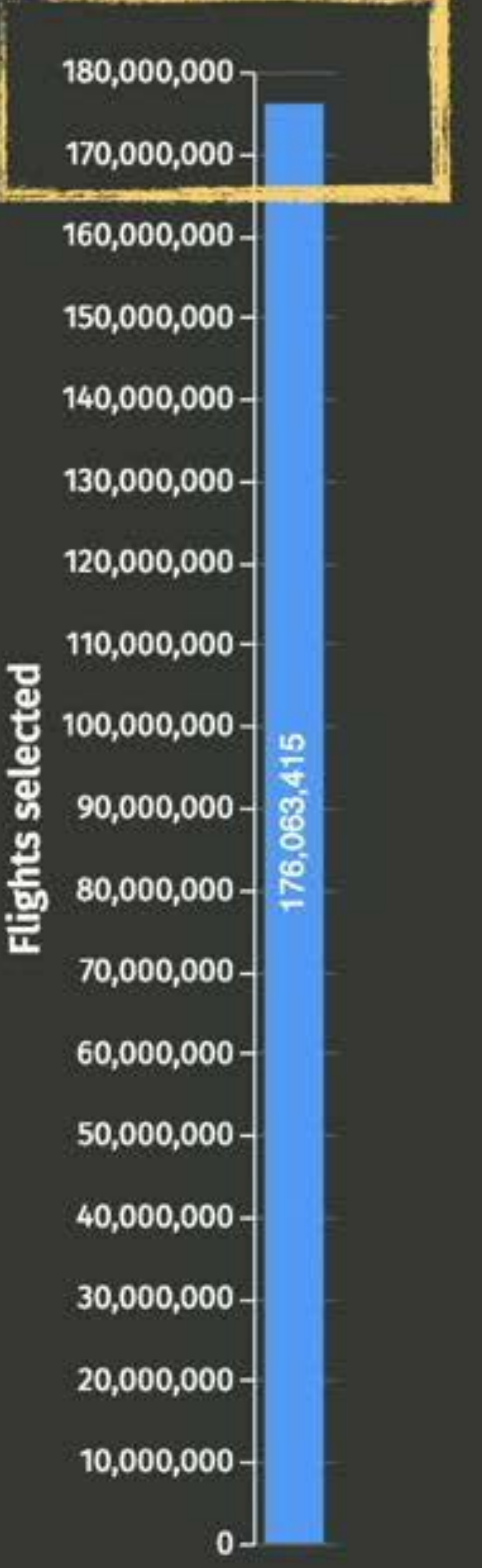
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Distance in Miles

Count

40,000,000
20,000,000
0

0 400 800 1,200 1,600 2,000 2,400 2,800 3,200 3,600 4,000 4,400 4,800



Arrival Delay in Minutes

Count

40,000,000
20,000,000
0

-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180

Departure Time

Count

10,000,000
5,000,000
0

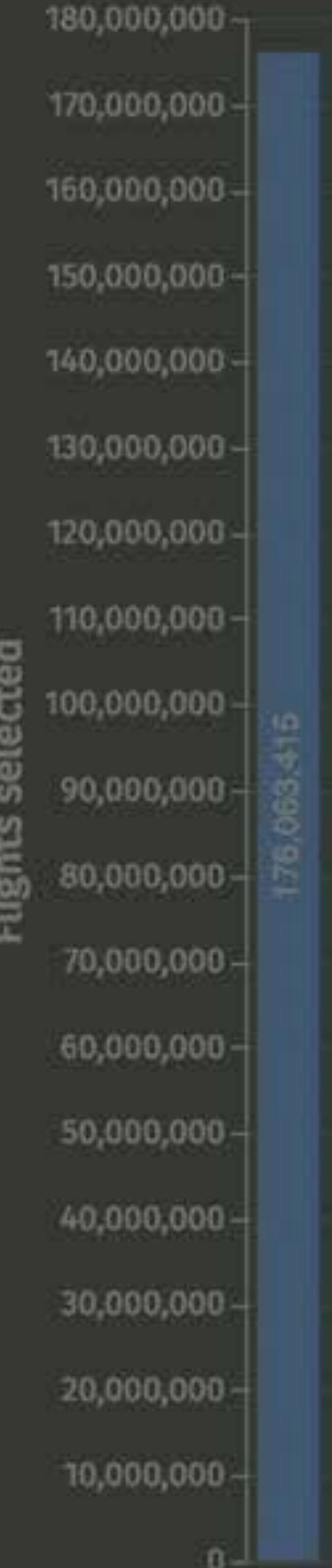
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Distance in Miles

Count

40,000,000
20,000,000
0

0 400 800 1,200 1,600 2,000 2,400 2,800 3,200 3,600 4,000 4,400 4,800



Arrival Delay in Minutes

Count

40,000,000
20,000,000
0

-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180

Departure Time

Count

10,000,000
5,000,000
0

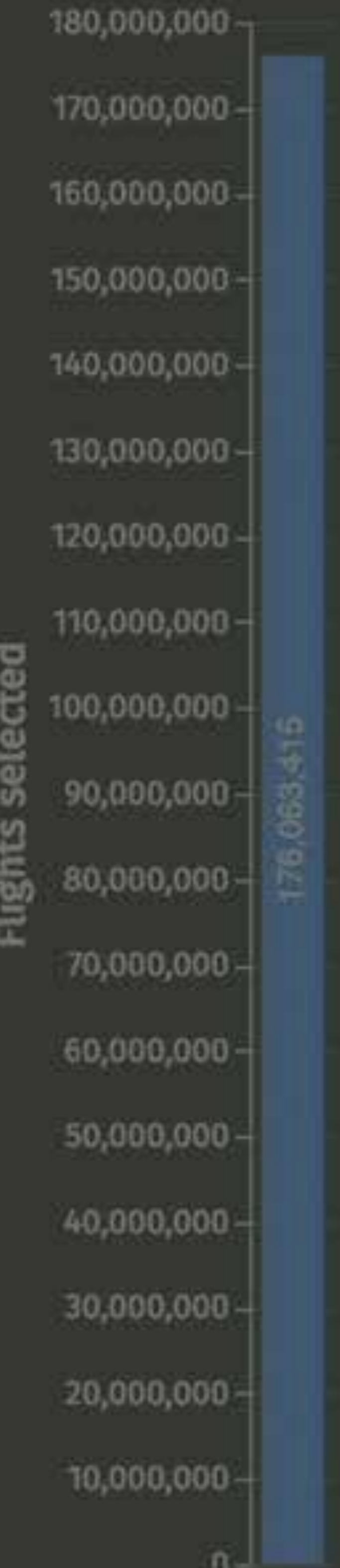
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Distance in Miles

Count

40,000,000
20,000,000
0

0 400 800 1,200 1,600 2,000 2,400 2,800 3,200 3,600 4,000 4,400 4,800



Arrival Delay in Minutes

Count

40,000,000
20,000,000
0

-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180

Departure Time

Count

10,000,000
5,000,000
0

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Distance in Miles

Count

40,000,000
20,000,000
0

0 400 800 1,200 1,600 2,000 2,400 2,800 3,200 3,600 4,000 4,400 4,800

Arrival Delay in Minutes

Count

40,000,000
20,000,000
0

-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180

Departure Time

Count

10,000,000
5,000,000
0

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Distance in Miles

Count

40,000,000
20,000,000
0

0 400 800 1,200 1,600 2,000 2,400 2,800 3,200 3,600 4,000 4,400 4,800

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.

The Effect of Interactive Latency. Liu, Heer. *IEEE Infovis 2014.*

Delays may bias analysts towards convenient data.

Falcon

uwdata.github.io/falcon

Arrival Delay in Minutes

Count

40,000,000
20,000,000
0

-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180

Departure Time

Count

10,000,000
5,000,000
0

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Distance in Miles

Count

40,000,000
20,000,000
0

0 400 800 1,200 1,600 2,000 2,400 2,800 3,200 3,600 4,000 4,400 4,800

Arrival Delay in Minutes

Count

40,000,000
20,000,000
0

-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180

Departure Time

Count

3,000,000
2,000,000
1,000,000
0

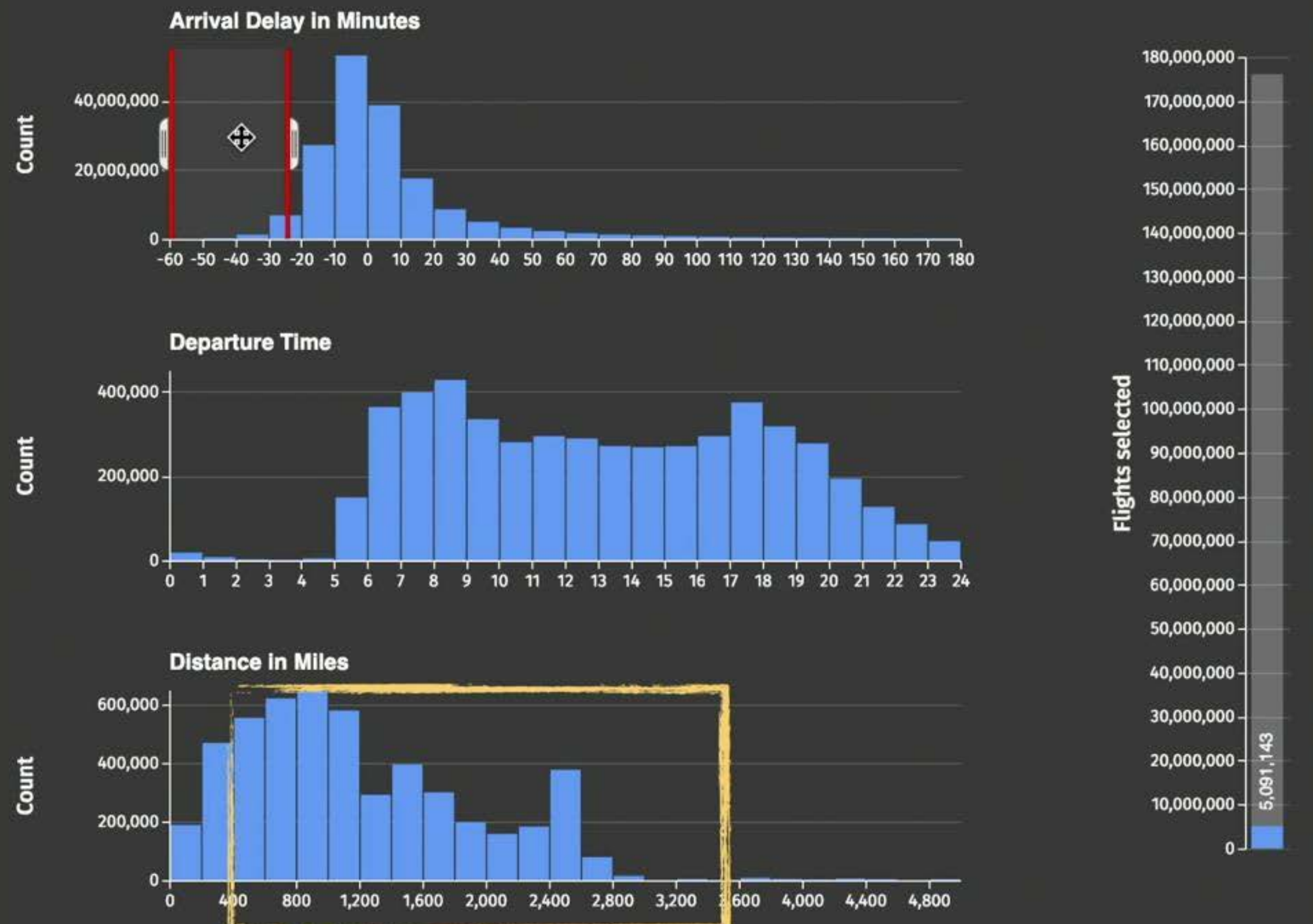
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

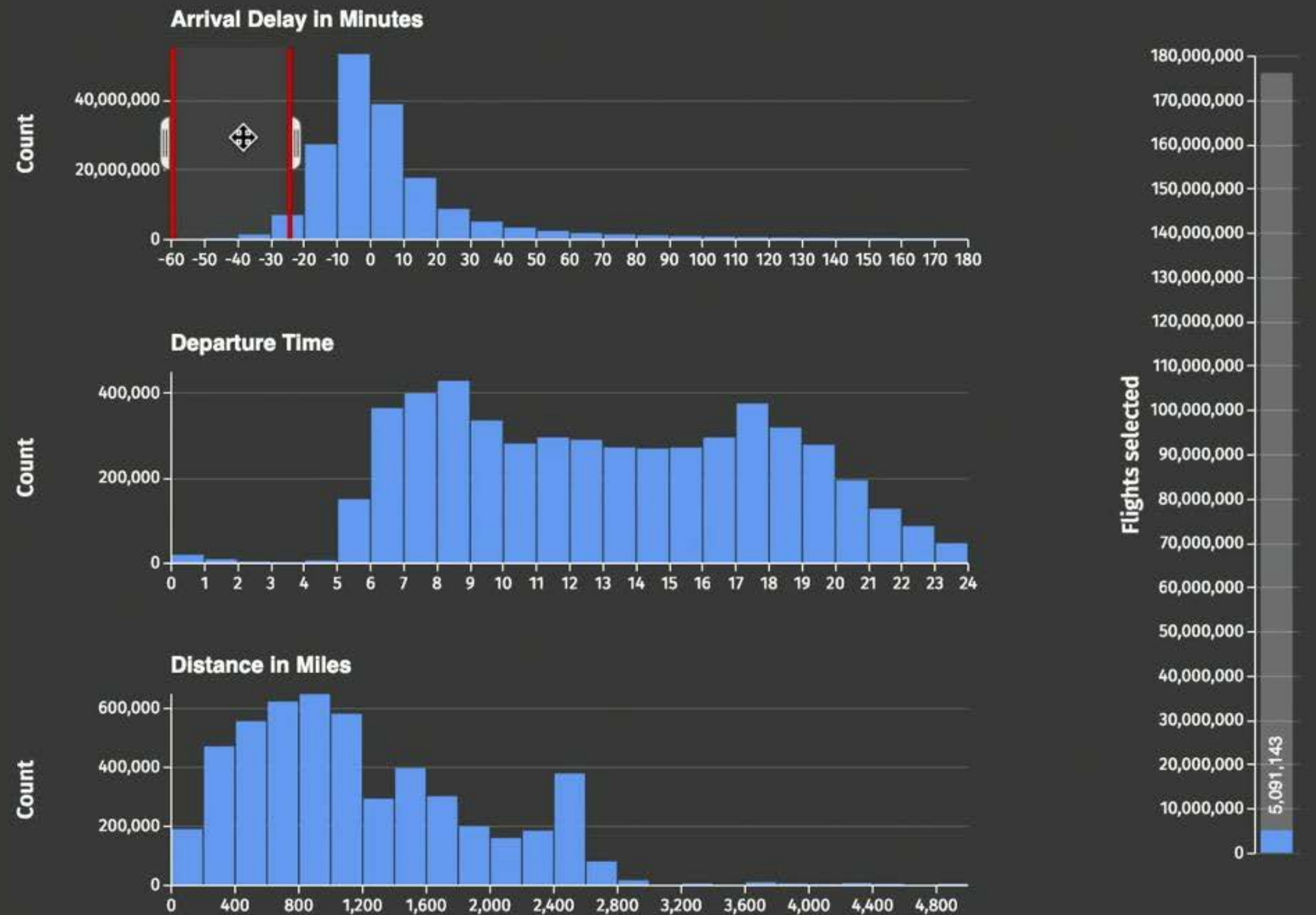
Distance in Miles

Count

10,000,000
5,000,000
0

0 400 800 1,200 1,600 2,000 2,400 2,800 3,200 3,600 4,000 4,400 4,800


Arrival Delay in Minutes
Count
40,000,000
20,000,000
0
-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180
Departure Time
Count
400,000
200,000
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Distance in Miles
Count
600,000
400,000
200,000
0
0 400 800 1,200 1,600 2,000 2,400 2,800 3,200 3,600 4,000 4,400 4,800
Flights selected
180,000,000
170,000,000
160,000,000
150,000,000
140,000,000
130,000,000
120,000,000
110,000,000
100,000,000
90,000,000
80,000,000
70,000,000
60,000,000
50,000,000
40,000,000
30,000,000
20,000,000
10,000,000
0
5,091,143



Arrival Delay in Minutes
Count
40,000,000
20,000,000
0
-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180
Departure Time
Count
400,000
200,000
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Distance in Miles
Count
600,000
400,000
200,000
0
0 400 800 1,200 1,600 2,000 2,400 2,800 3,200 3,600 4,000 4,400 4,800
Flights selected
180,000,000
170,000,000
160,000,000
150,000,000
140,000,000
130,000,000
120,000,000
110,000,000
100,000,000
90,000,000
80,000,000
70,000,000
60,000,000
50,000,000
40,000,000
30,000,000
20,000,000
10,000,000
0
5,091,143

Arrival Delay in Minutes
Count
40,000,000
20,000,000
0
-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180
Departure Time
80,000
60,000
40,000
20,000
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Distance in Miles
150,000
100,000
50,000
0
0 400 800 1,200 1,600 2,000 2,400 2,800 3,200 3,600 4,000 4,400 4,800
Flights selected
180,000,000
170,000,000
160,000,000
150,000,000
140,000,000
130,000,000
120,000,000
110,000,000
100,000,000
90,000,000
80,000,000
70,000,000
60,000,000
50,000,000
40,000,000
30,000,000
20,000,000
10,000,000
758,432

Arrival Delay in Minutes

Count

40,000,000
20,000,000
0

-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180

Departure Time

Count

80,000
60,000
40,000
20,000
0

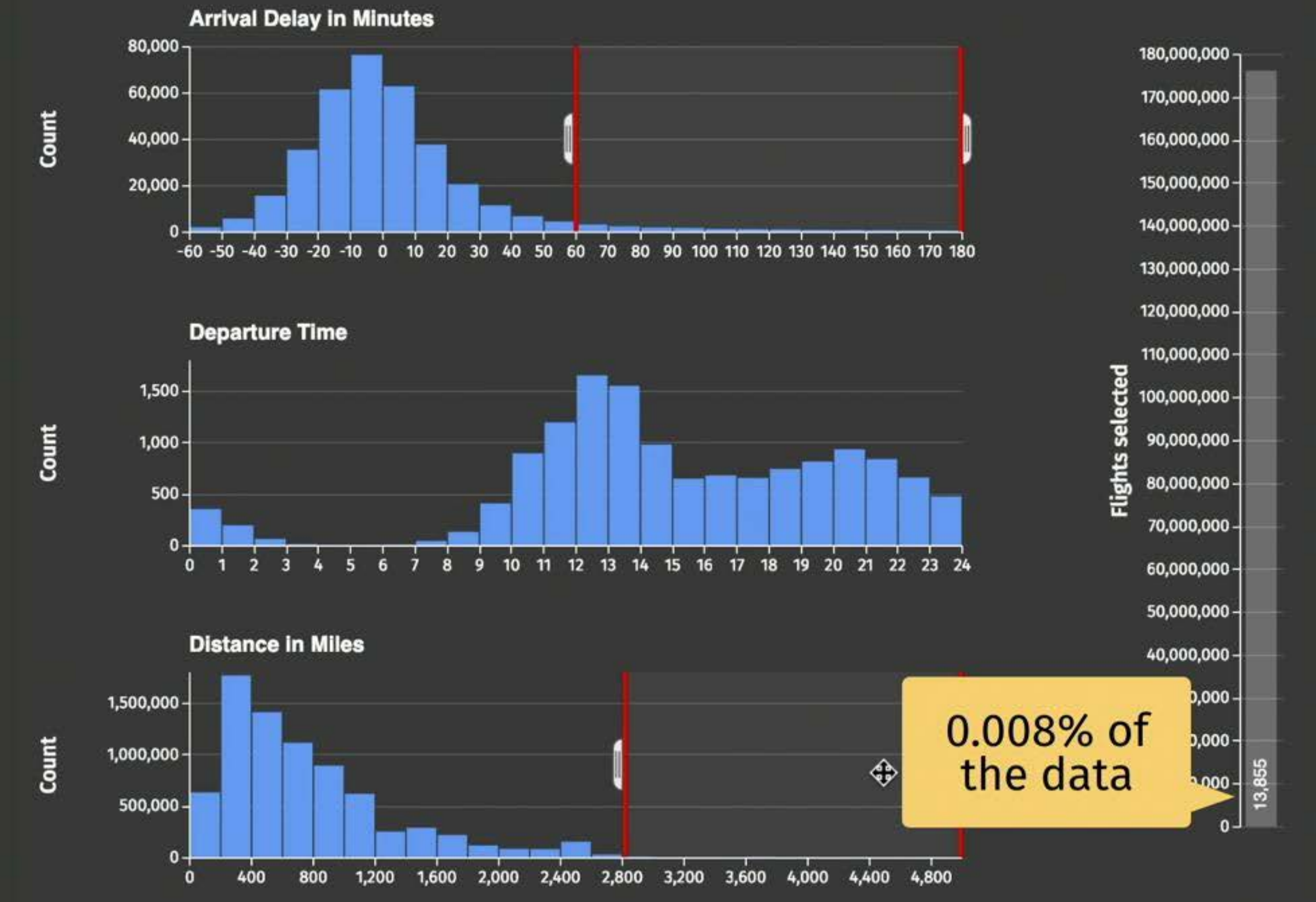
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Distance in Miles

Count

150,000
100,000
50,000
0

0 400 800 1,200 1,600 2,000 2,400 2,800 3,200 3,600 4,000 4,400 4,800



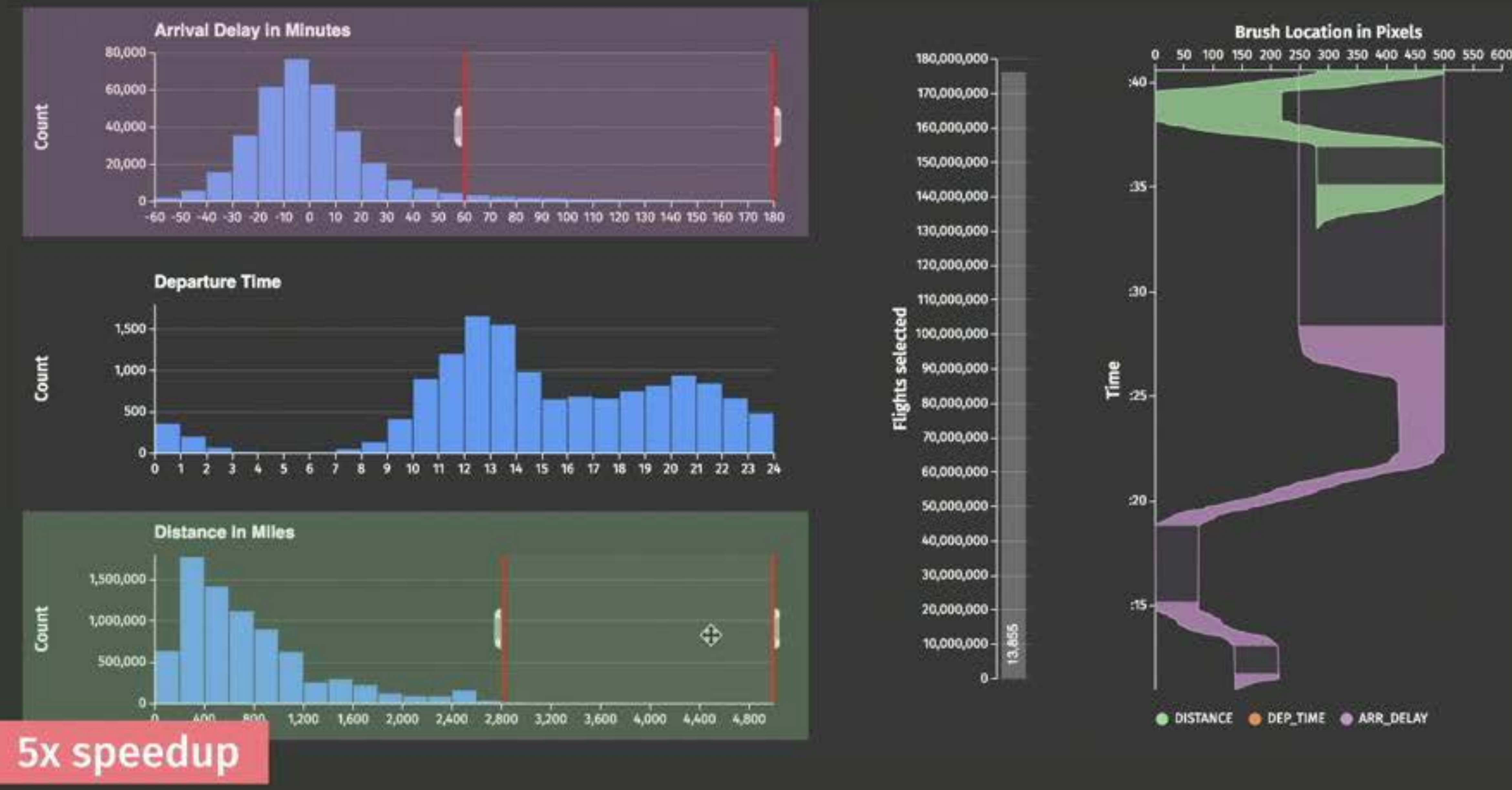
Arrival Delay in Minutes
Count
80,000
60,000
40,000
20,000
0
-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180
Departure Time
Count
1,500
1,000
500
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Distance in Miles
Count
1,500,000
1,000,000
500,000
0
0 400 800 1,200 1,600 2,000 2,400 2,800 3,200 3,600 4,000 4,400 4,800
Flights selected
180,000,000
170,000,000
160,000,000
150,000,000
140,000,000
130,000,000
120,000,000
110,000,000
100,000,000
90,000,000
80,000,000
70,000,000
60,000,000
50,000,000
40,000,000
0
13,855
0.008% of the data

How can Falcon be real-time?

Falcon Interaction Log

5x speedup

Falcon Interaction Log



5x speedup

Falcon Interaction Log

- 👁 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.

Arrival Delay in Minutes

Count

40,000,000
20,000,000
0

-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180

Departure Time

Count

10,000,000
5,000,000
0

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Distance in Miles

Count

40,000,000
20,000,000
0

0 400 800 1,200 1,600 2,000 2,400 2,800 3,200 3,600 4,000 4,400 4,800

brushes in the precomputed view

Arrival Delay in Minutes

Count

40,000,000
20,000,000
0

-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180

Departure Time

Count

10,000,000
5,000,000
0

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Distance in Miles

Count

40,000,000
20,000,000
0

0 400 800 1,200 1,600 2,000 2,400 2,800 3,200 3,600 4,000 4,400 4,800

brushes in the precomputed view

serves requests from a data cube

Data Cube. Gray et al. *1997.*

Arrival Delay in Minutes

Count

40,000,000
20,000,000
0

-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180

Departure Time

Count

10,000,000
5,000,000
0

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Distance in Miles

Count

40,000,000
20,000,000
0

0 400 800 1,200 1,600 2,000 2,400 2,800 3,200 3,600 4,000 4,400 4,800

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

Arrival Delay in Minutes

Count

40,000,000
20,000,000
0

Loading View...

-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180

Departure Time

Count

10,000,000
5,000,000
0

Loading View...

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Distance in Miles

Count

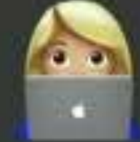
40,000,000
20,000,000
0

0 400 800 1,200 1,600 2,000 2,400 2,800 3,200 3,600 4,000 4,400 4,800

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

Arrival Delay in Minutes

Count

40,000,000
20,000,000
0

-60 -50 -40 -30 -20 -10 0 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180

Departure Time

Count

10,000,000
5,000,000
0

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24

Distance in Miles

Count

40,000,000
20,000,000
0

0 400 800 1,200 1,600 2,000 2,400 2,800 3,200 3,600 4,000 4,400 4,800

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

Constant data & time.
Client only.

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

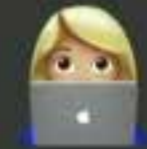
Constant data & time.
Client only.

brushes in the precomputed view

serves requests from a data cube

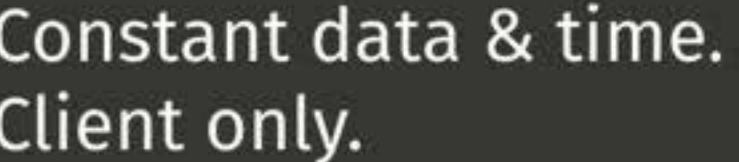
Data Cube. Gray et al. *1997.*

💡 Aggregation decouples interactions from queries over the raw data.



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

Data Cube. Gray et al. *1997.*

💡 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.

imMens. Liu et al. *Eurovis 2013.*

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.*

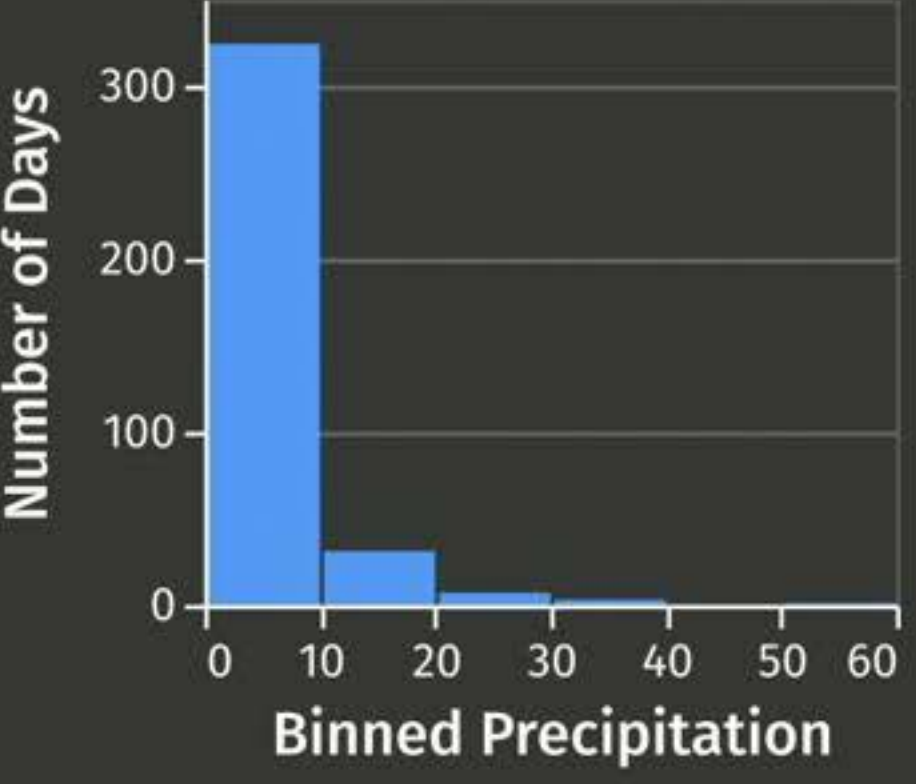
Small cubes for single active view.
Small cubes are built on the fly. View switches require new cube.

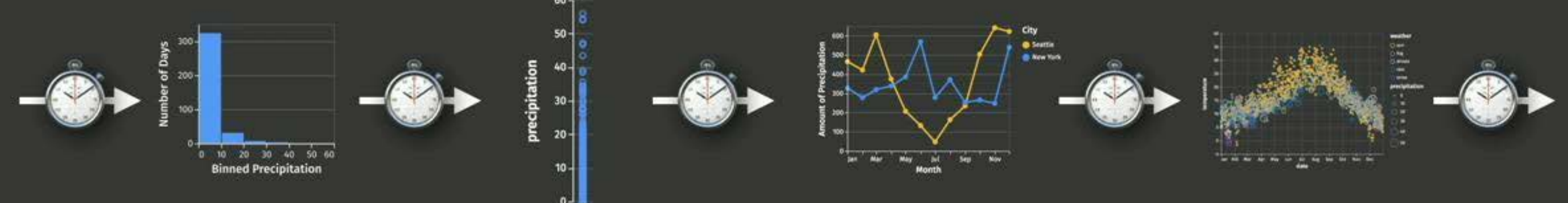
Explore GAIA in Falcon with M
localhost:1234
Magnitude
Hide Unfiltered
Count
Parallax
Reset Brush
Dec
Ra
Mg
G_BP - G_RP
Stars selected
1,021,039,816
Radial Velocity
1.7 B stars.
1.2 TB of data.
Visualizations running in my browser.
Data stored in OmniSci database.
Powered by Falcon 0.13.0

"With Falcon it feels like I'm really interacting with my data."

Data Platform Engineer at Stitch Fix

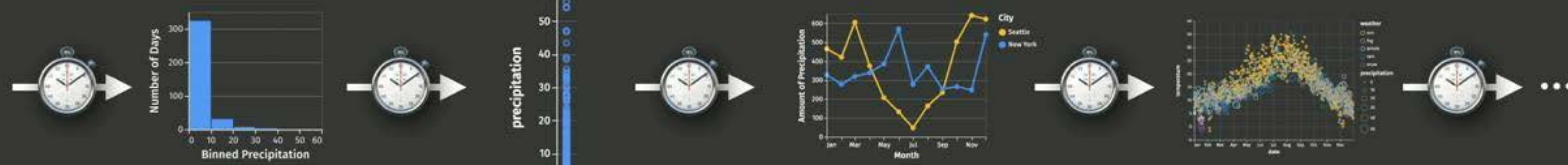
What if data too large to even query it in a reasonable time?


Number of Days
300
200
100
0
0
10
20
30
40
50
60
Binned Precipitation

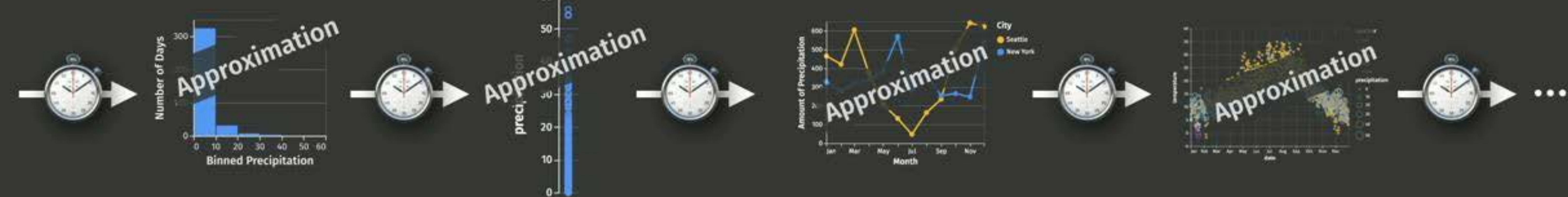


Latencies reduce engagement
and lead to fewer observations.

The Effect of Interactive Latency. Liu, Heer. *IEEE Infovis 2014.*

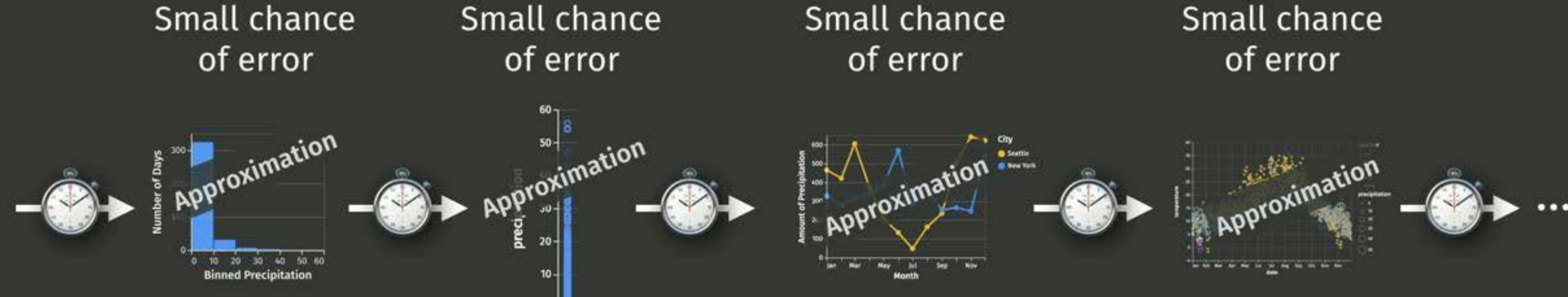


Number of Days
Binned Precipitation
precipitation
Amount of Precipitation
Month
City
Seattle
New York



Approximation. Accuracy → Speed

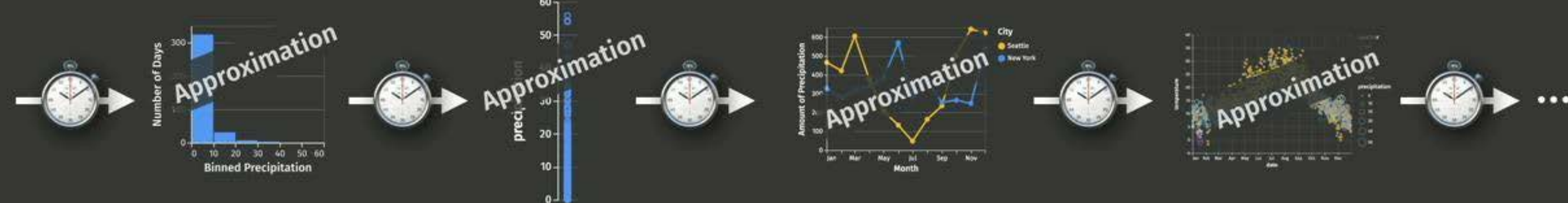
Approximate query processing (AQP)
Uncertainty estimation in statistics
Uncertainty visualization
Probabilistic programming
Approximate hardware



Approximation. Accuracy → Speed

Approximate query processing (AQP)
Uncertainty estimation in statistics
Uncertainty visualization
Probabilistic programming
Approximate hardware

Very likely to have at least one error



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.

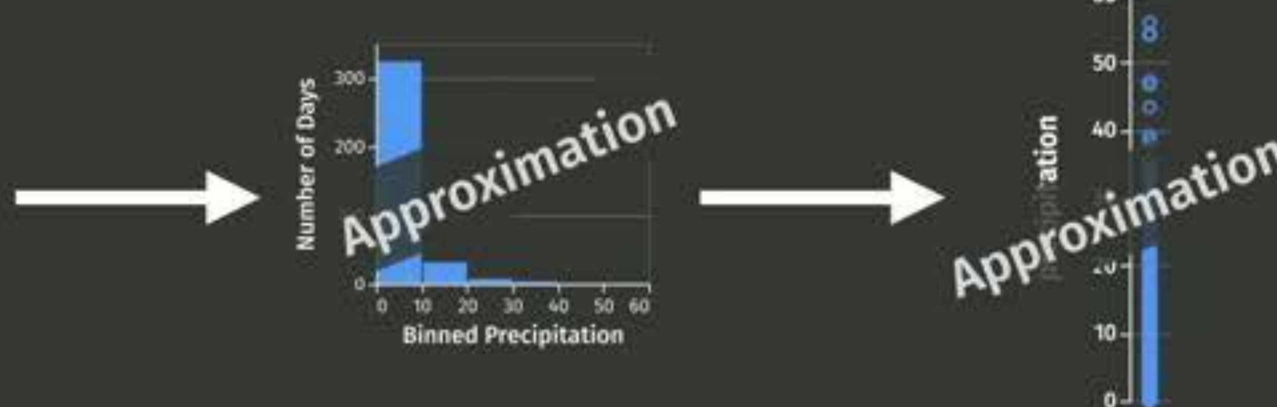
Optimistic Visualization

Trust but Verify

What if we think of the issues with approximation as user-experience problems?

Optimistic Visualization

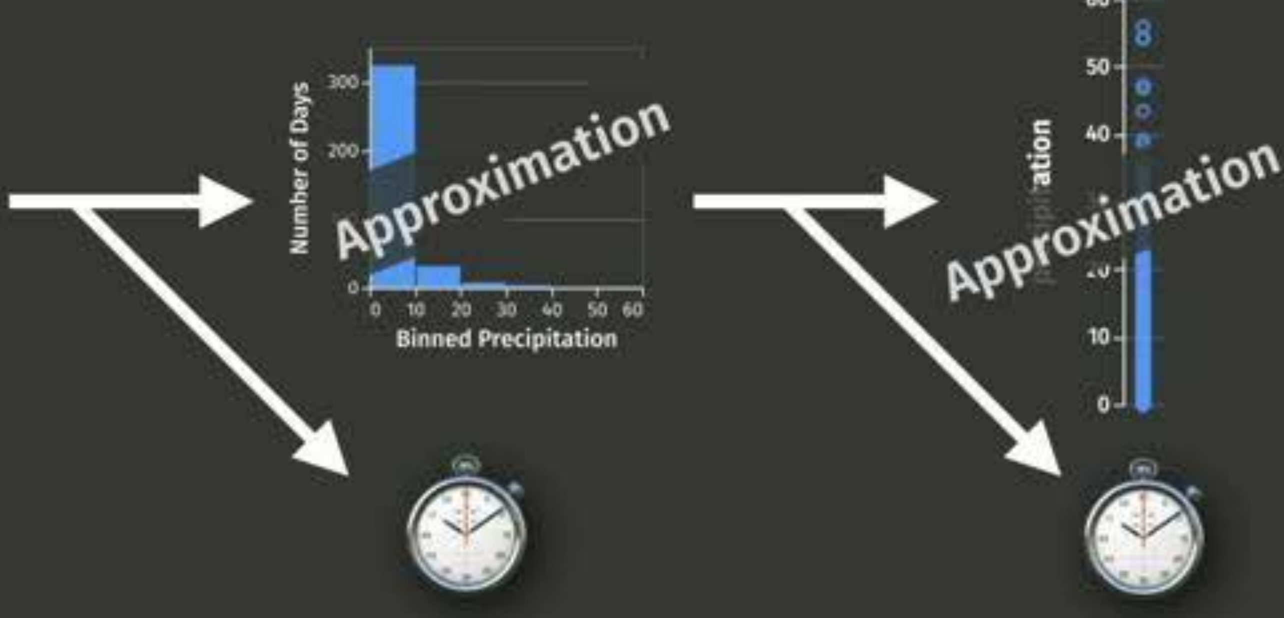
Trust but Verify. Moritz et al. *CHI 2017.*



1. Analysts uses initial estimates.

Optimistic Visualization

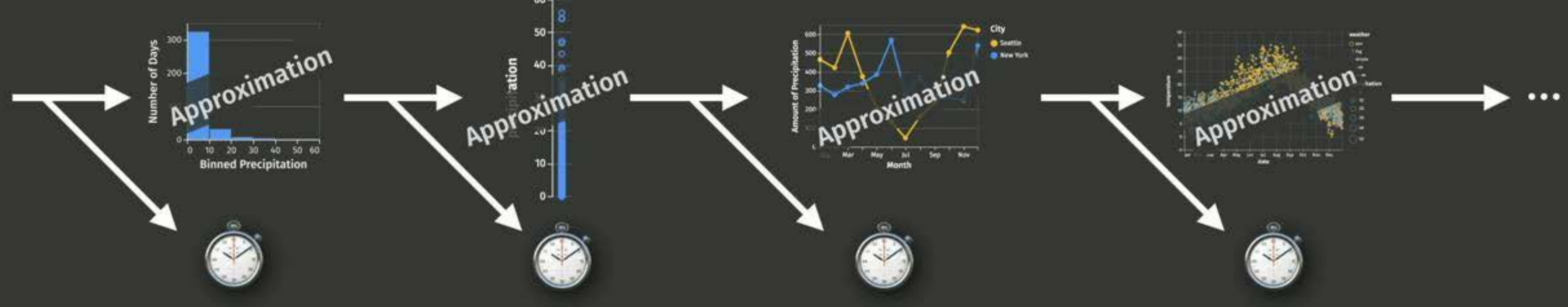
Trust but Verify. Moritz et al. *CHI 2017.*



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

Optimistic Visualization

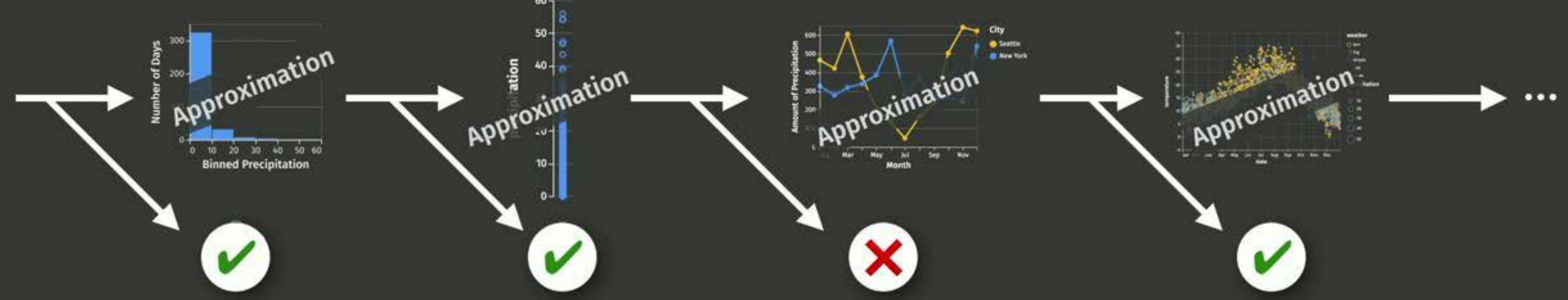
Trust but Verify. Moritz et al. *CHI 2017.*



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

Optimistic Visualization

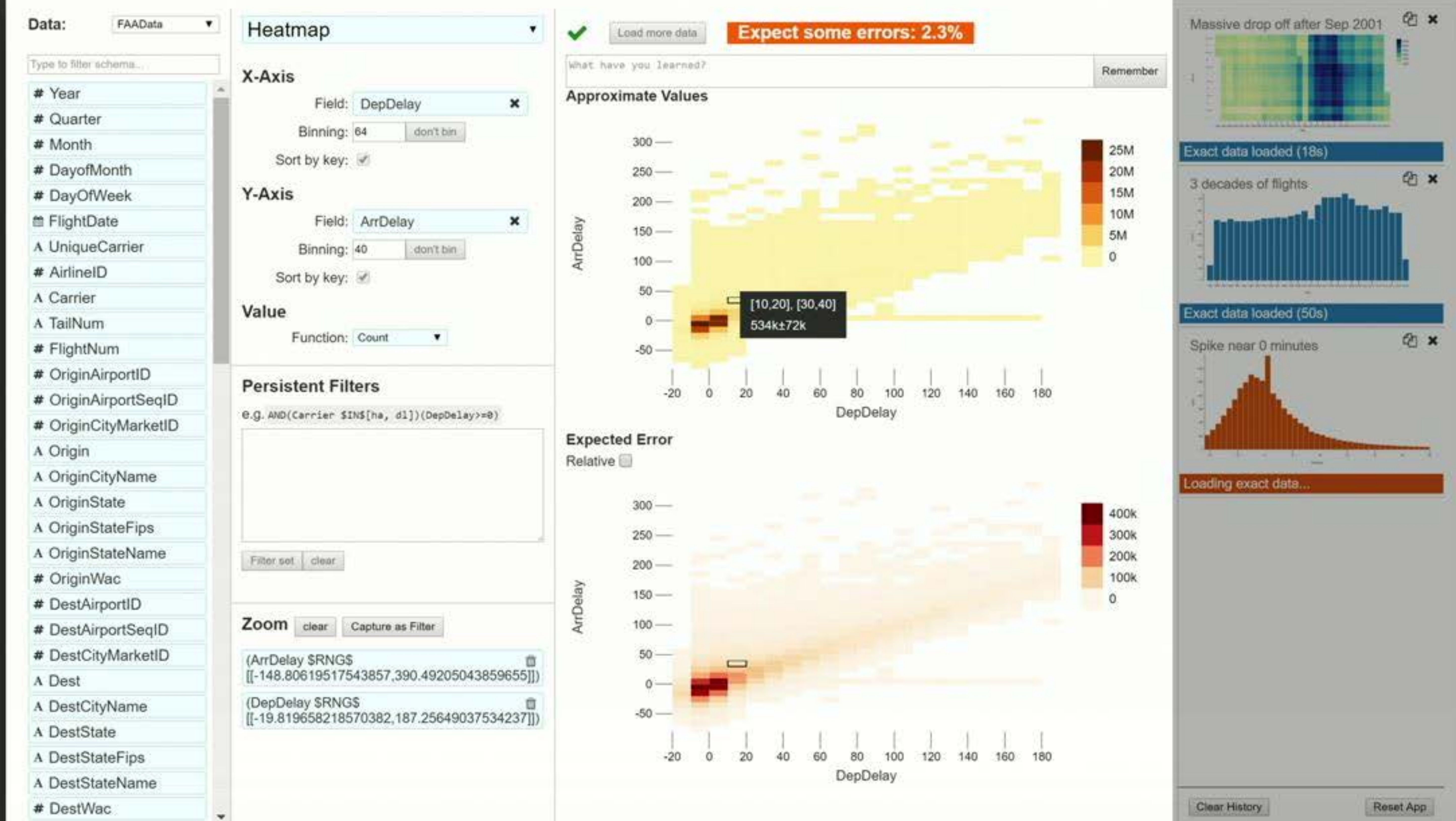
Trust but Verify. Moritz et al. *CHI 2017.*



1. Analysts uses initial estimates.
2. Precise queries run in the background.
3. System confirms results. Analyst detects errors.

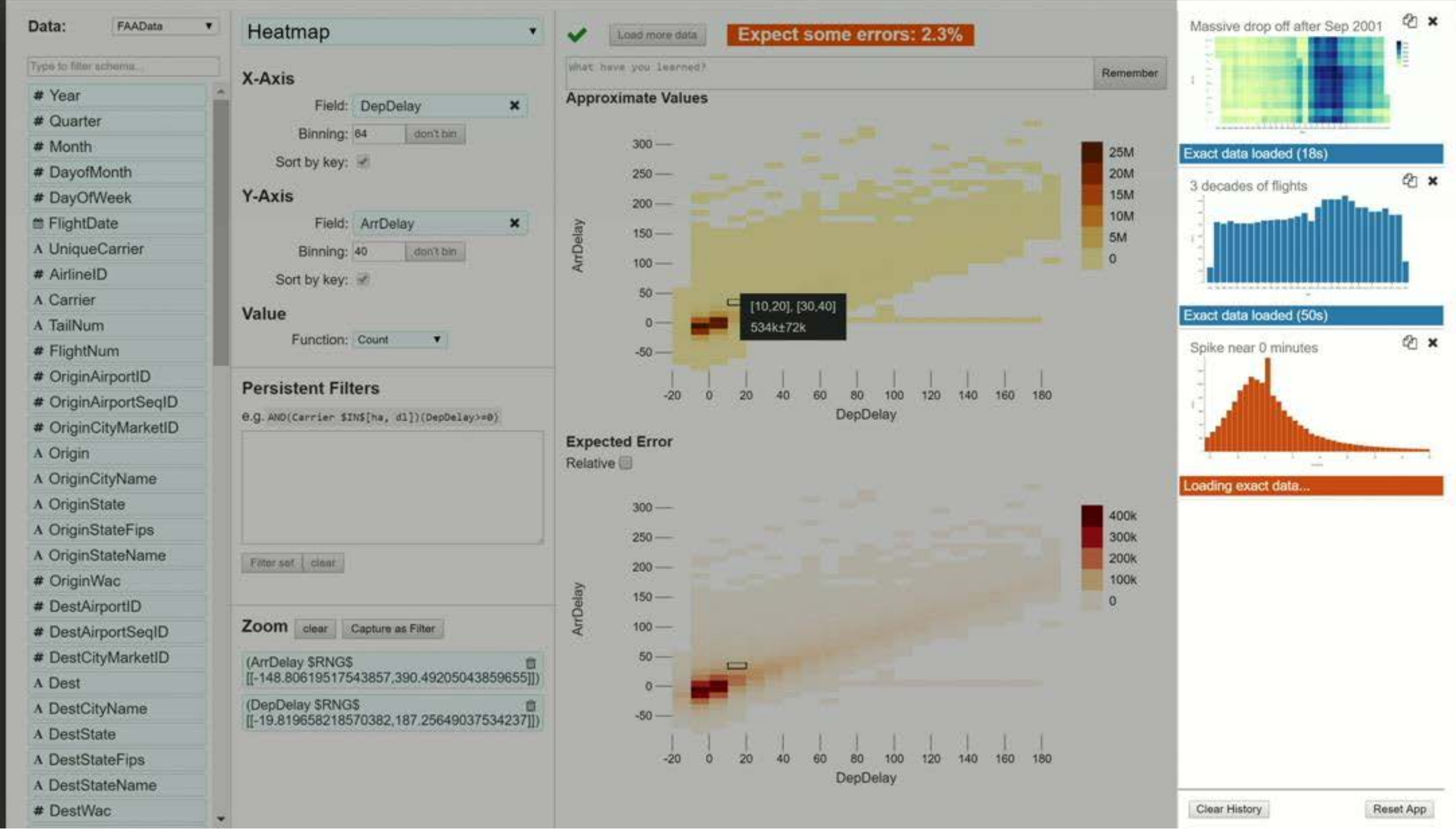
Analysts can use approximations and also trust them.

Pangloss Implements Optimistic Visualization

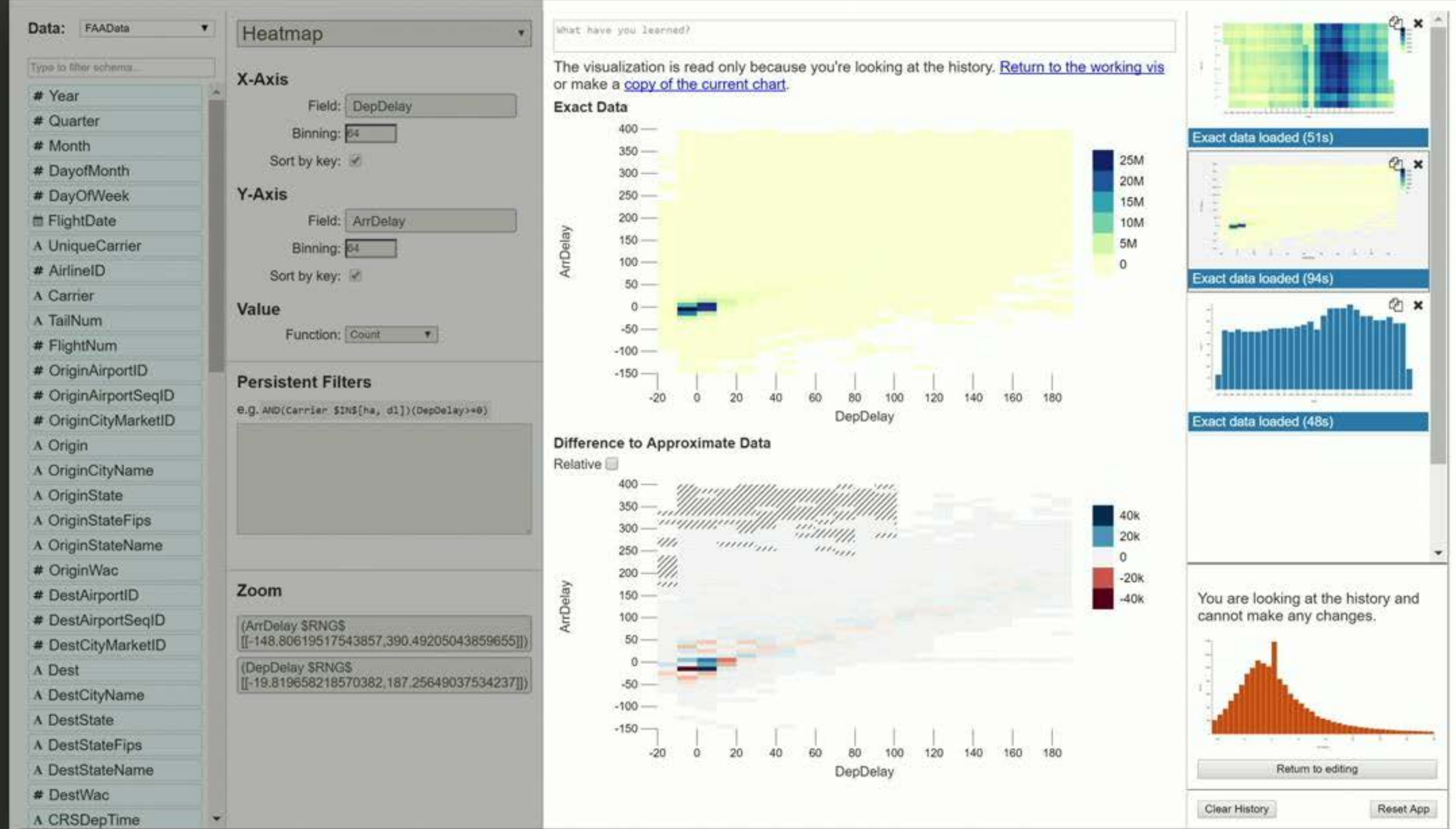


Pangloss Visualizes Uncertainty

Pangloss shows a History of Previous Charts



In Pangloss, Analysts can Confirm results



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

Draco *Infovis 2018.* **Best Paper**

Formal reasoning for visualization design

Scalable Visualization

Falcon *CHI 2019.*

Real-time linked interactions with billions of records

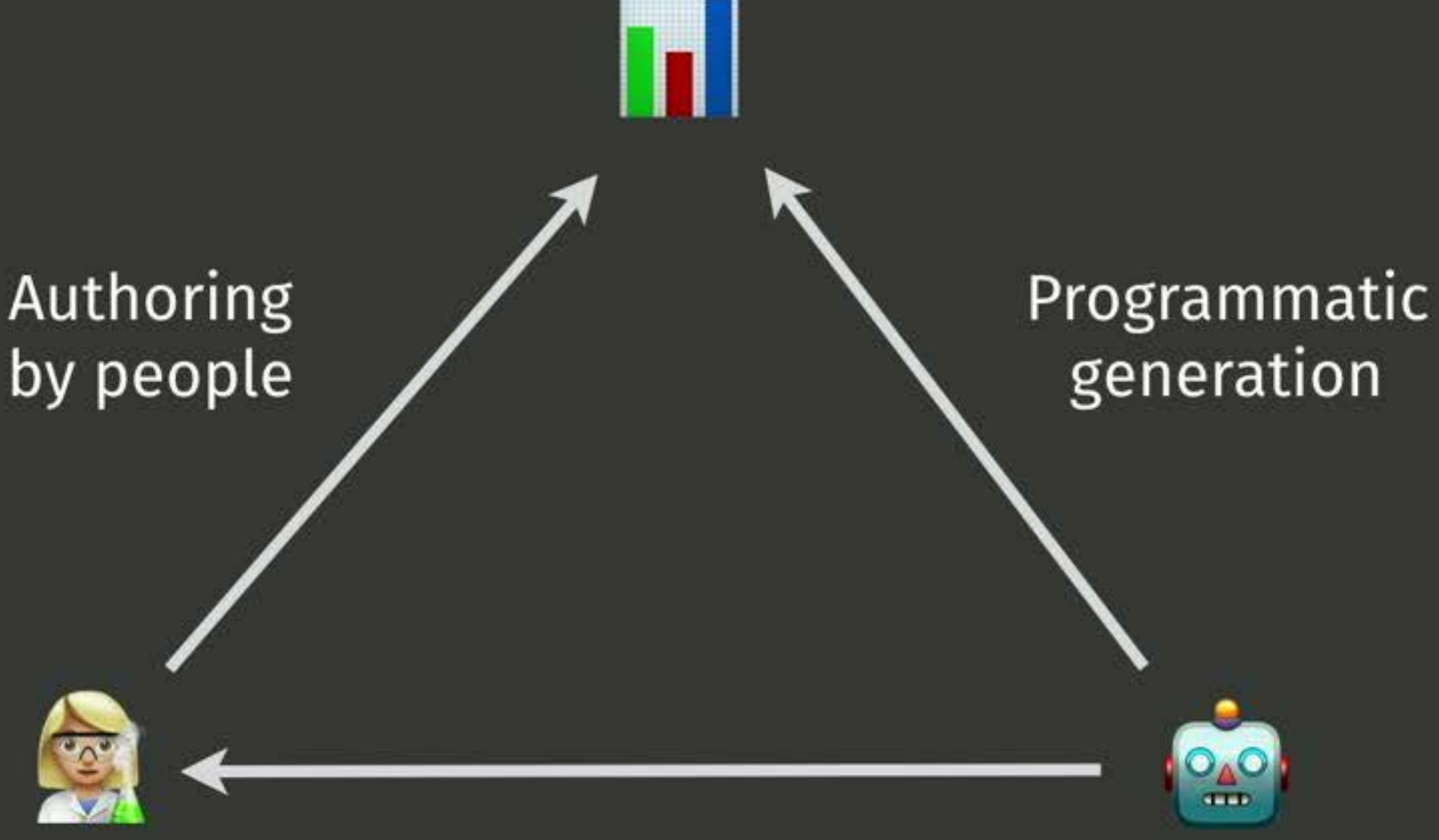
Optimistic Visualization *CHI 2017.*

Fast and reliable approximations for data exploration

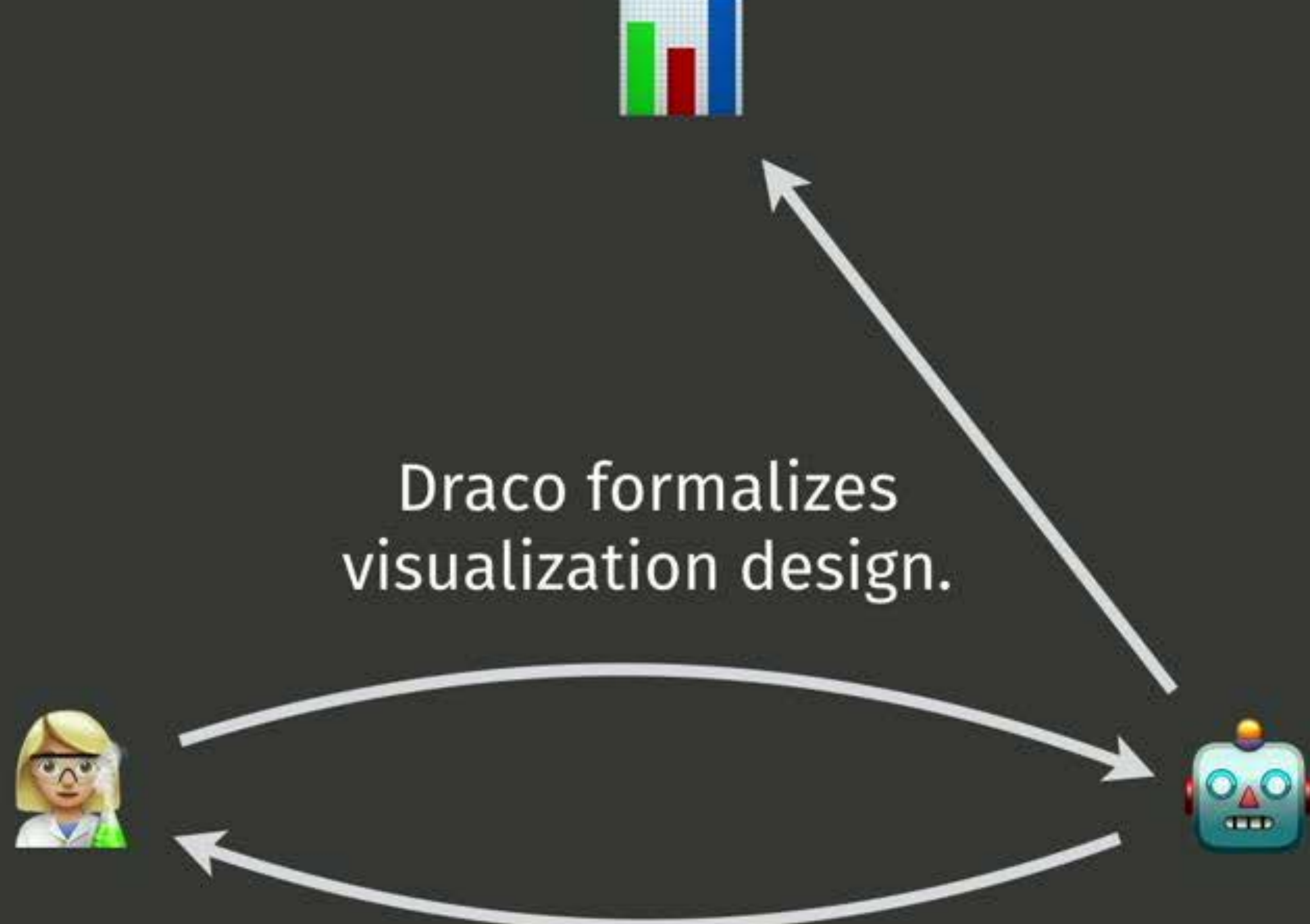
My Mission:

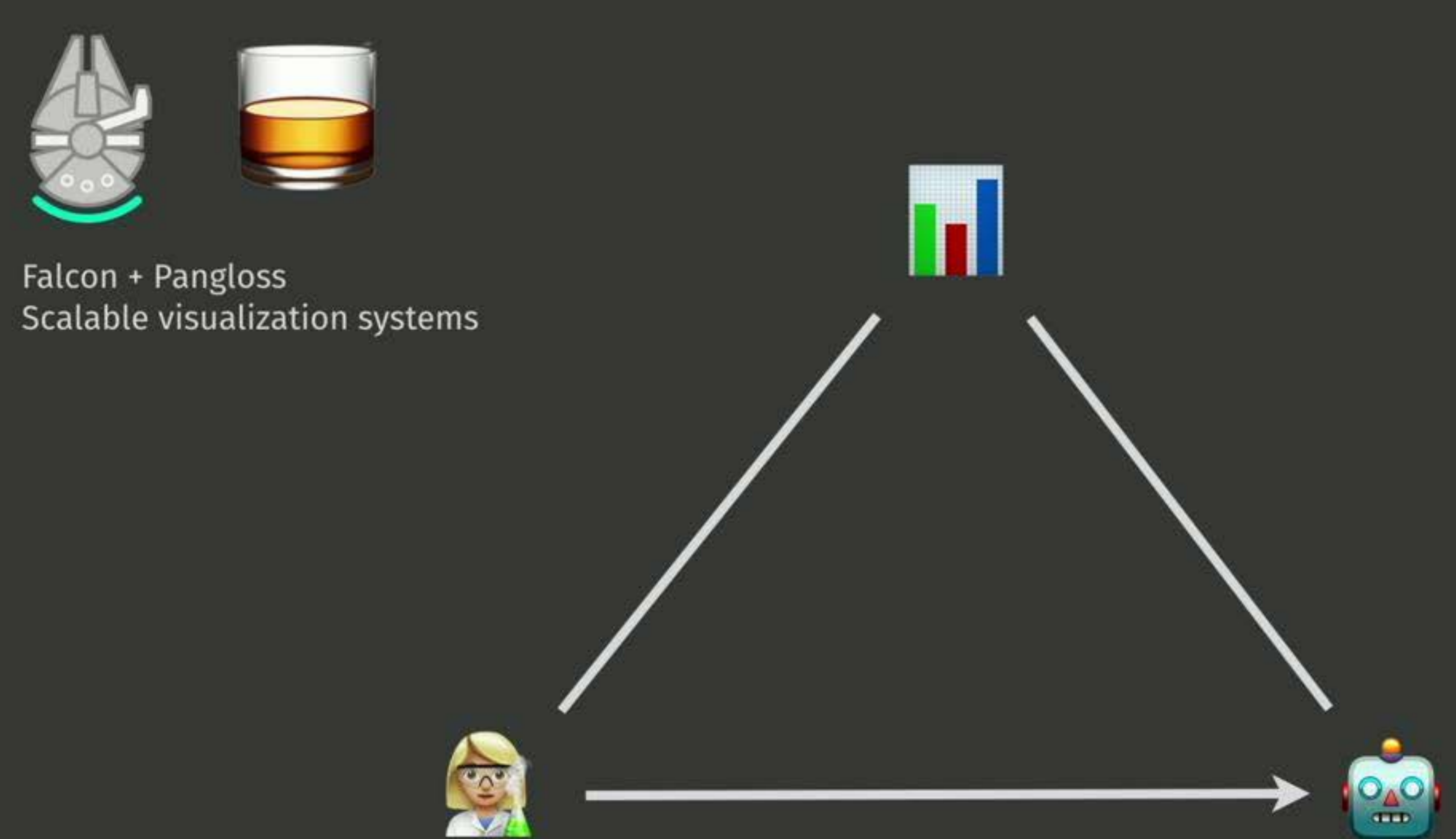
Develop **tools for data analysis and communication** that richly integrate the strengths of both **people** and **machines**.

Vega-Lite
High-level visualization language



Draco
Model of visualization design





Understanding how 👩‍🔬 interact with visualizations enabled new 🤖 optimizations.

Challenge for the Future:

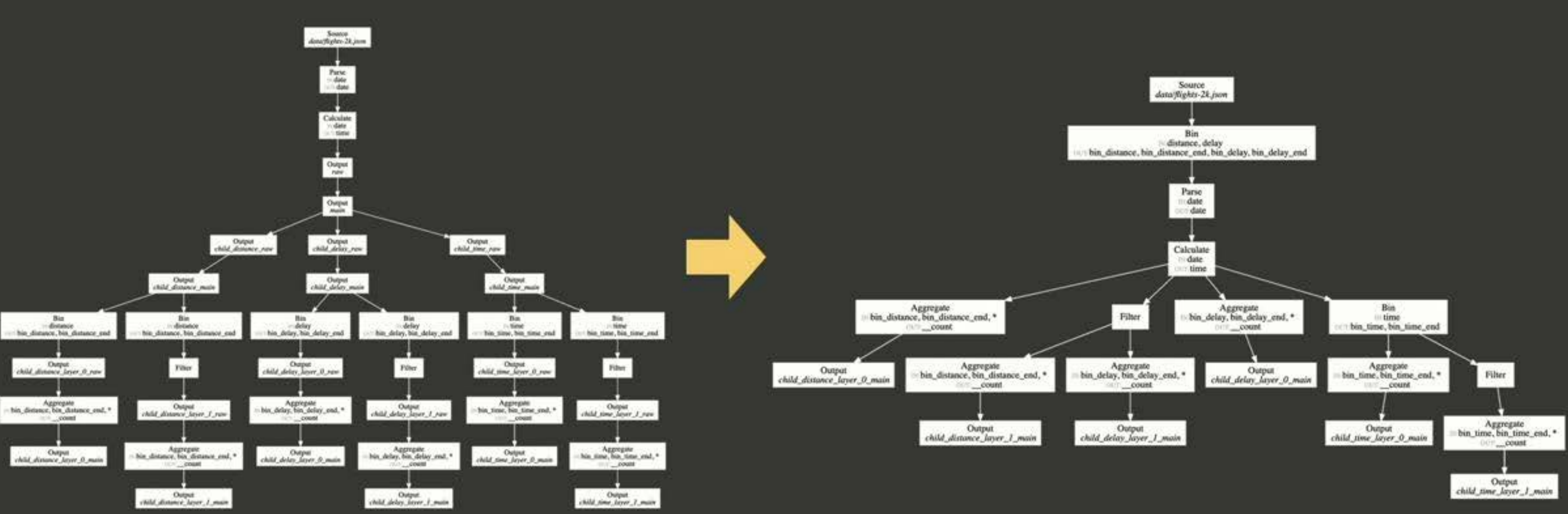
Reasoning about user task and system concerns are largely not available in end-user tools.

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

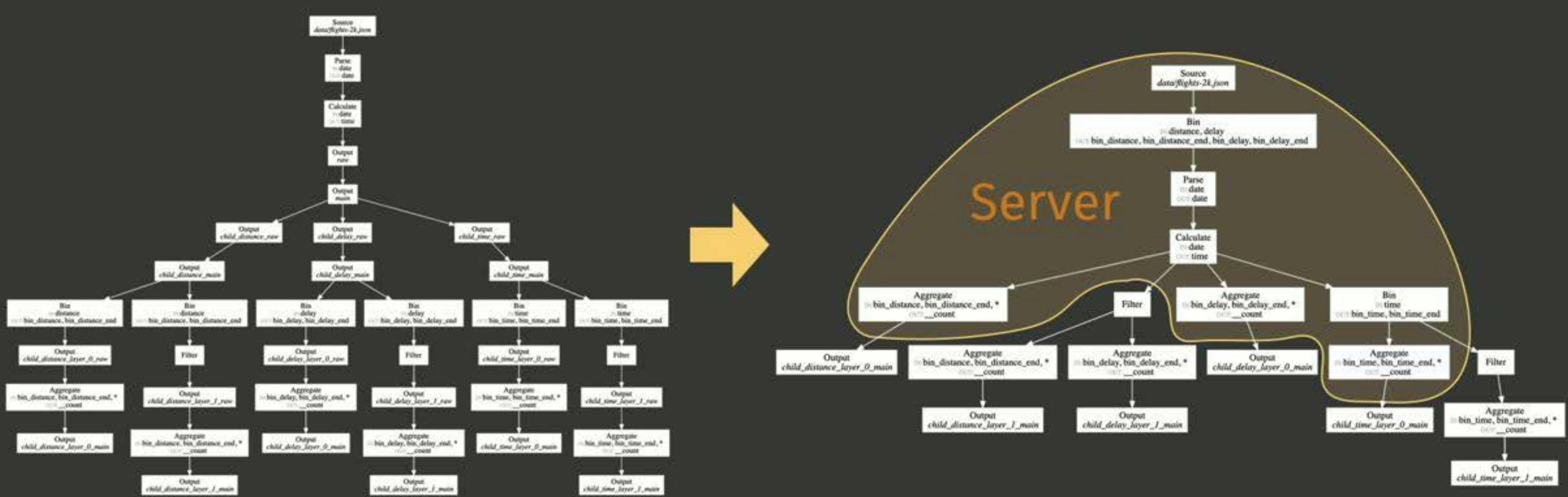
Reduce redundant computation.



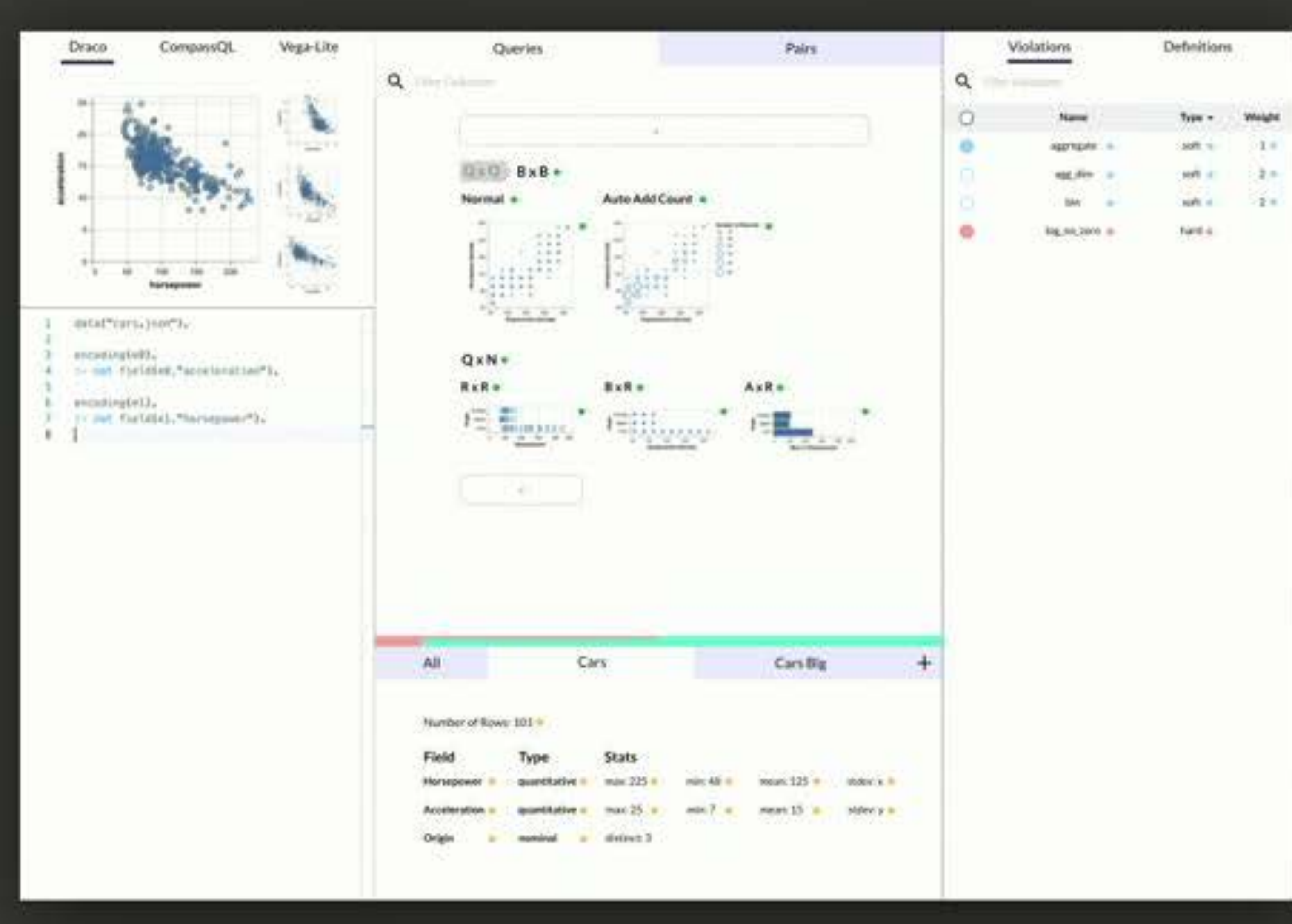
Vega-Lite's Research Frontier

Reduce redundant computation.

Push expensive computation into scalable backends. (UW, Google, OmniSci)



Draco's Research Frontier



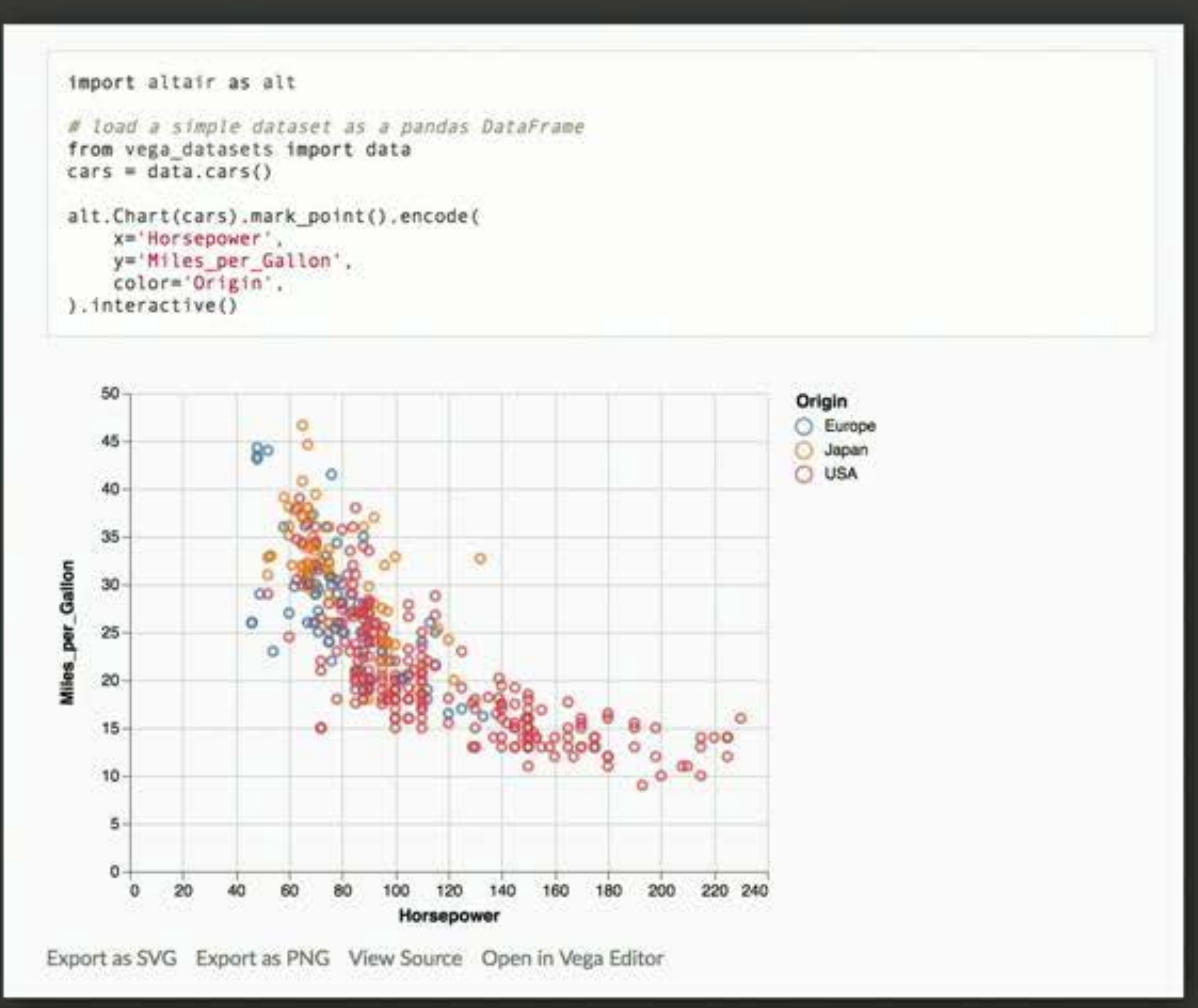
uwdata.github.io/draco-tuner

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

Evaluate impact of new perceptual models

(UW, Apple)

Draco's Research Frontier

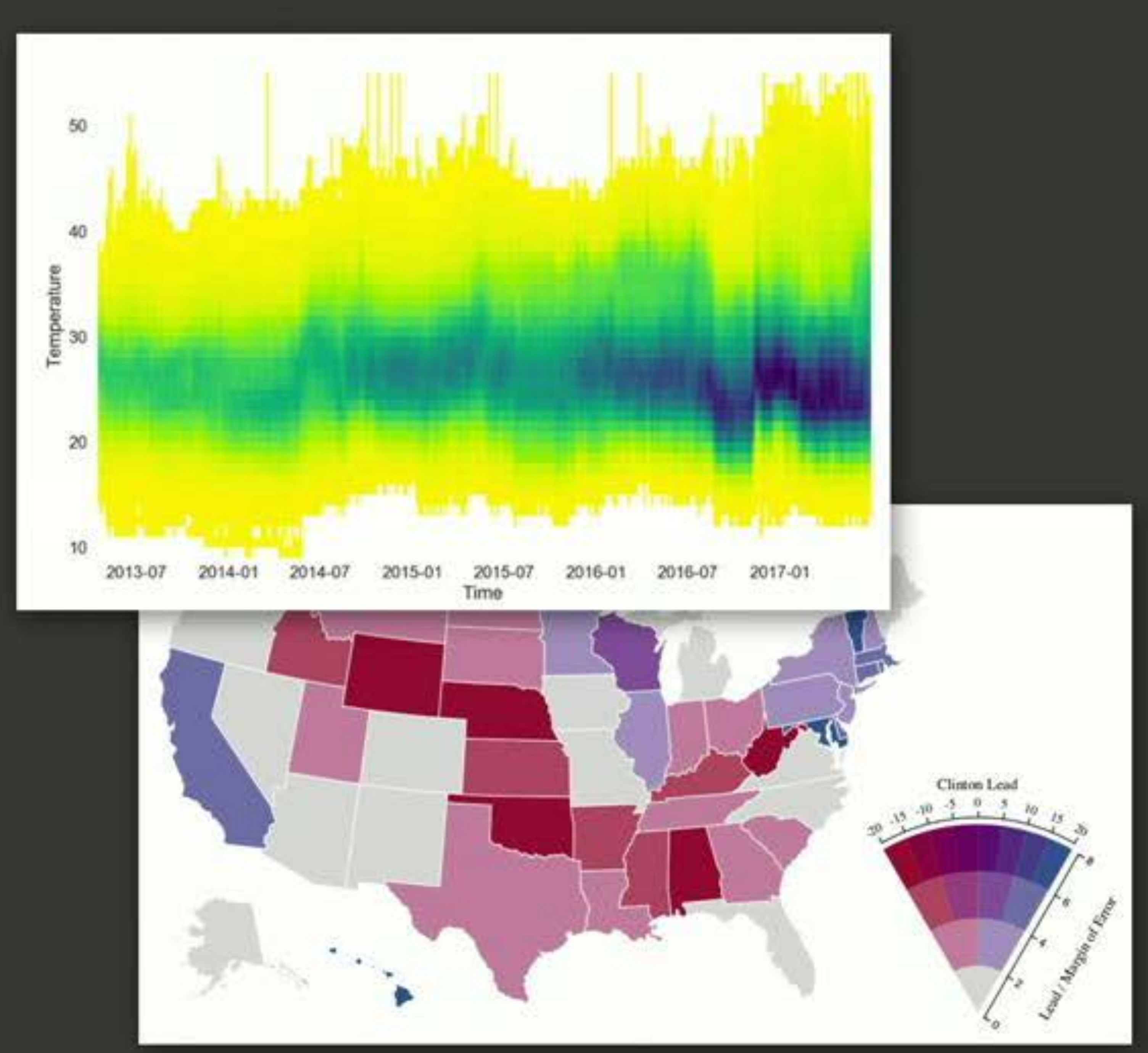


UI Tools to browse, update, and compare Draco knowledge bases.

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.

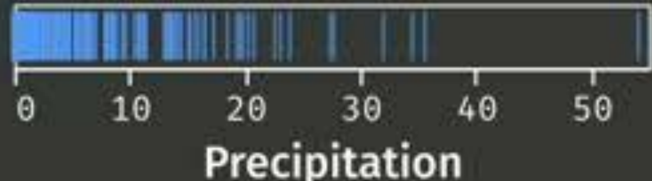
Integrate Draco into tools (e.g. Altair) to collect feedback.

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

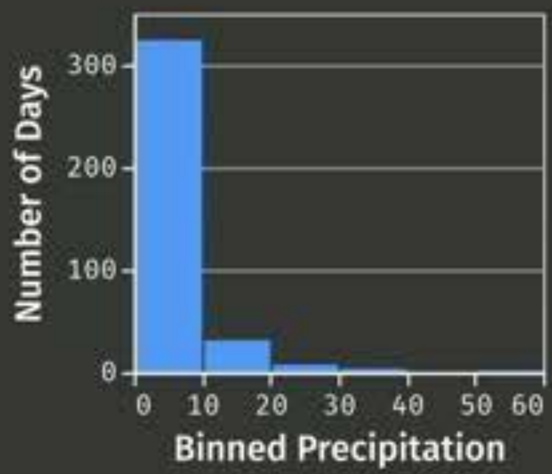
(UW, MIT, Northwestern)

Draco's Research Frontier

See Values



See Summaries



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

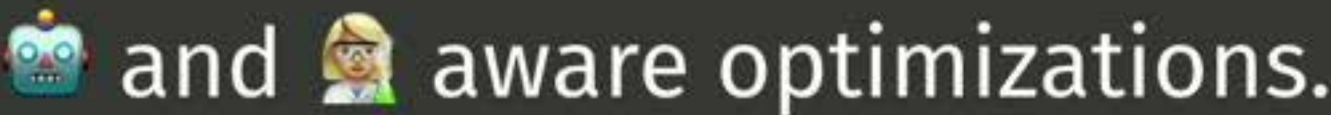
(NYU, Northwestern)

Runtime Engine for Visualization

Interactive analysis regardless of scale.

Automatically apply Falcon's optimizations. Combined with approximation.

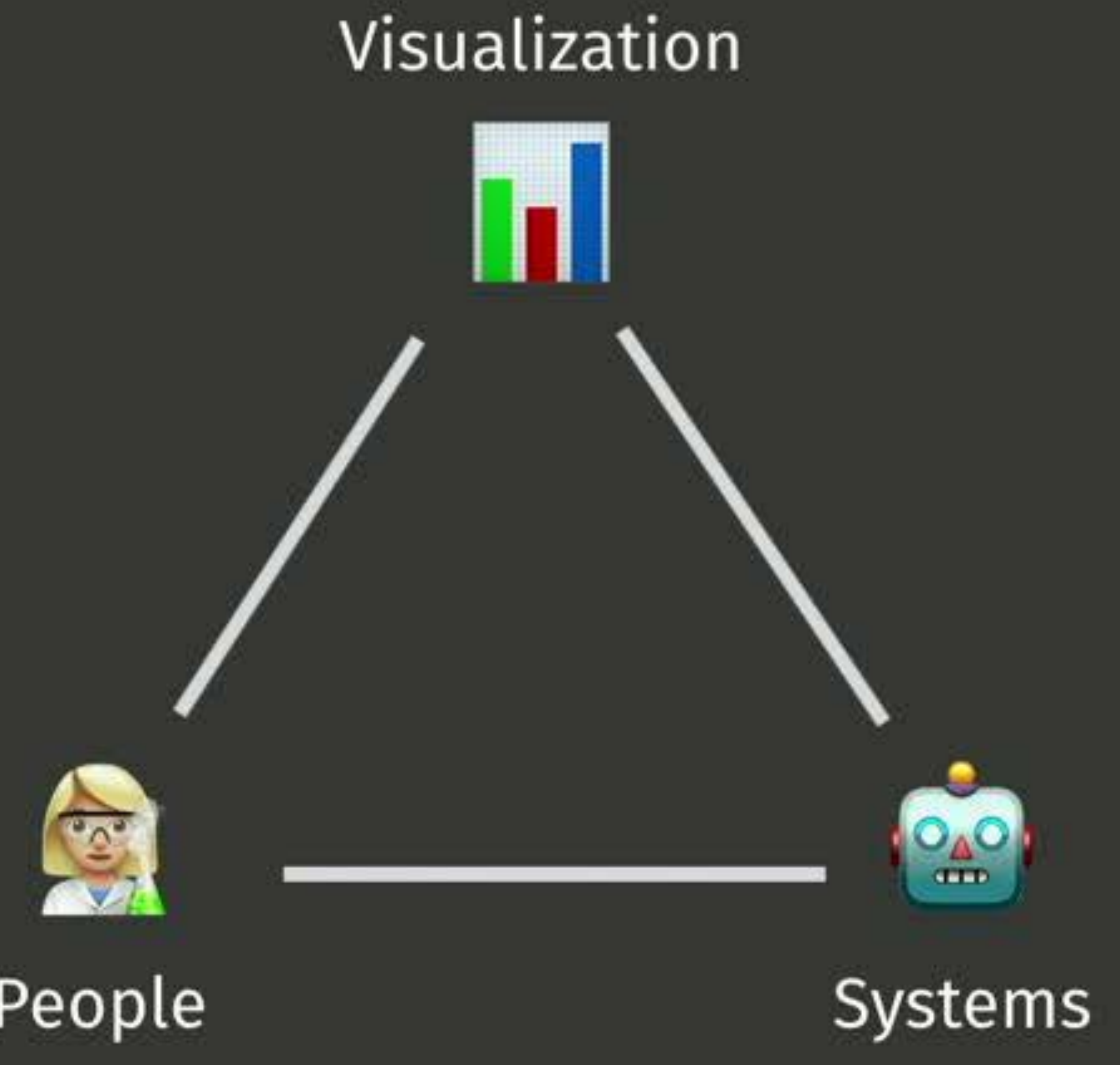
Dynamic Client-Server Optimizations. Moritz et al. *DSIA 2015*.



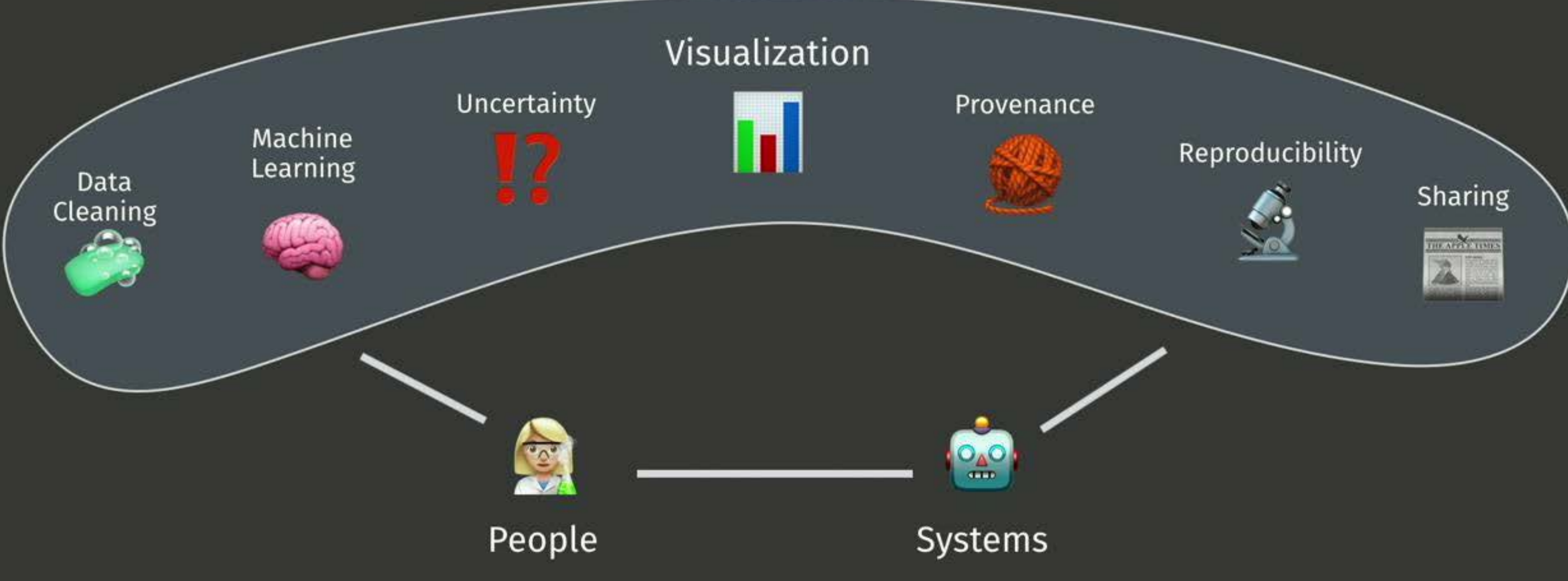
and aware optimizations.

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

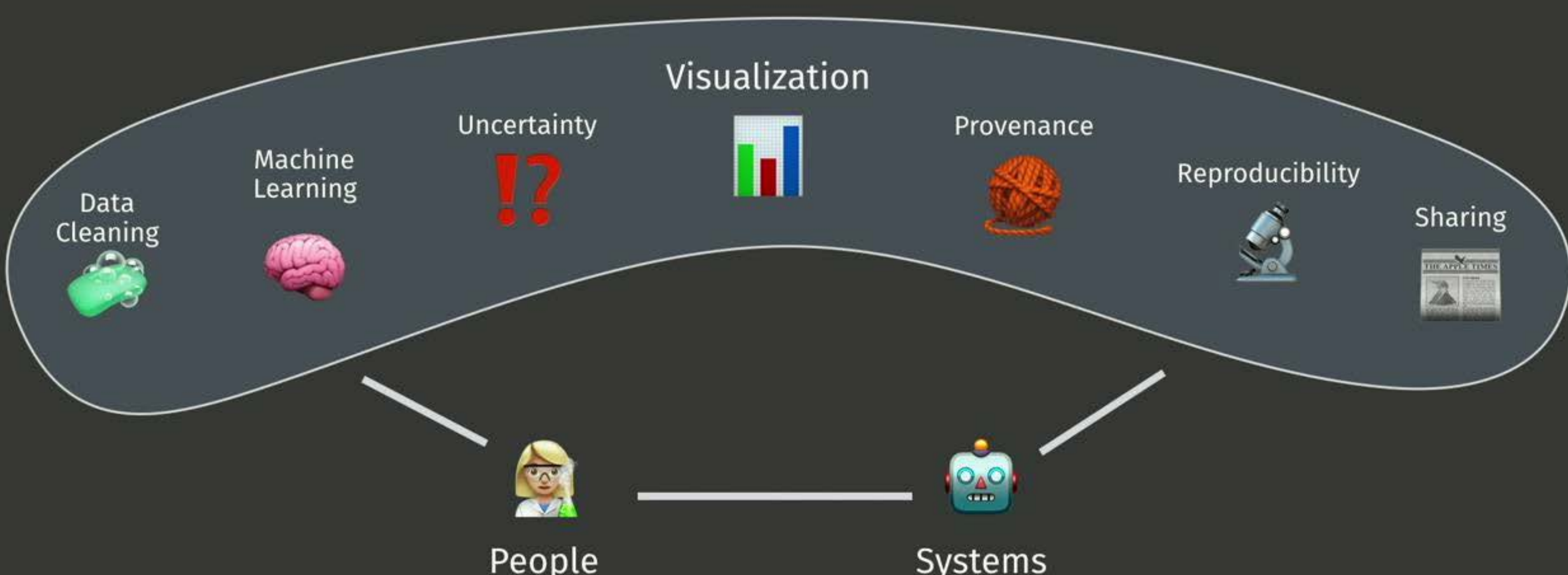
Data Analysis



Describe the analysis process
Separate specification from execution

Reason about analysis
End-to-end optimizations
Improved understanding

Data Analysis



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