



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

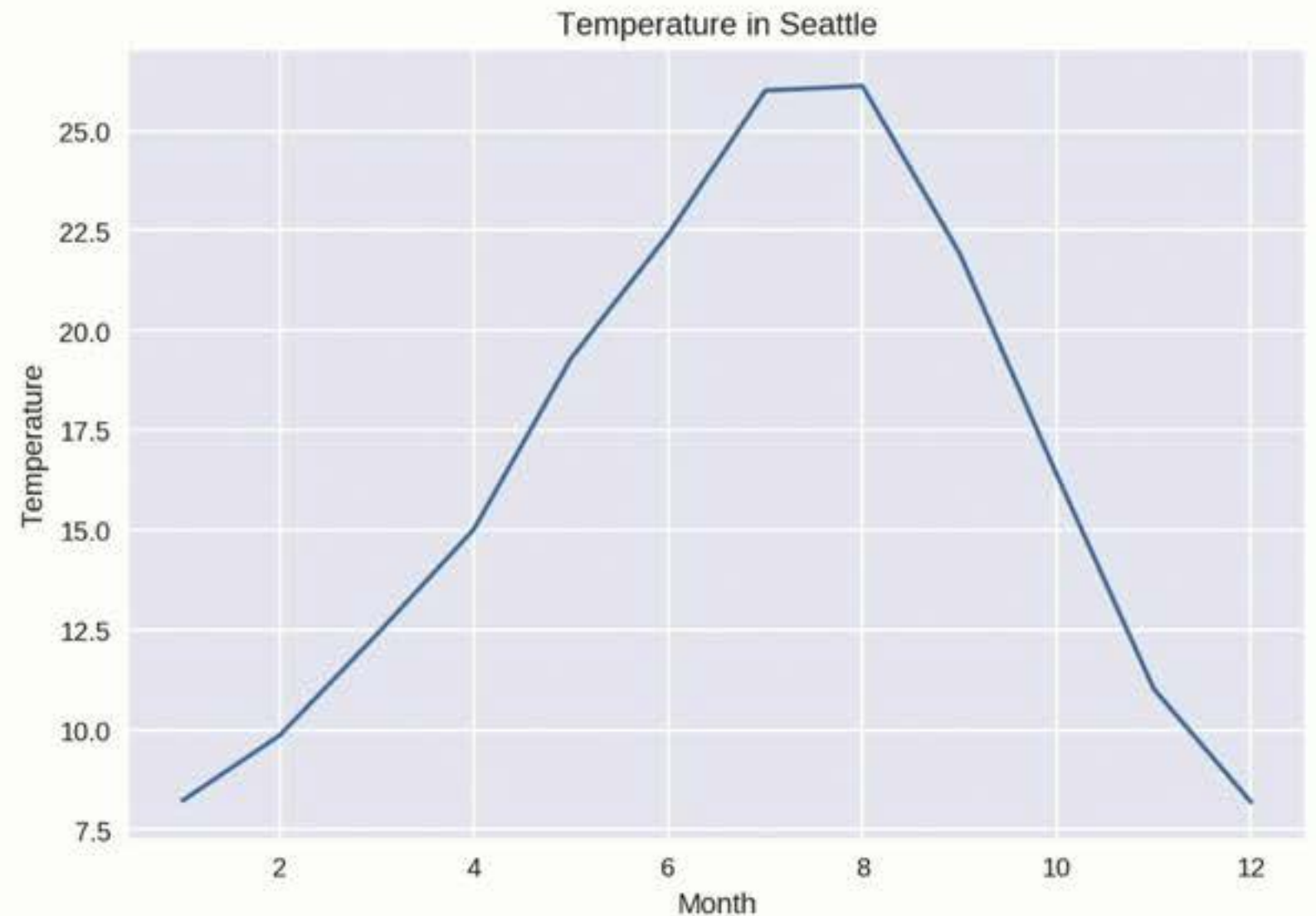
# group data and flatten it again
gb = df.groupby(['month']).temperature.mean()
grouped_df = gb.reset_index()

# 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()

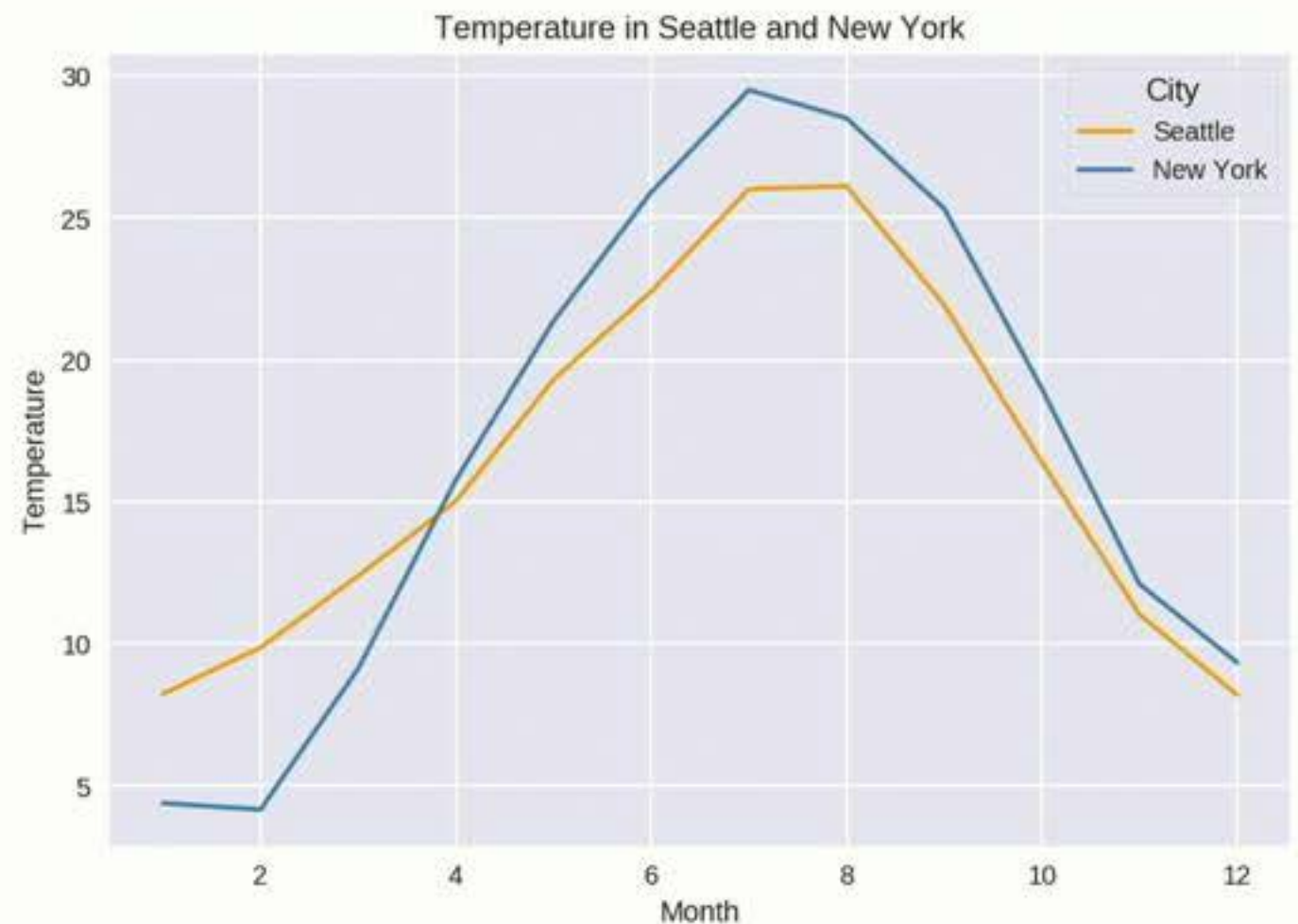
# create color map
color_map = dict(zip(df.location.unique(), ['orange', 'steelblue']))

# 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(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')

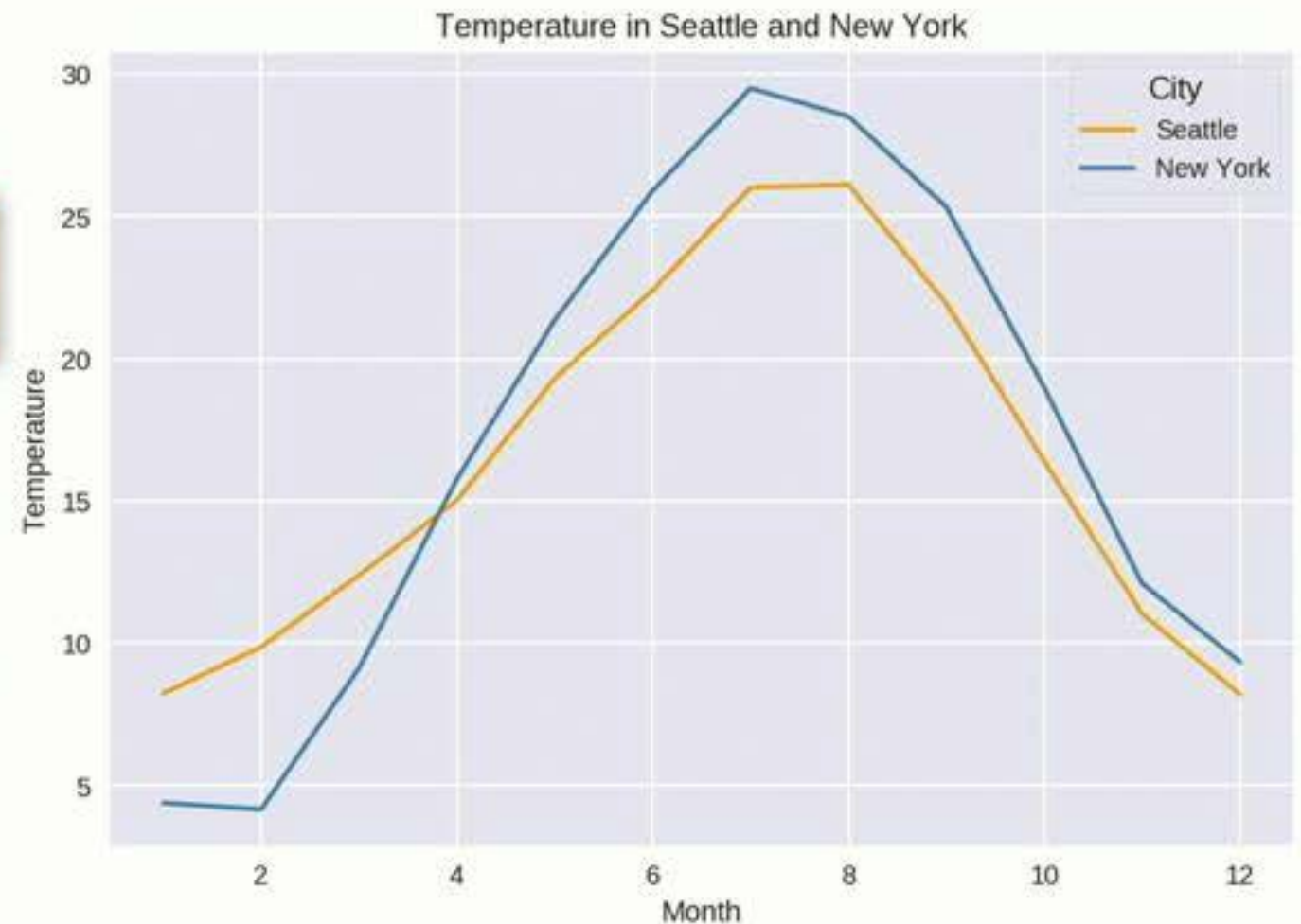
fig.show()
```

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')

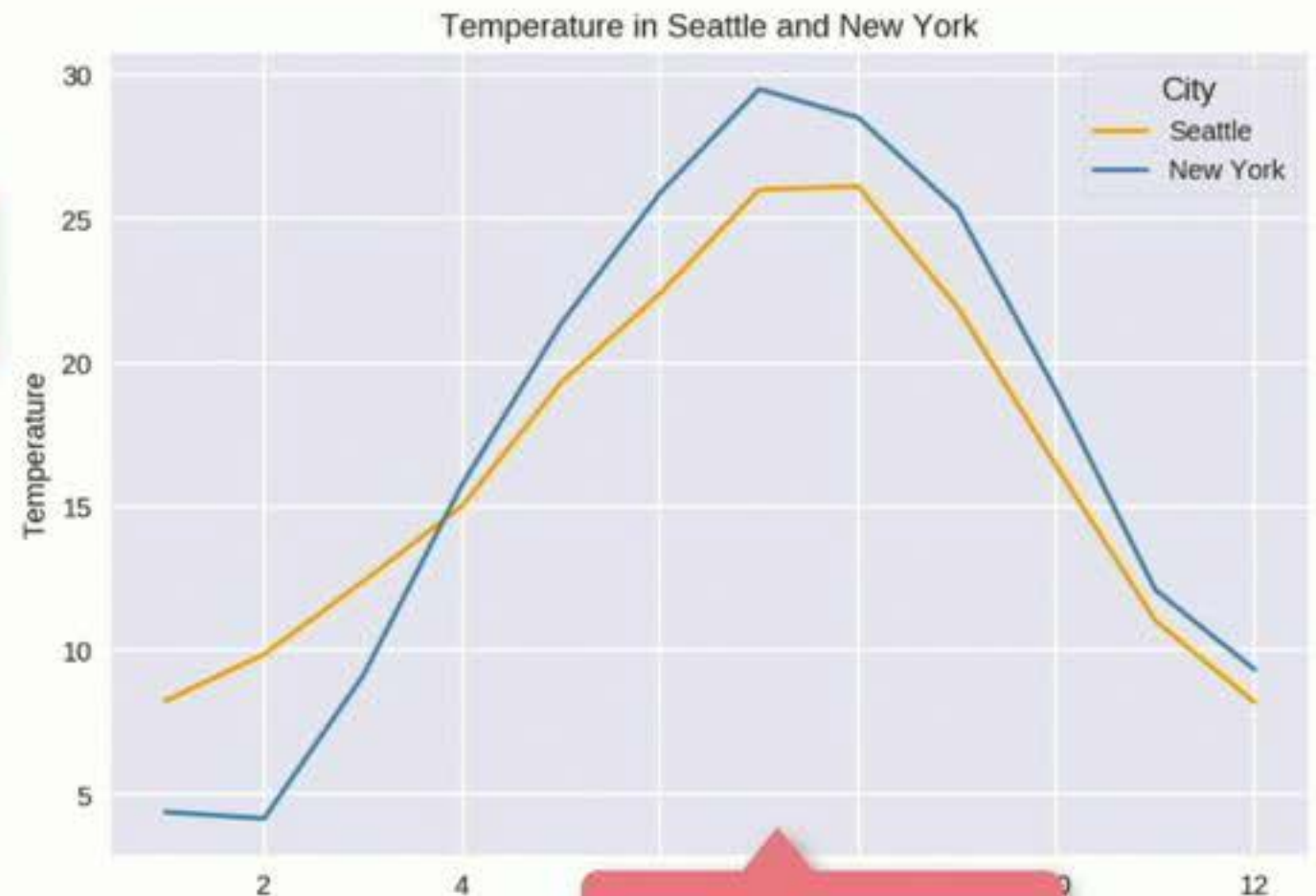
fig.show()
```

group by
location

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()
```

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

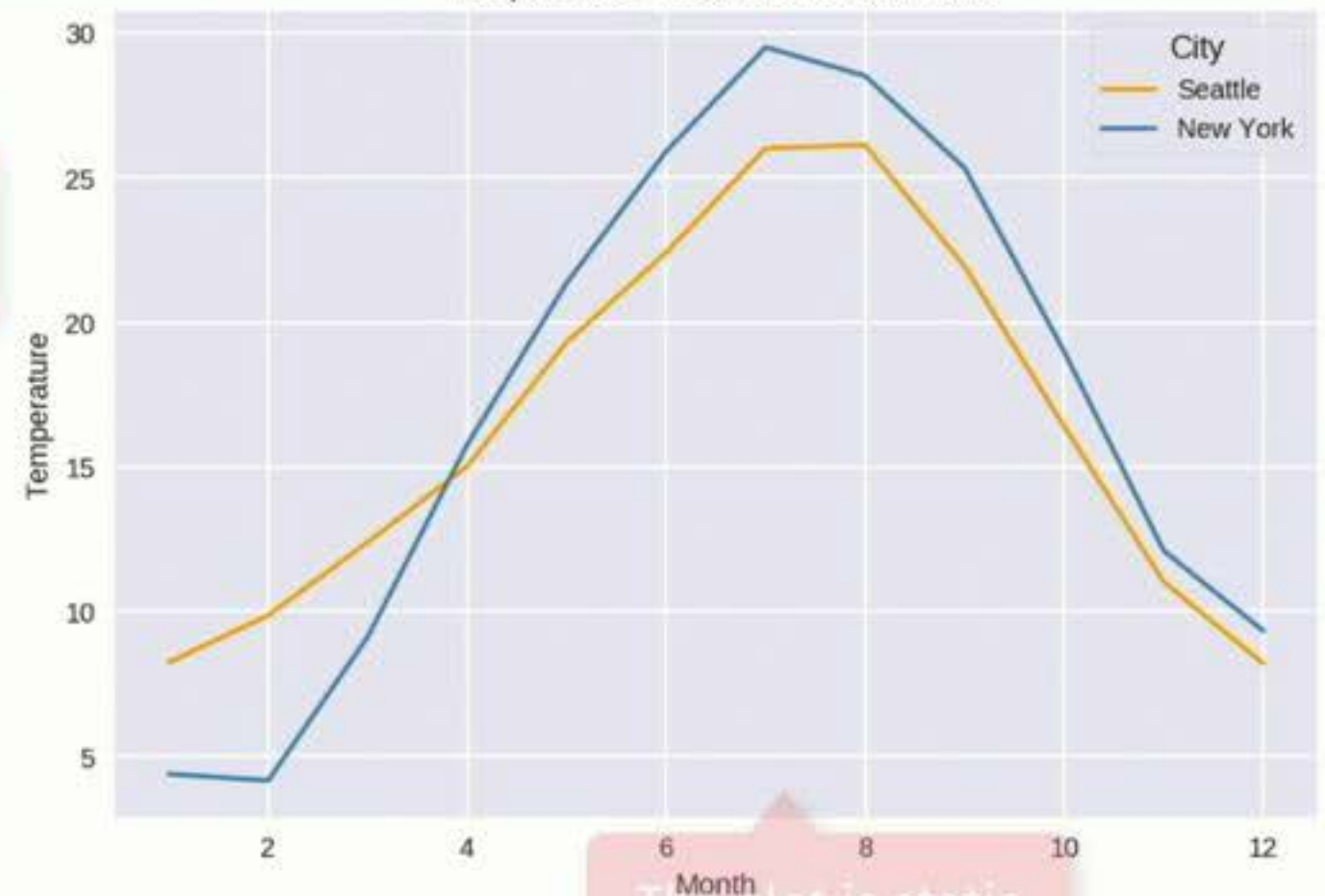
group by location

Create color map

Draw a line for each city

Don't forget the legend!

Temperature in Seattle and New York



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()
```

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

group by location

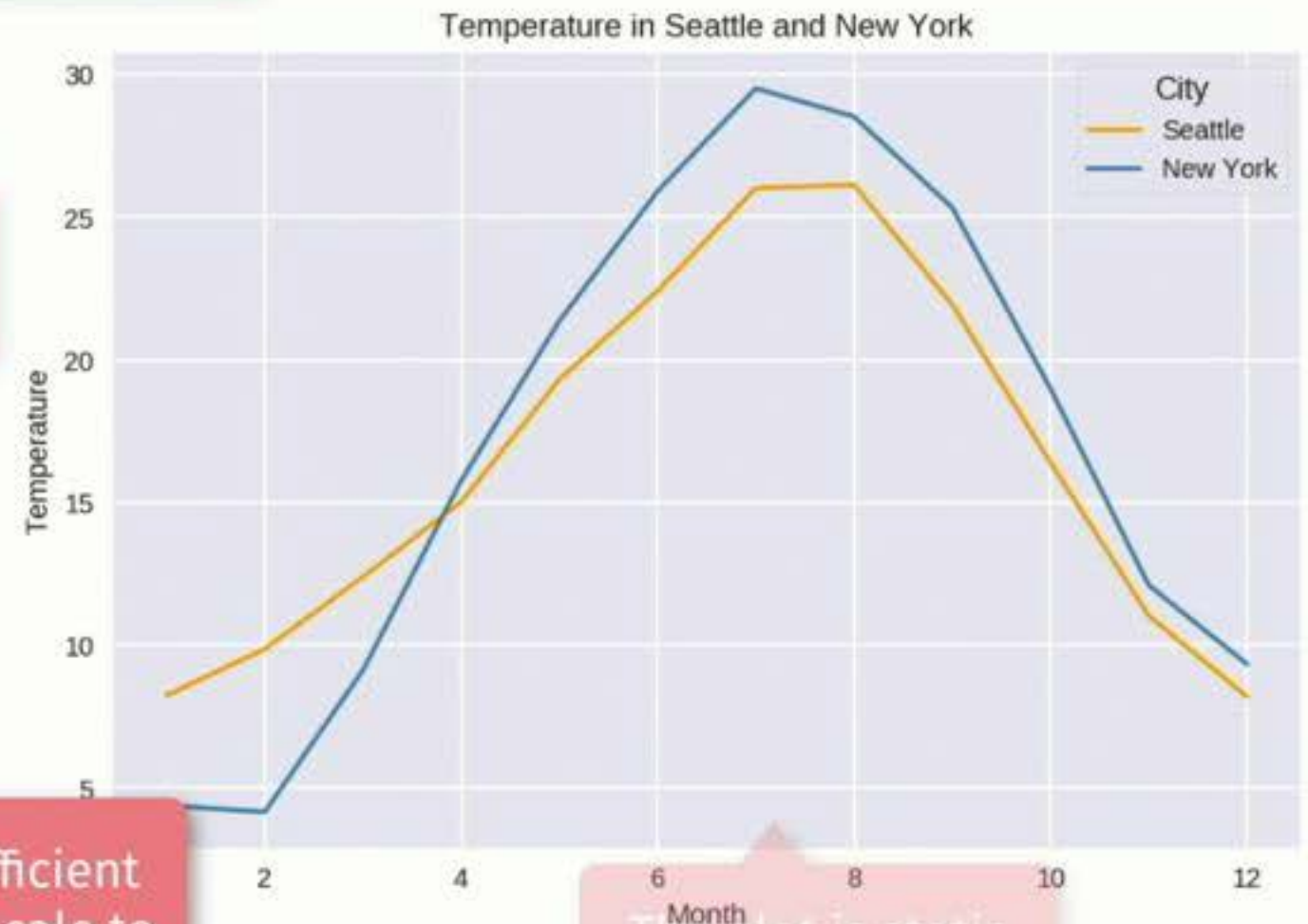
Create color map

Draw a line for each city

Don't forget the legend!

This is inefficient and won't scale to large data

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()
```

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

Is this design good?

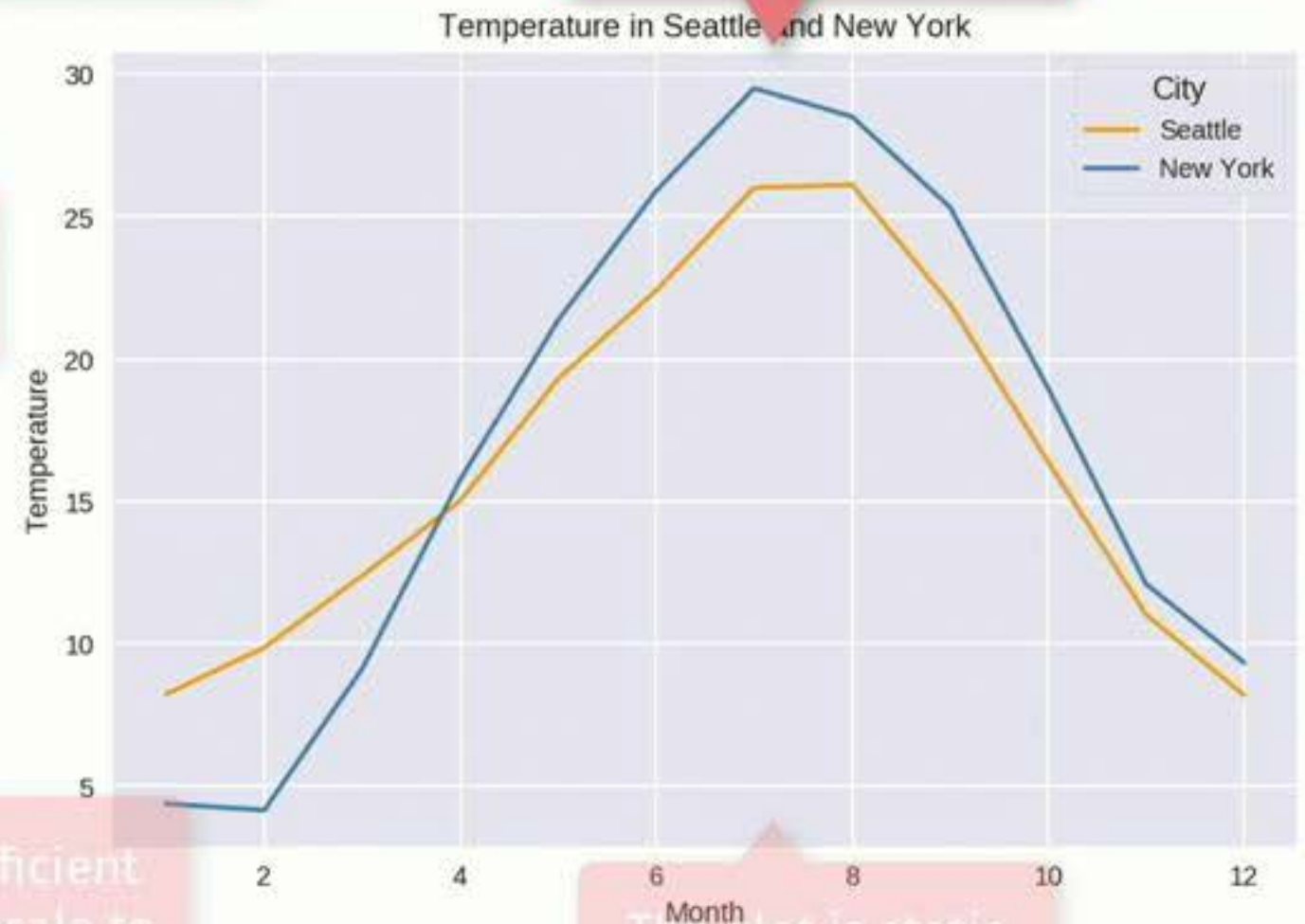
Create color map

Draw a line for each city

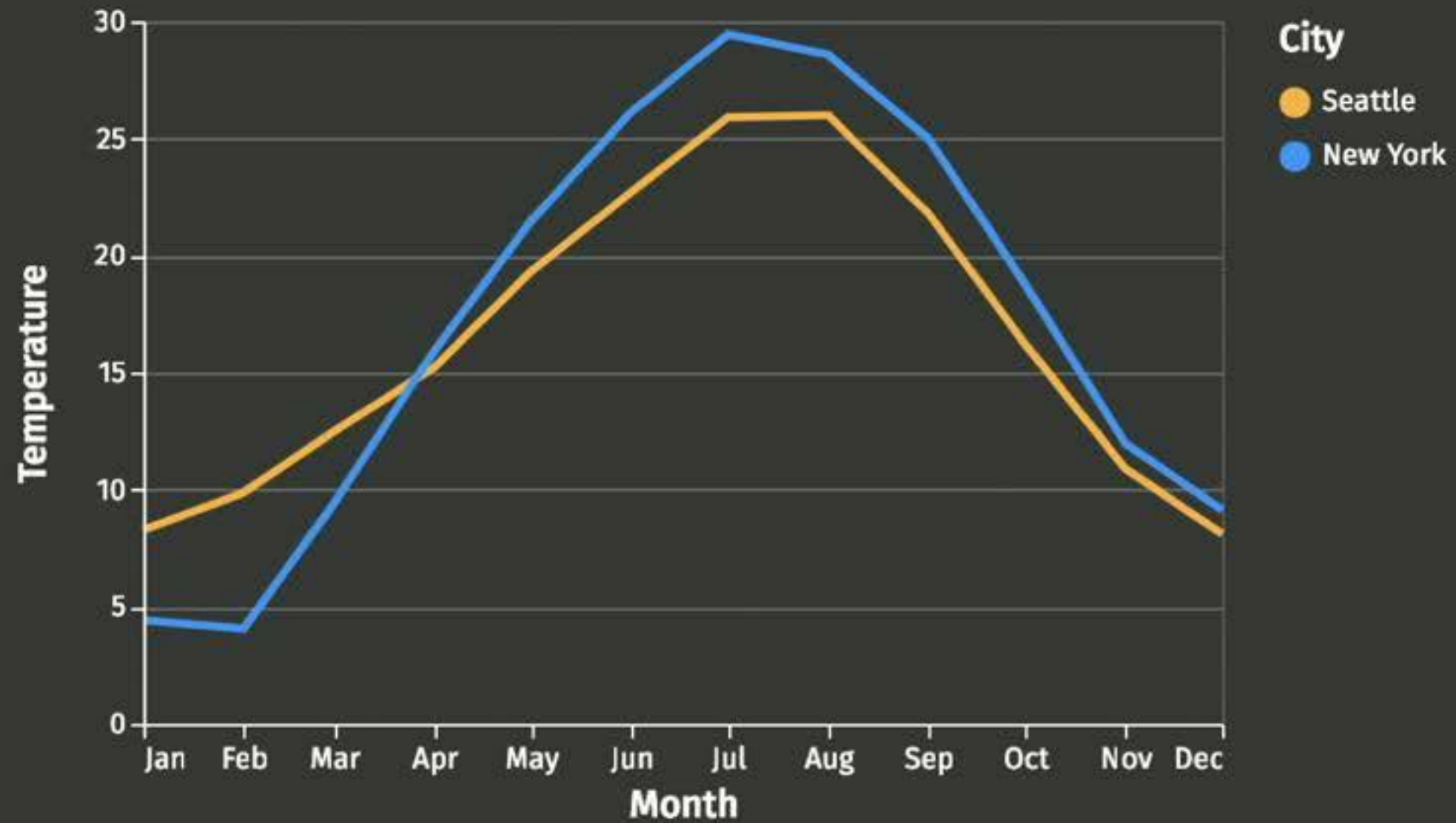
This is inefficient and won't scale to large data

Don't forget the legend!

The plot is static



For any good design...

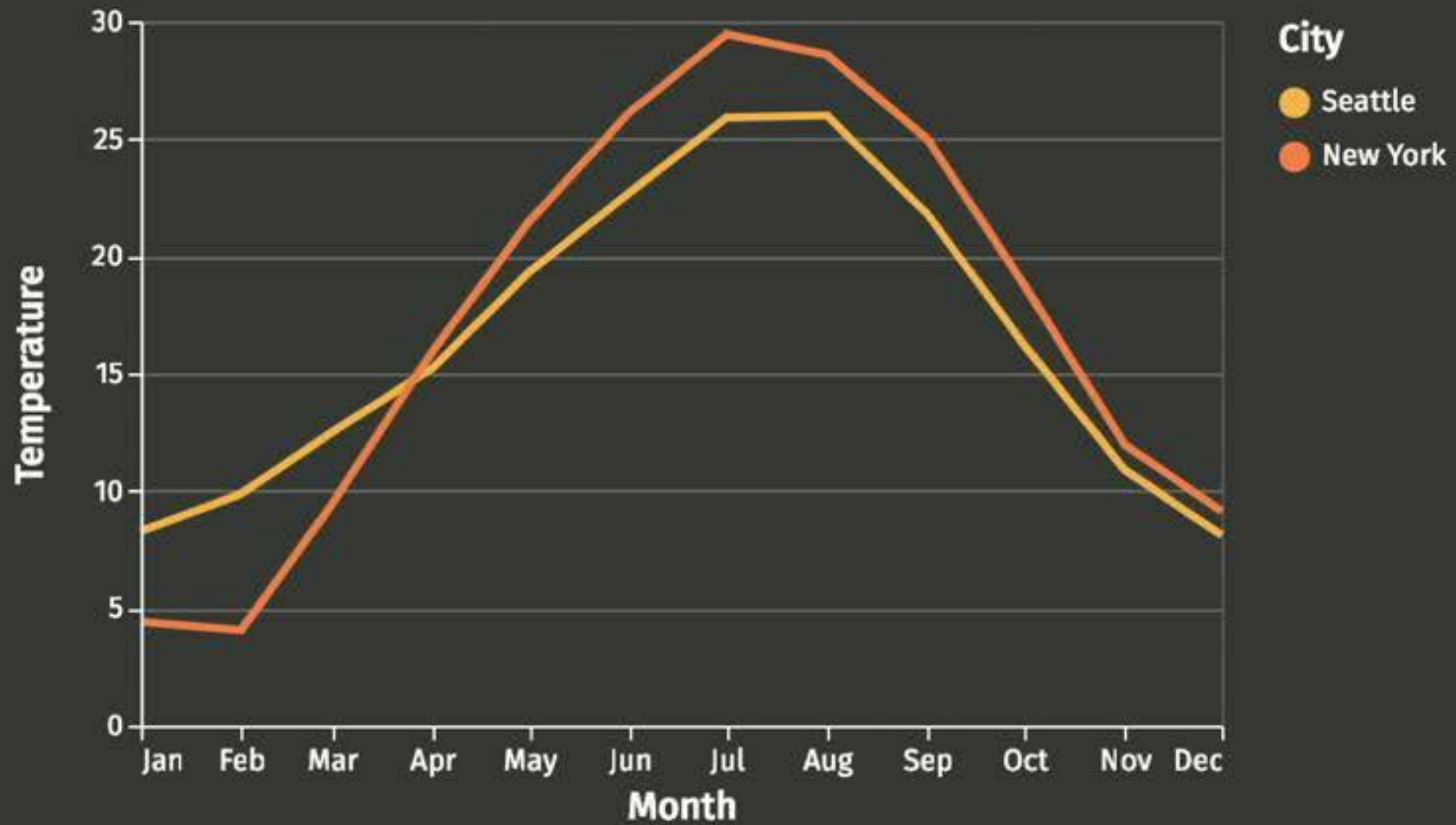


...there are many poor designs.



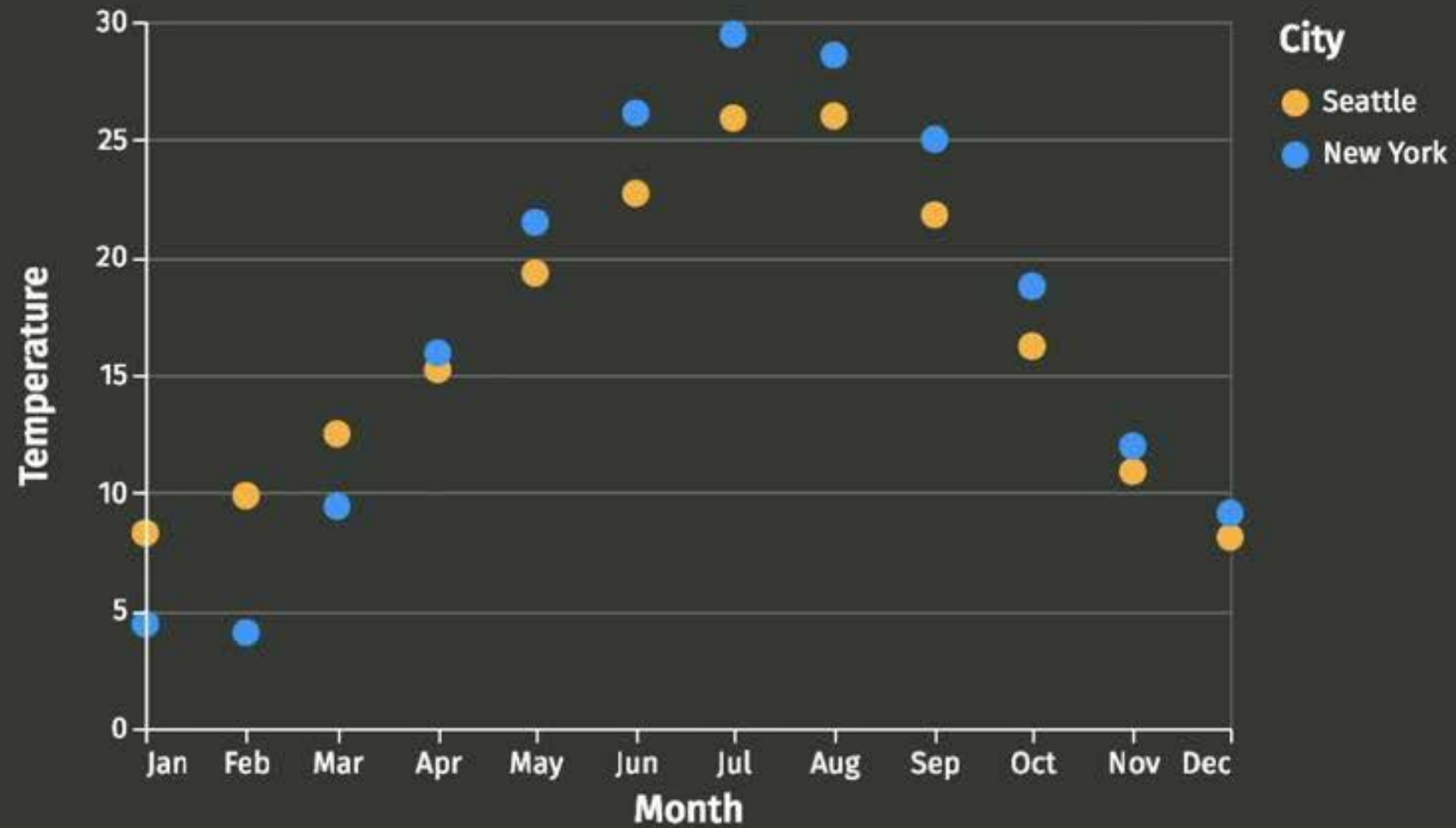
Missing legend

...there are many poor designs.



Poor color choice

...there are many poor designs.

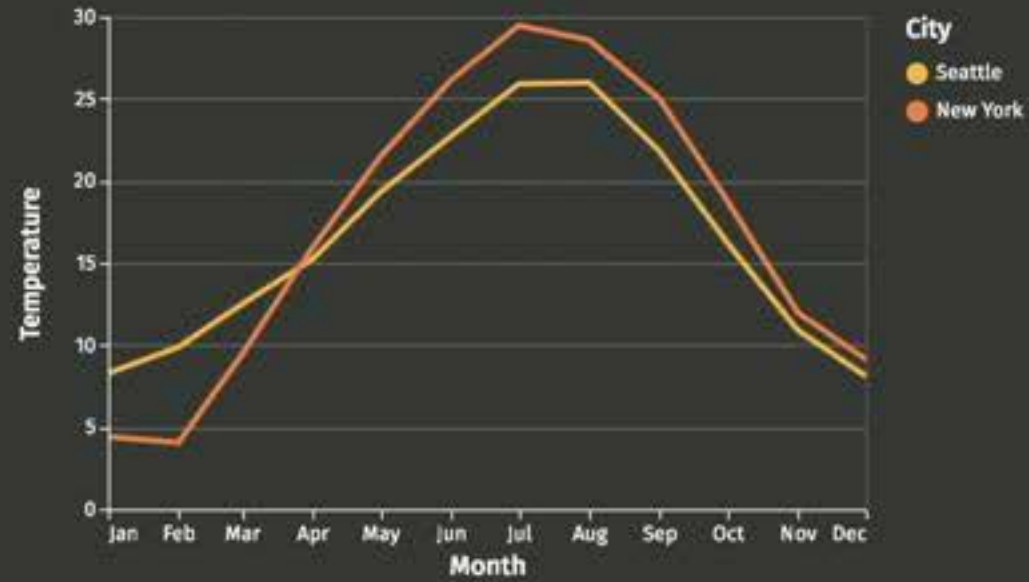


Non-optimal mark

...there are many poor designs.



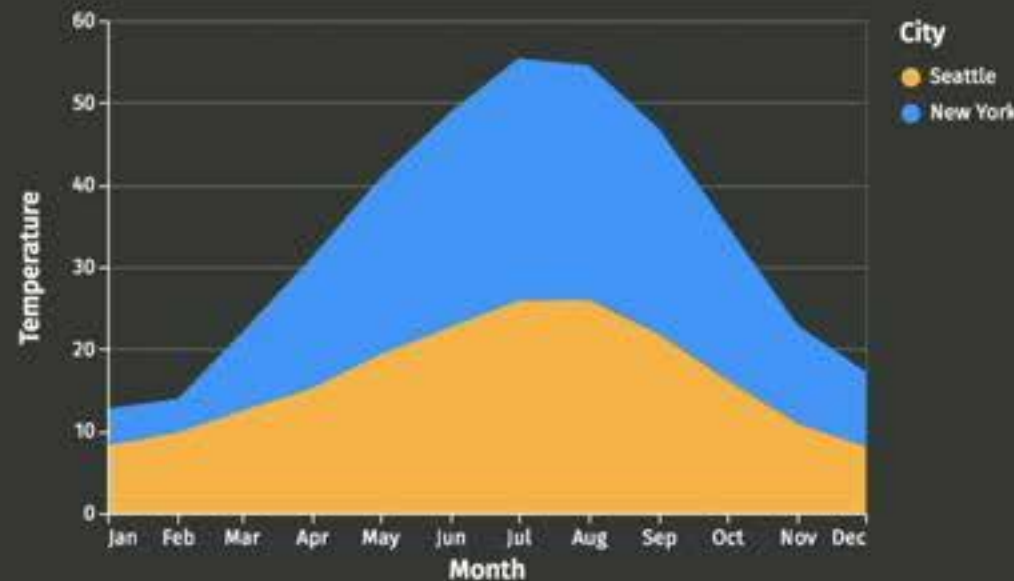
Missing legend



Poor color choice



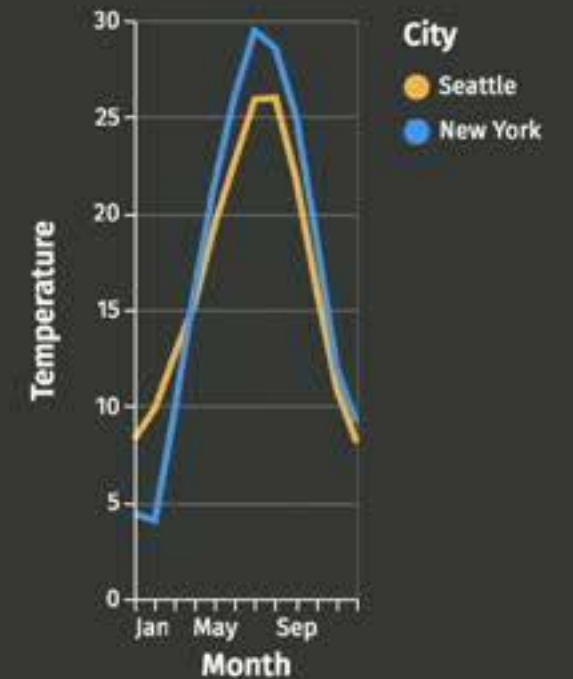
Non-optimal mark



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**?

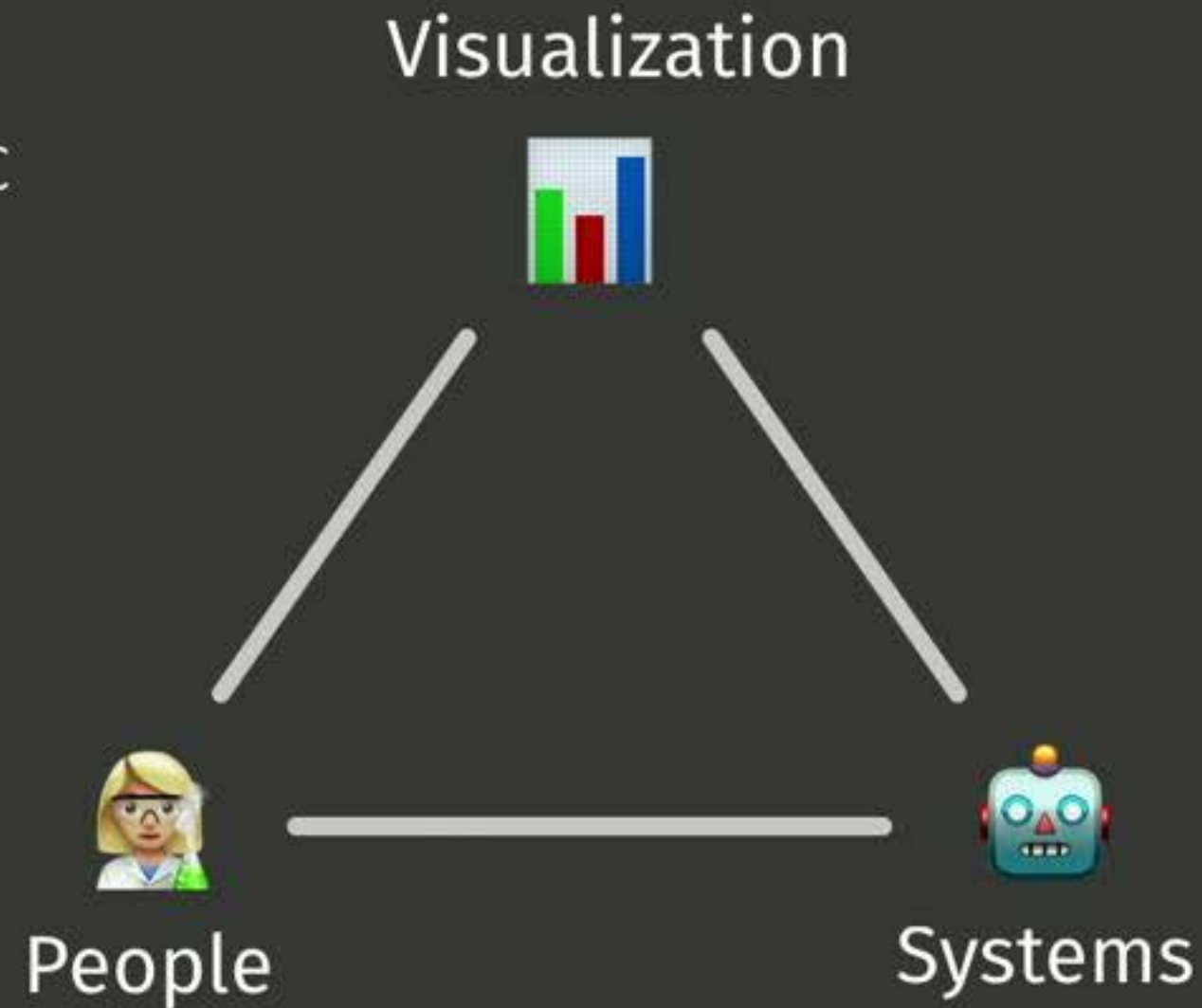
Programming tools are designed for **manual authoring**.
Good design is the **responsibility of the human designer**.



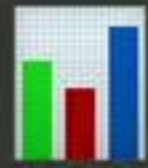
Tools do not provide computational guidance.



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



Visualization



People



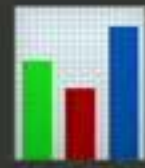
Systems

Optimizations often rely on **abstracting away the person** using the computer.



Tools have **little understanding** of the **user's goals**.

Visualization



People

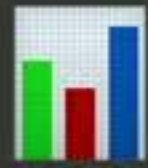


Systems

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**.

Visualization



People



Systems

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.

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

Uncertainty for Users. Moritz Fisher. *HILDA 2017*.

Lessons from Pangloss. Moritz Fisher. *Uncertainty 2017*.

Perfopticon. Moritz, et al. *Eurovis 2015*.

Spacegraphcats. Brown et al. *bioRxiv 2018*.

Uncertainty

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.*

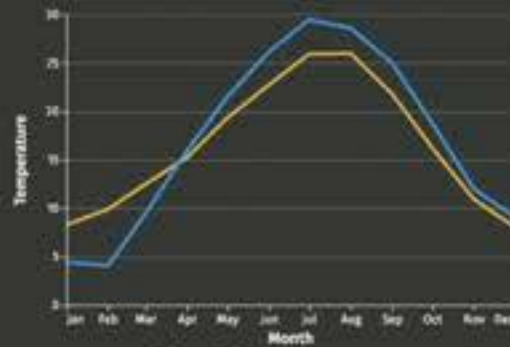
Fast and reliable approximations for
data exploration

Guidance

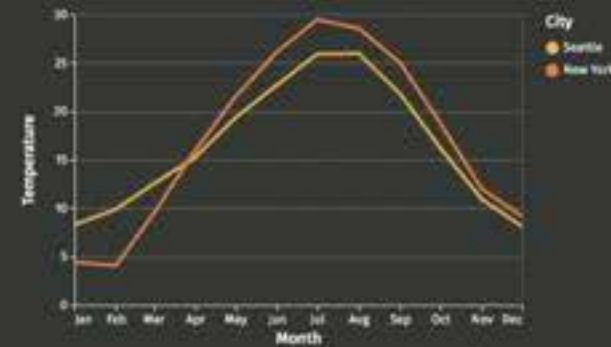
?

Representation

Guidance



Missing legend



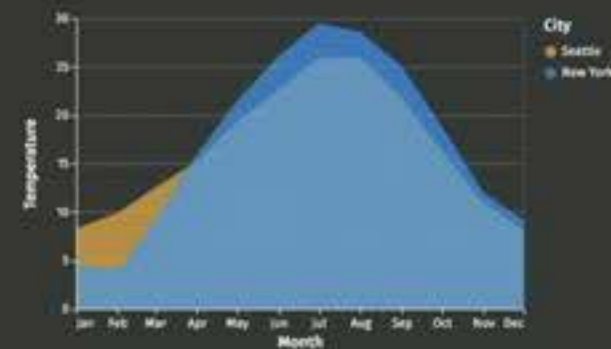
Poor color choice



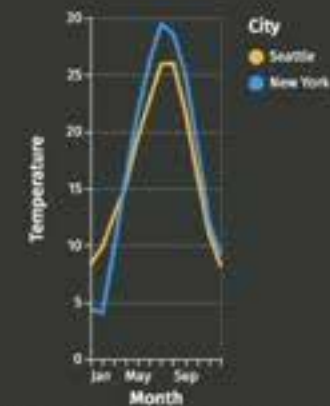
Non-optimal mark



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



Vega-Lite

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

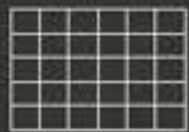


Vega-Lite

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



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	4.6

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	9.9	0.0	6.2	sun

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	21.4

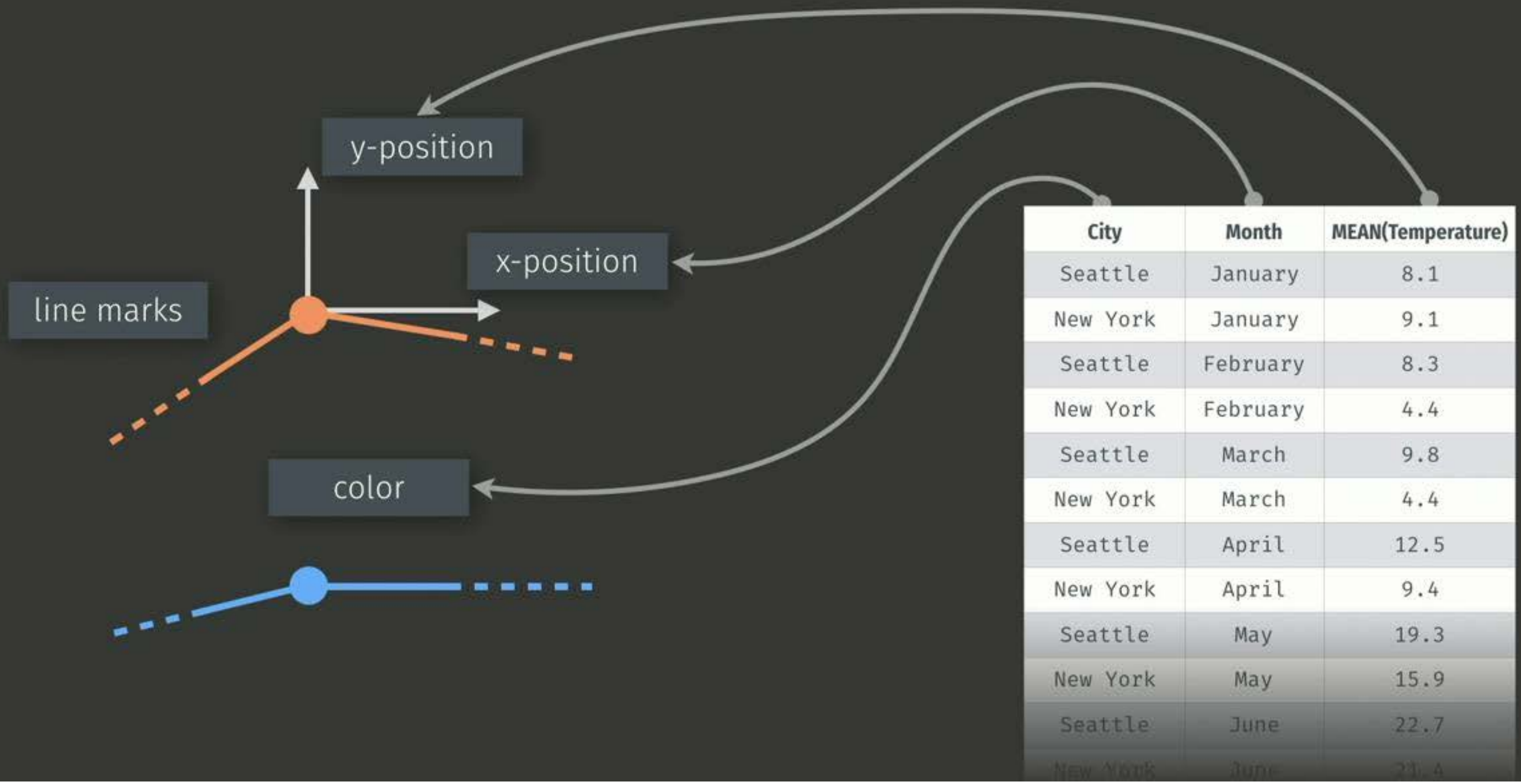
Visualizations Encode Data as Visual Properties

line marks



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



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	21.6

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

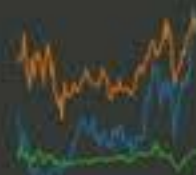
Marks Data-representative graphics



Symbol



Rect



Line



Area

Abc

Text

Building Blocks of Visualization

Grammar of Graphics. Wilkinson. 2015.

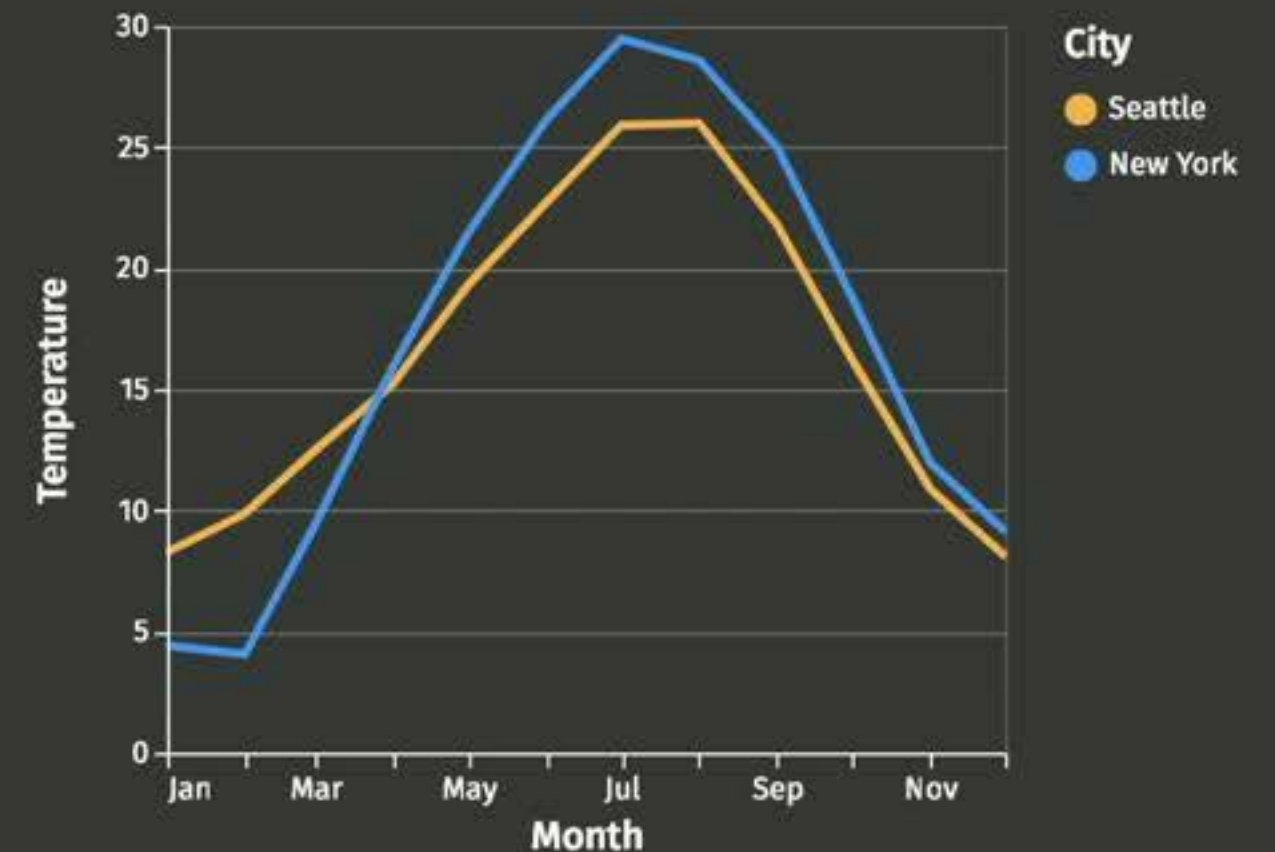
Data Input data to visualize
`weather.csv`

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

Abc

Text

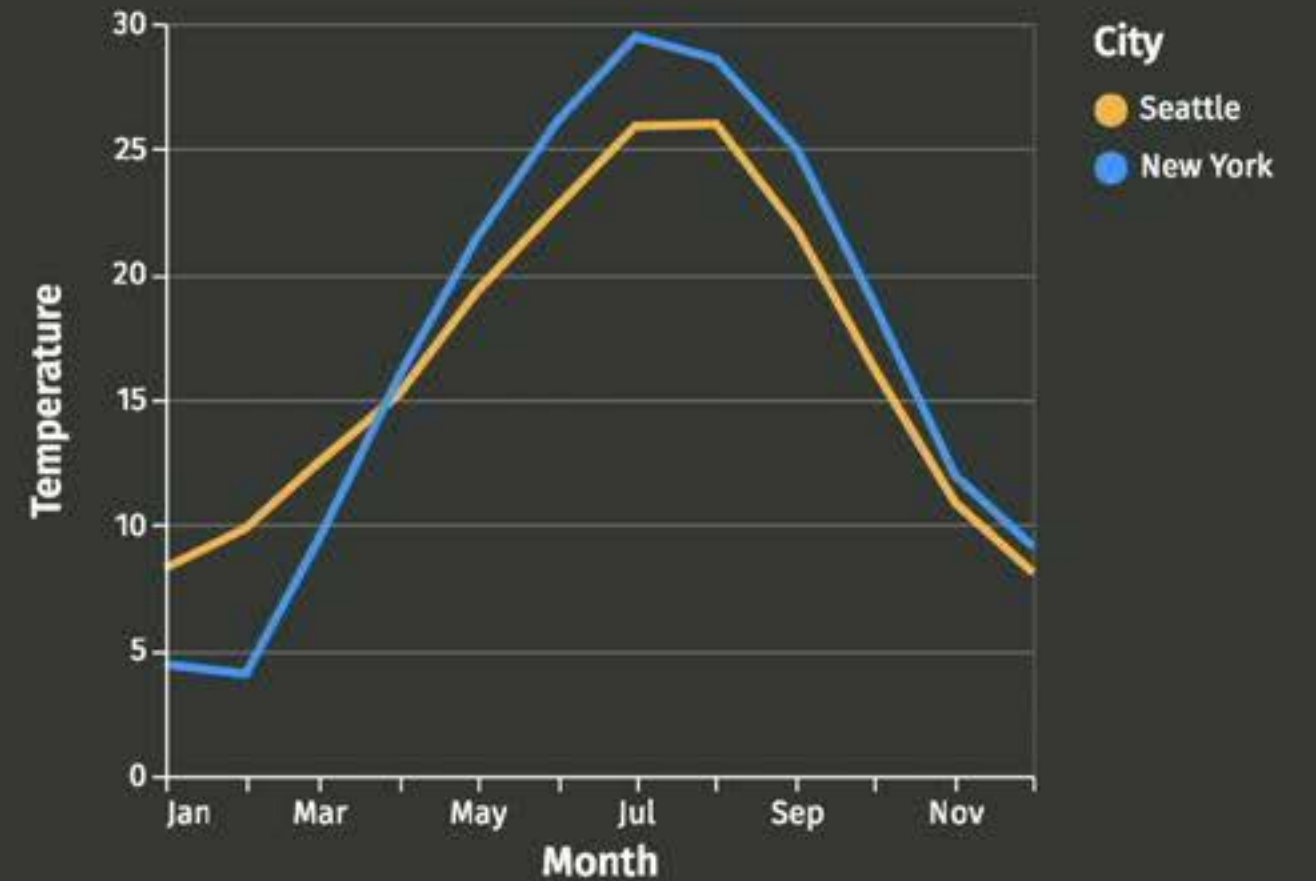


Vega-Lite

vega.github.io/vega-lite

Vega-Lite Encodings

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



Vega-Lite Encodings

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



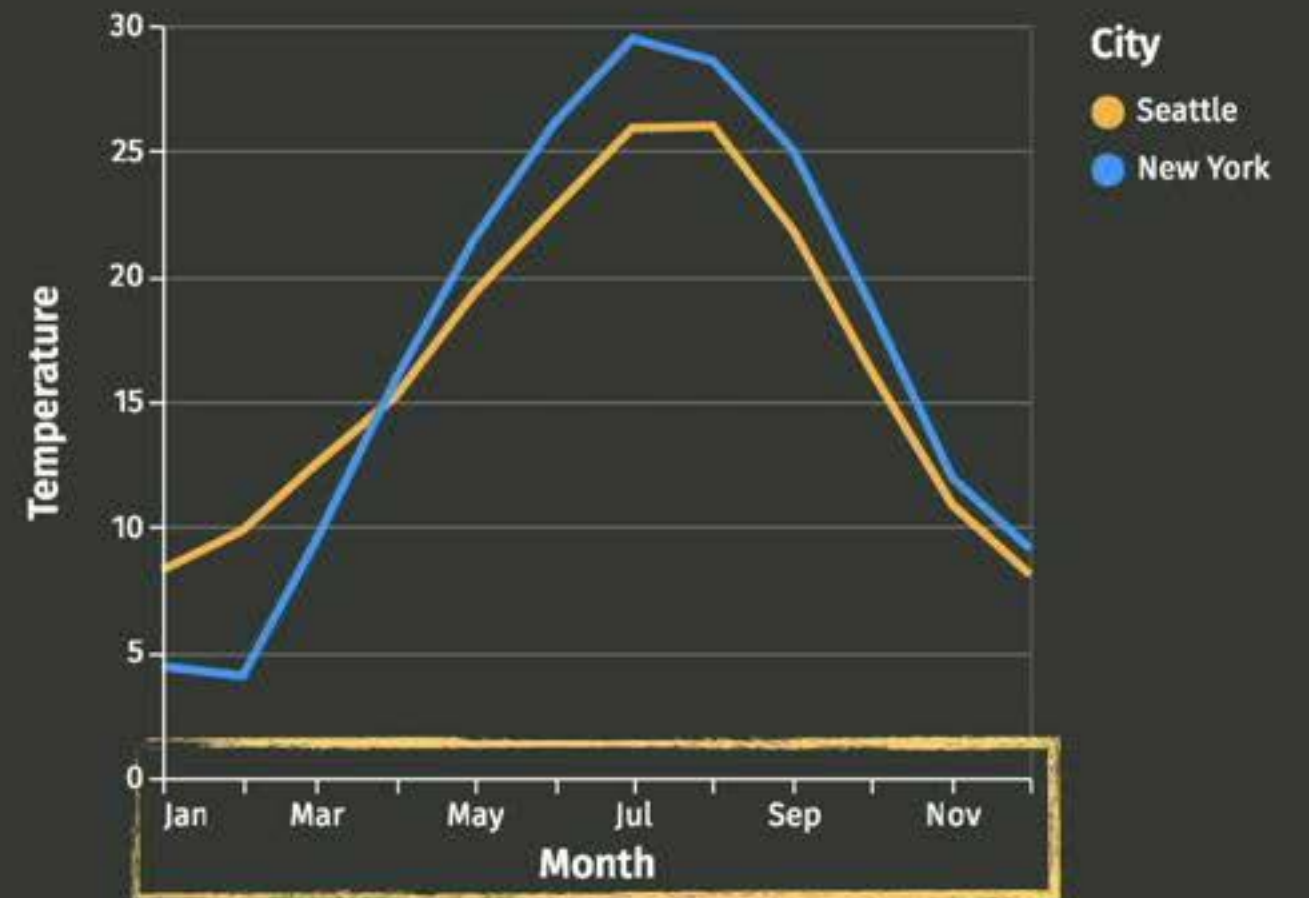
Vega-Lite Encodings

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



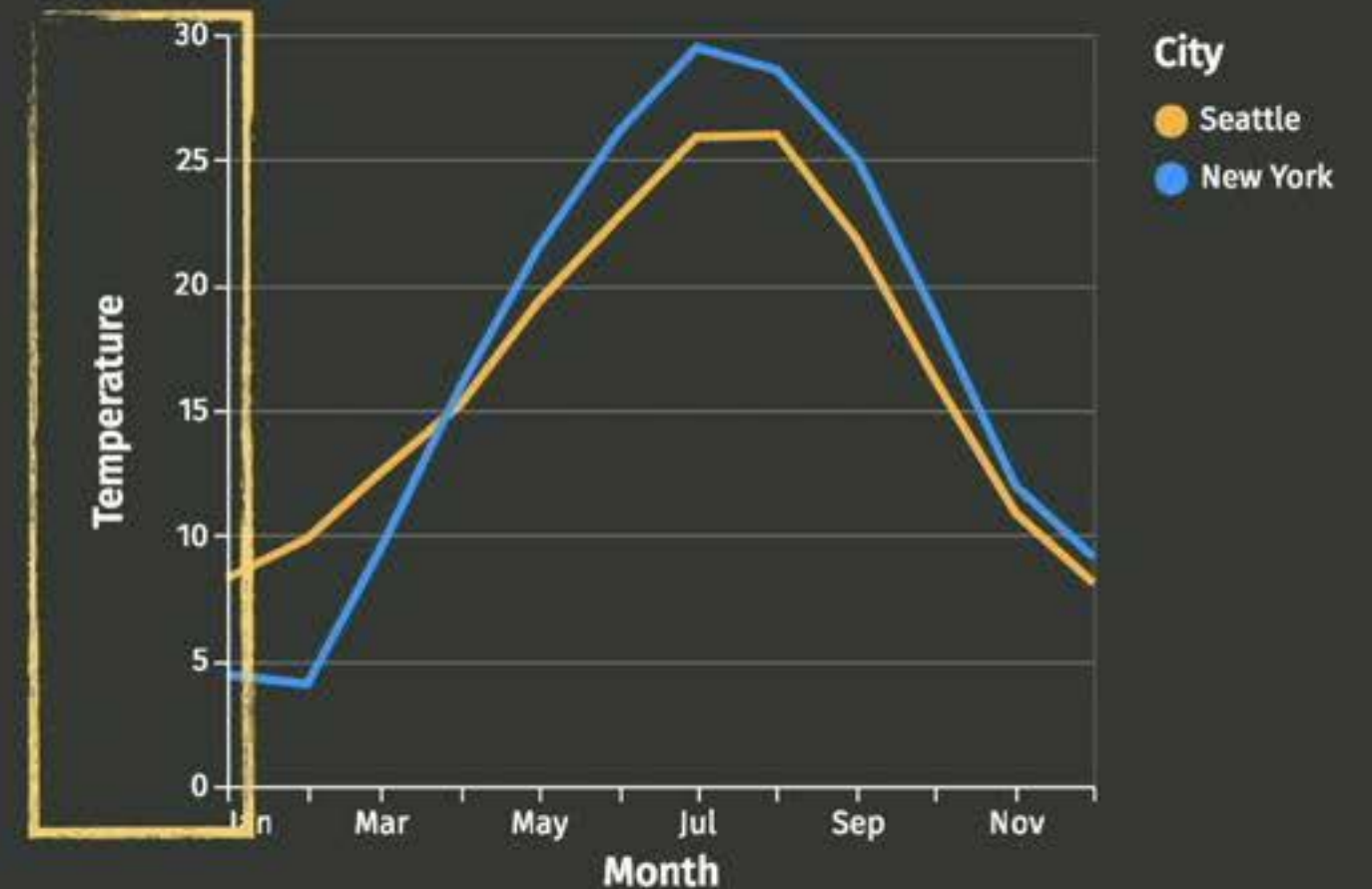
Vega-Lite Encodings

```
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
```



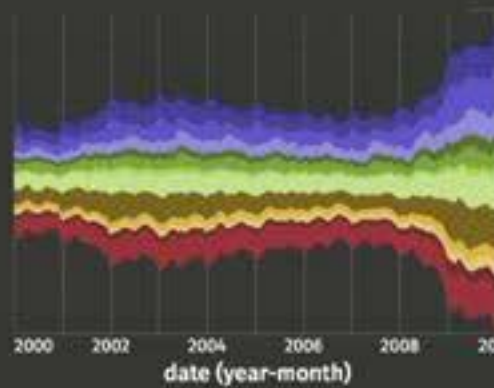
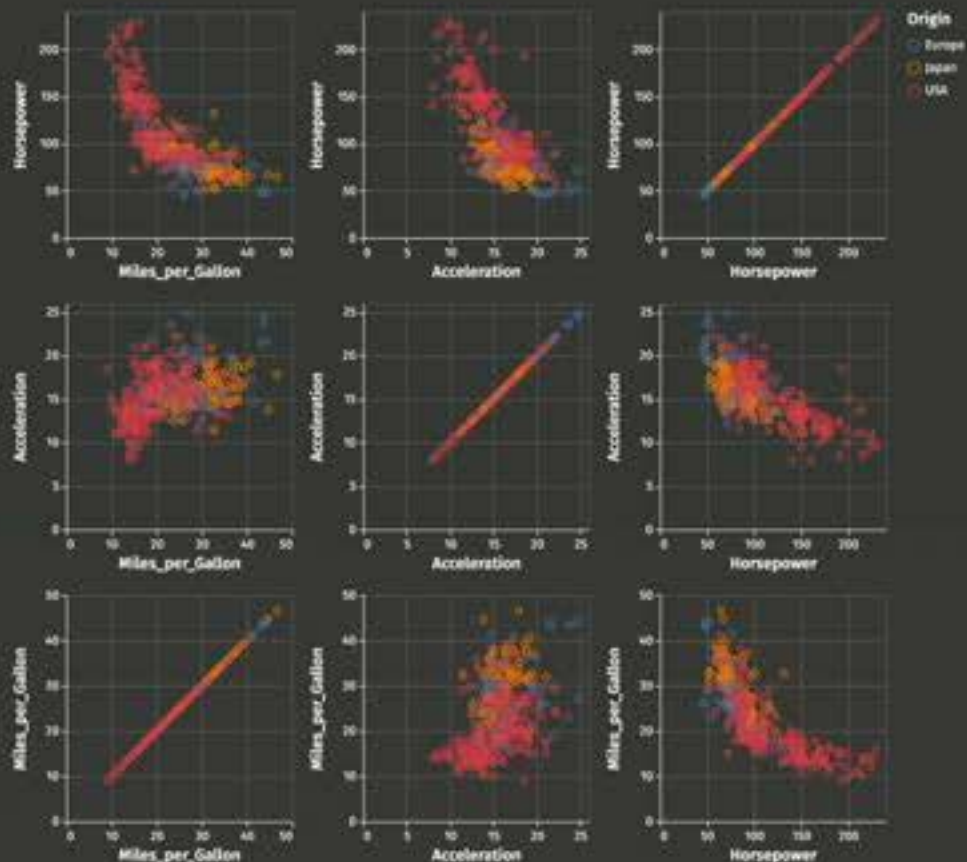
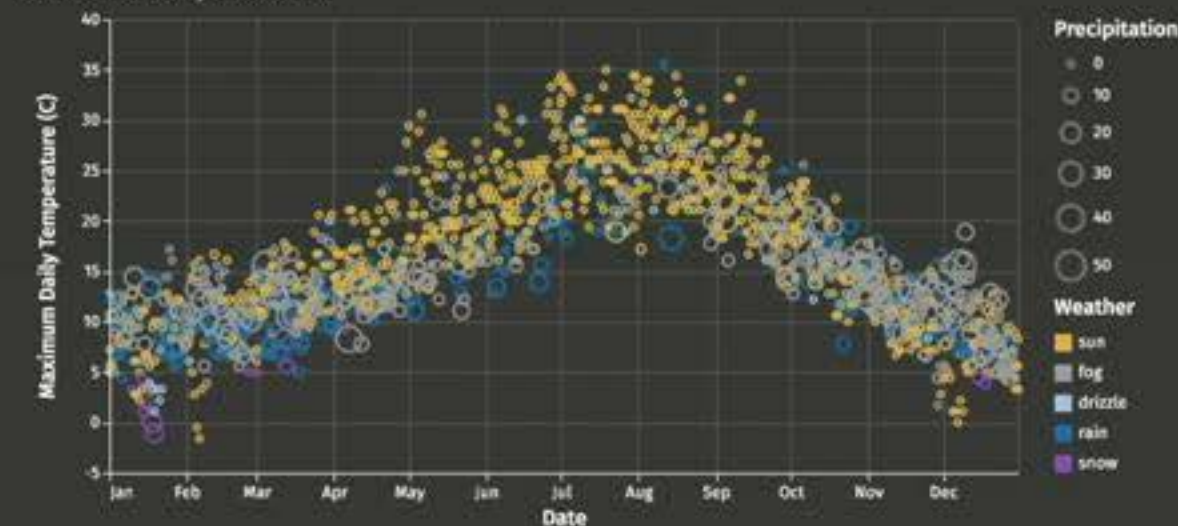
Vega-Lite Encodings

```
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
```

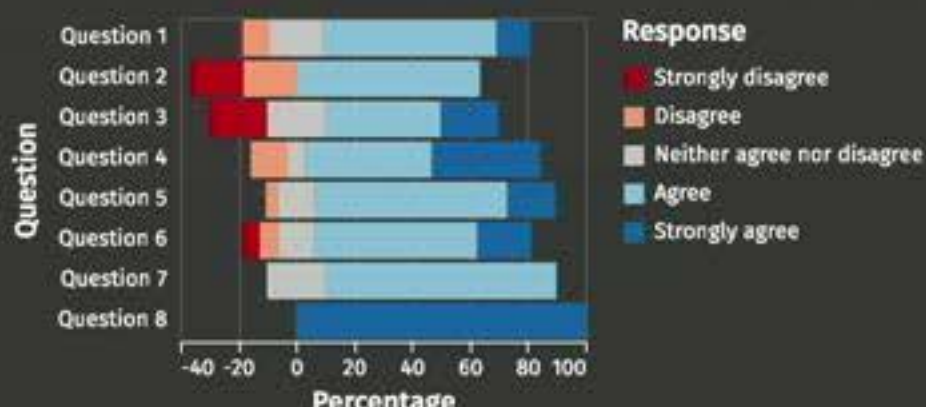


Vega-Lite is an Expressive Language.

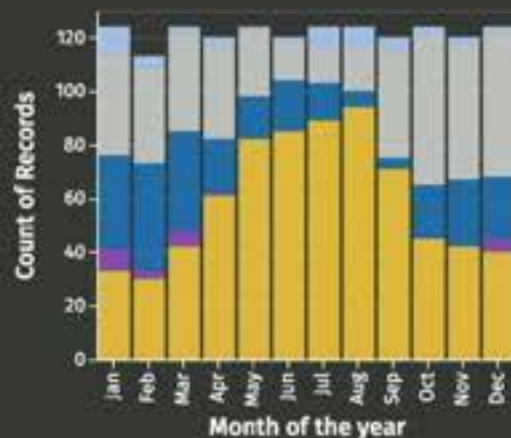
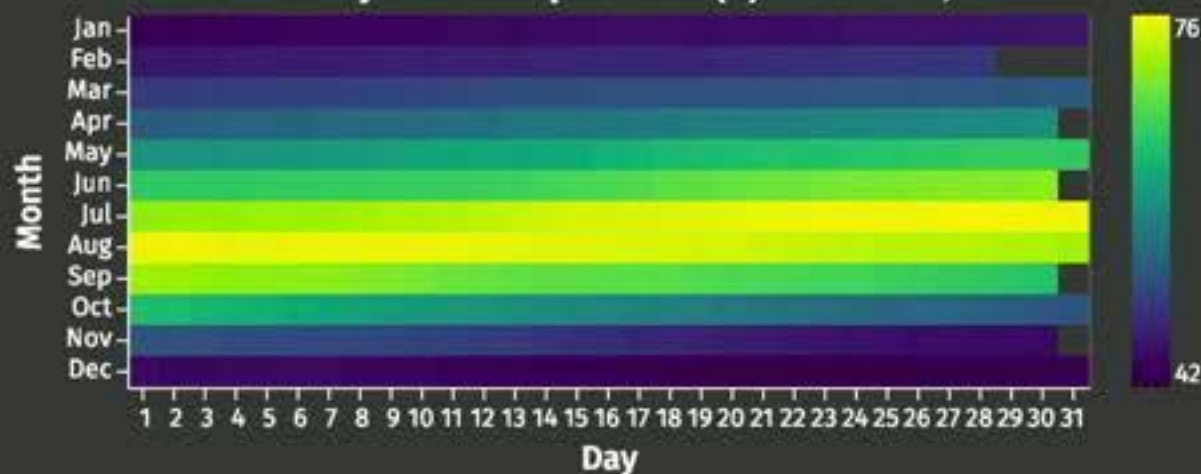
Seattle Weather, 2012-2015



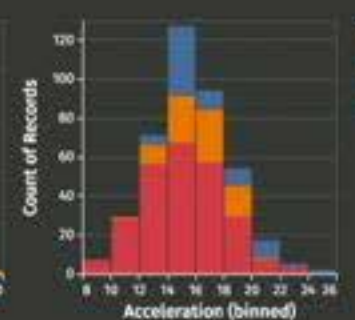
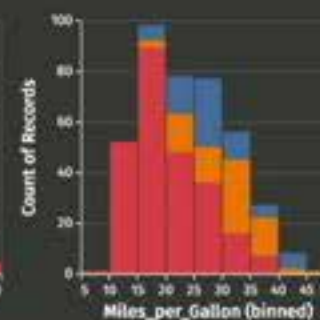
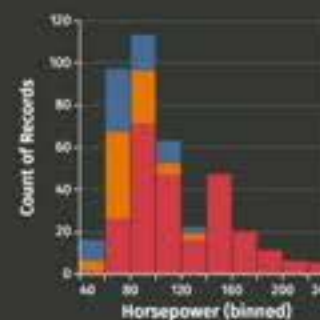
- Agriculture
- Business services
- Construction
- Education and Health
- Finance
- Government
- Information
- Leisure and hospitality
- Manufacturing
- Mining and Extraction
- Other
- Self-employed
- Transportation and Utilities
- Wholesale and Retail Trade



2010 Daily Max Temperature (F) in Seattle, WA

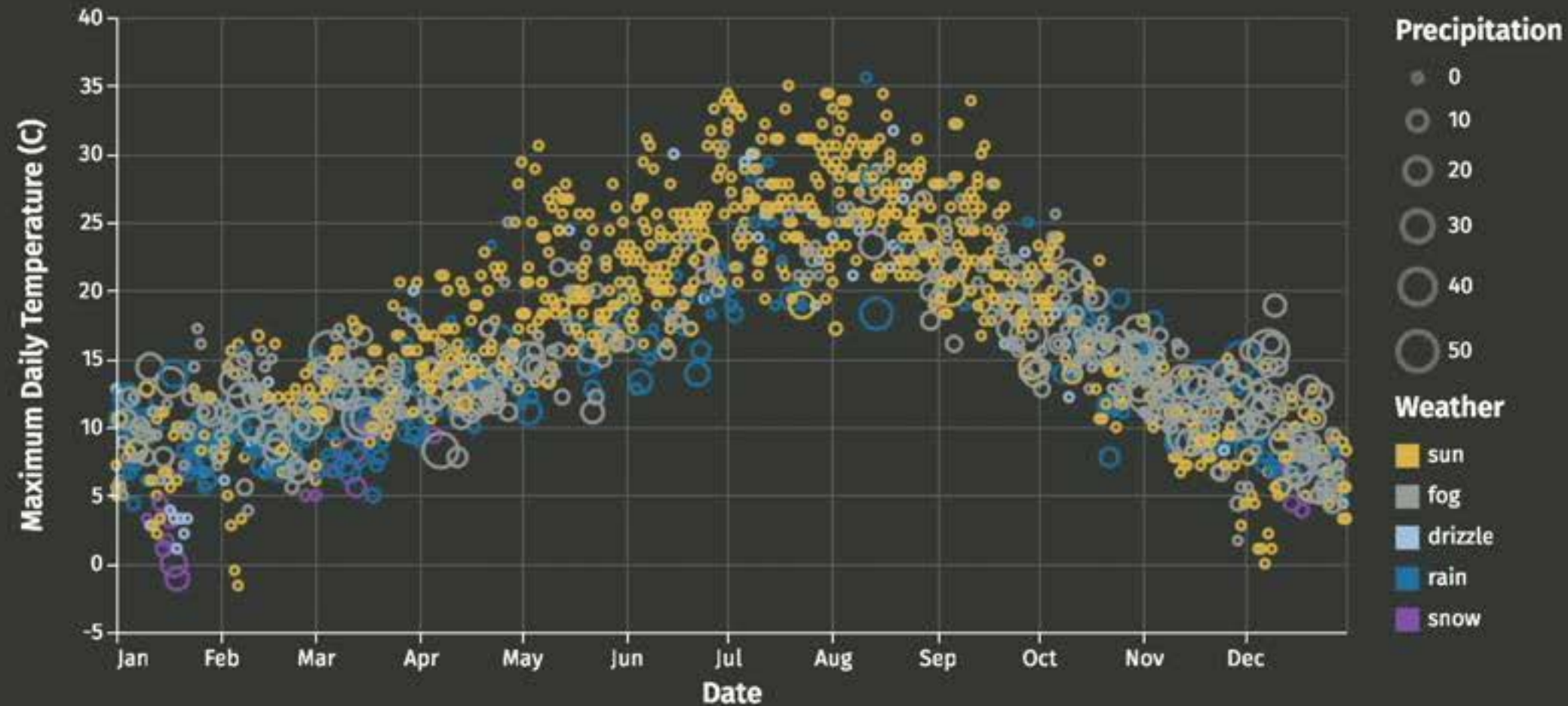


- sun
- fog
- drizzle
- rain
- snow



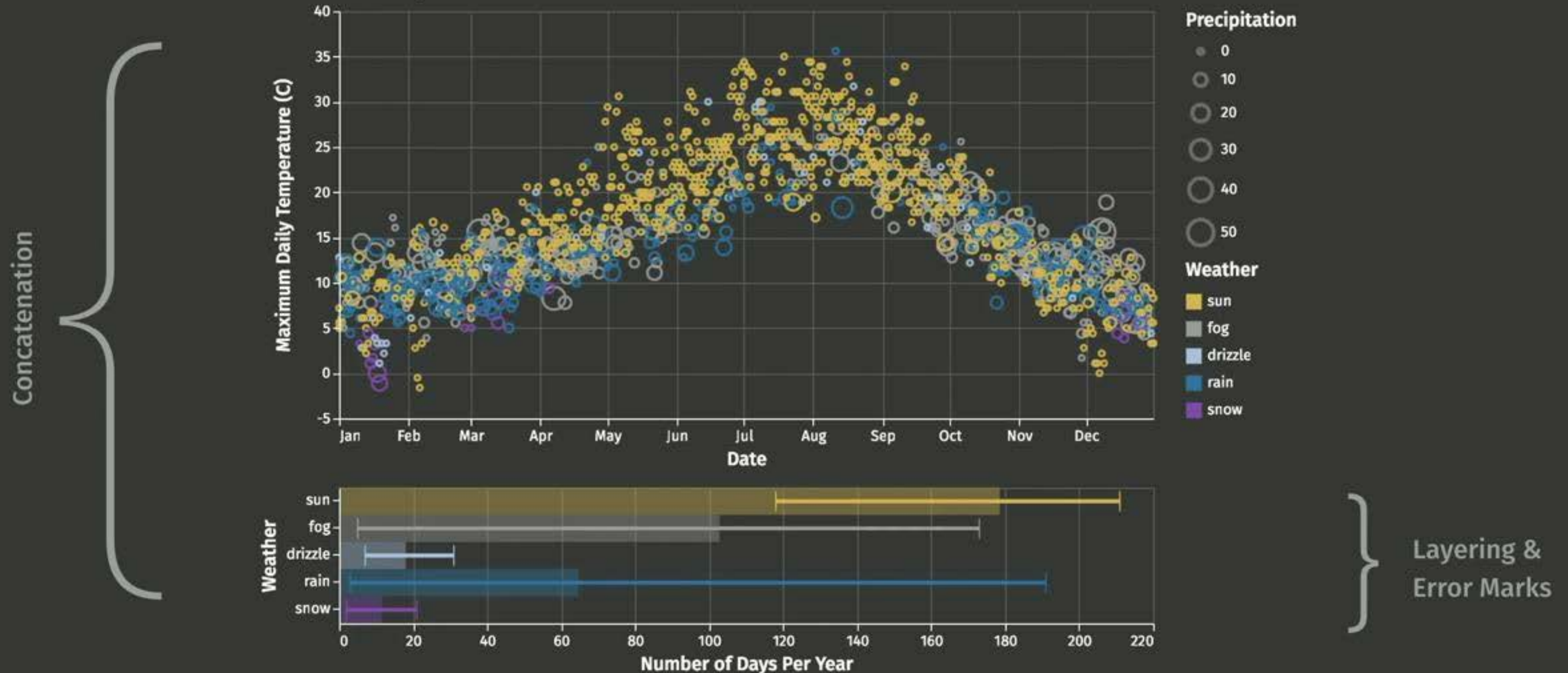
Vega-Lite is an Expressive Language for **Statistical** Graphics.

Seattle Weather, 2012-2015



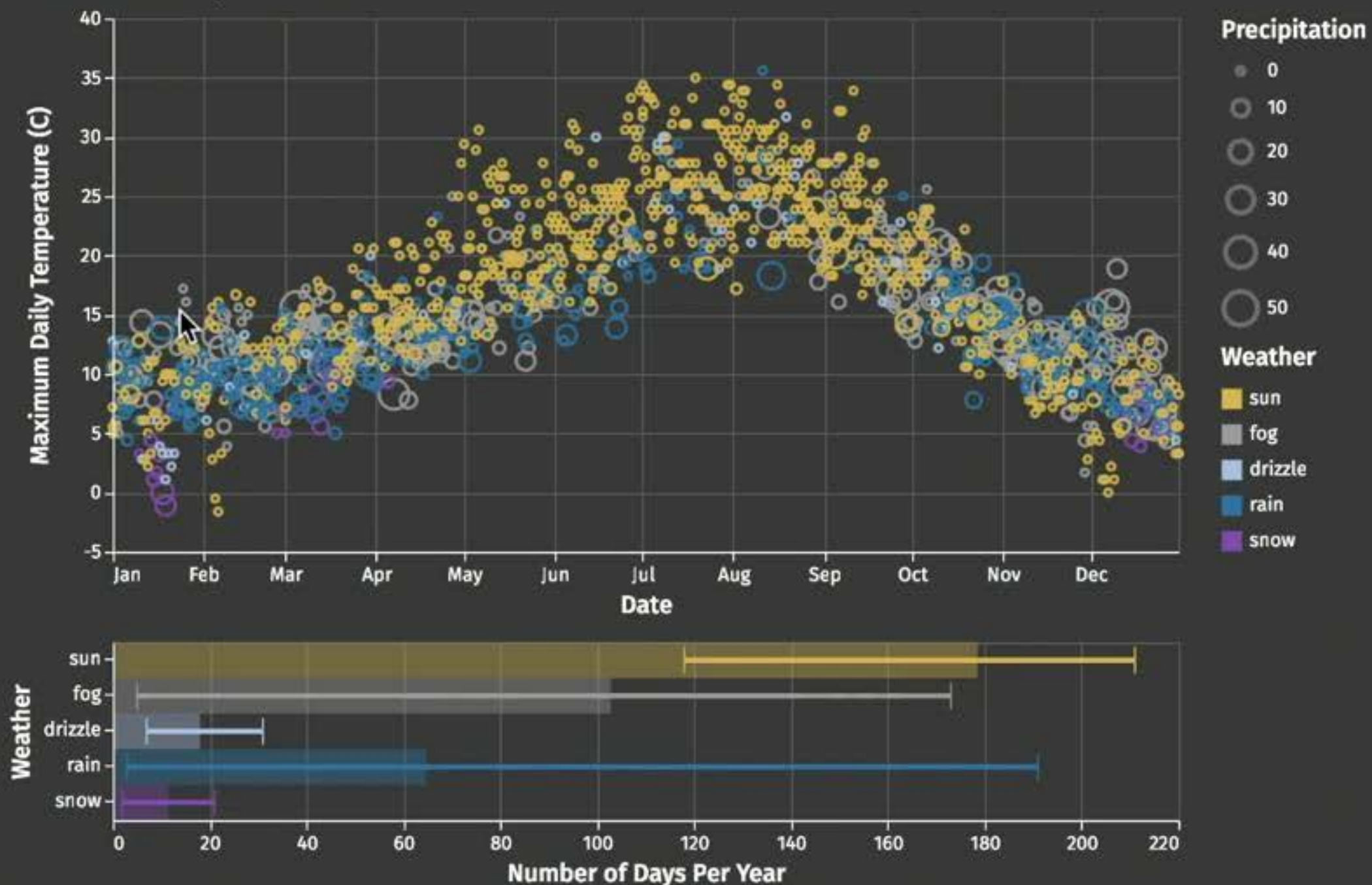
Vega-Lite is an Expressive Language for Statistical **Multi-View** Graphics.

Seattle Weather, 2012-2015



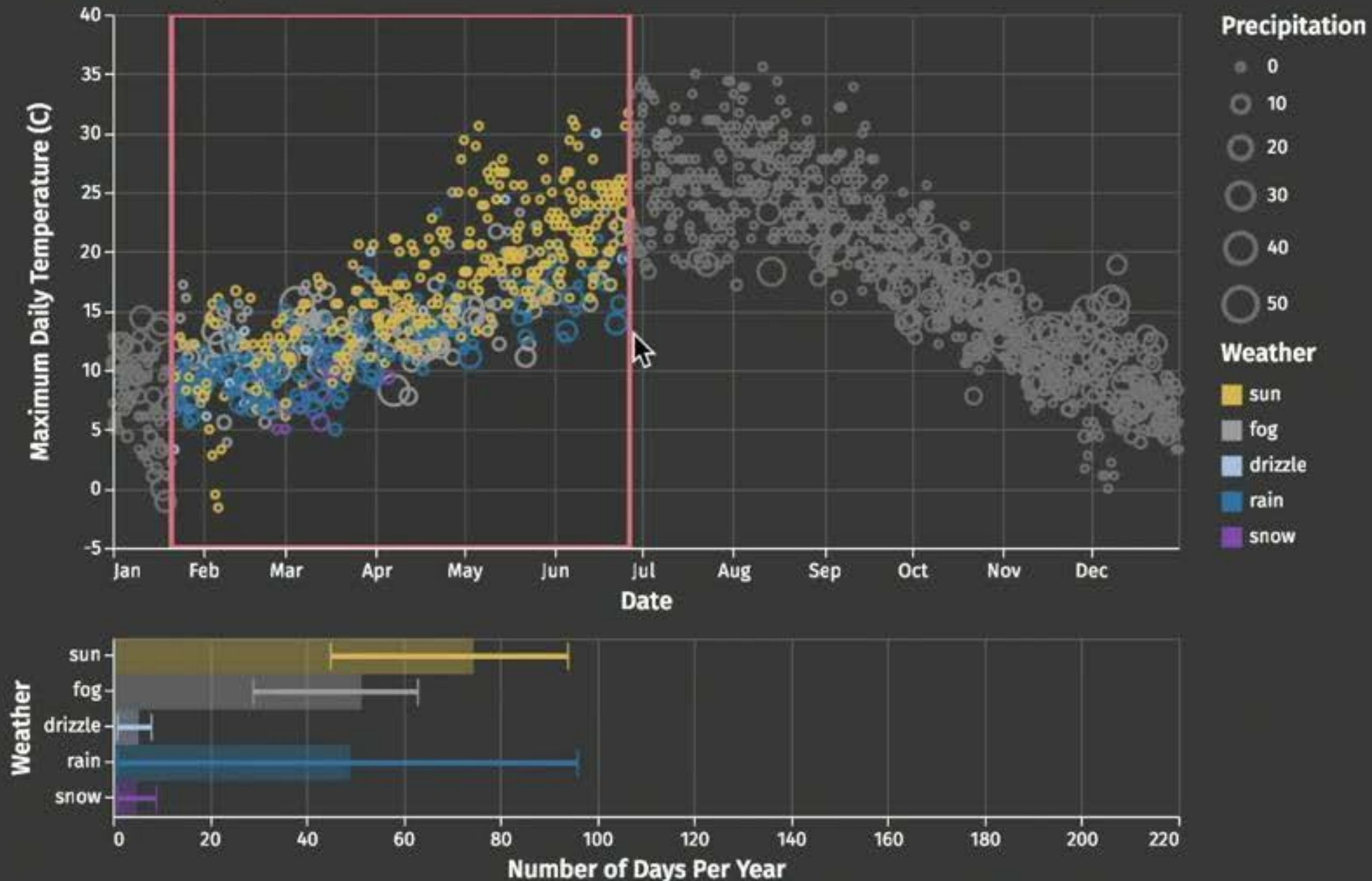
Vega-Lite is an Expressive Language for Statistical **Interactive** Multi-View Graphics.

Seattle Weather, 2012-2015



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

Seattle Weather, 2012-2015

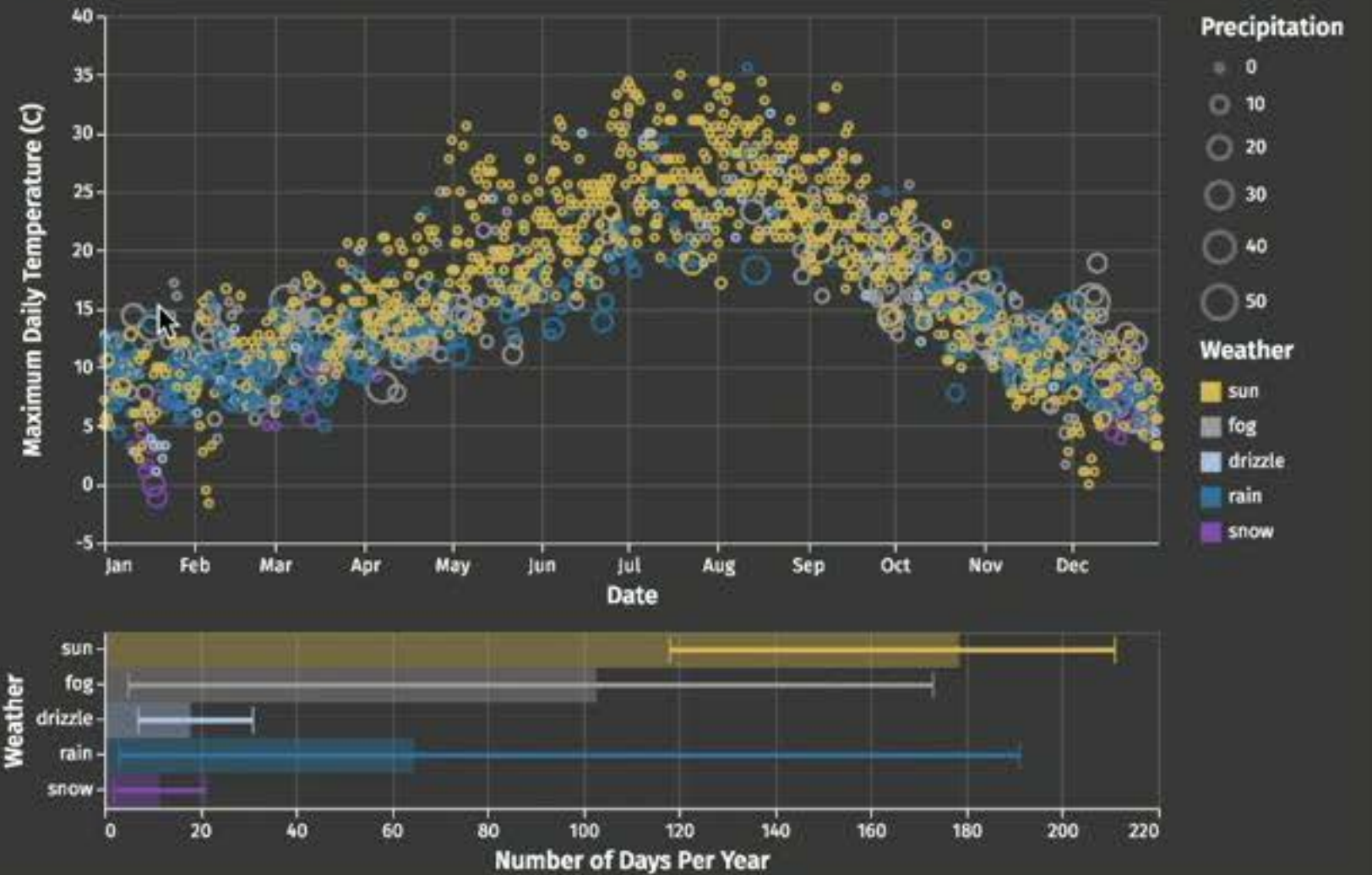


```

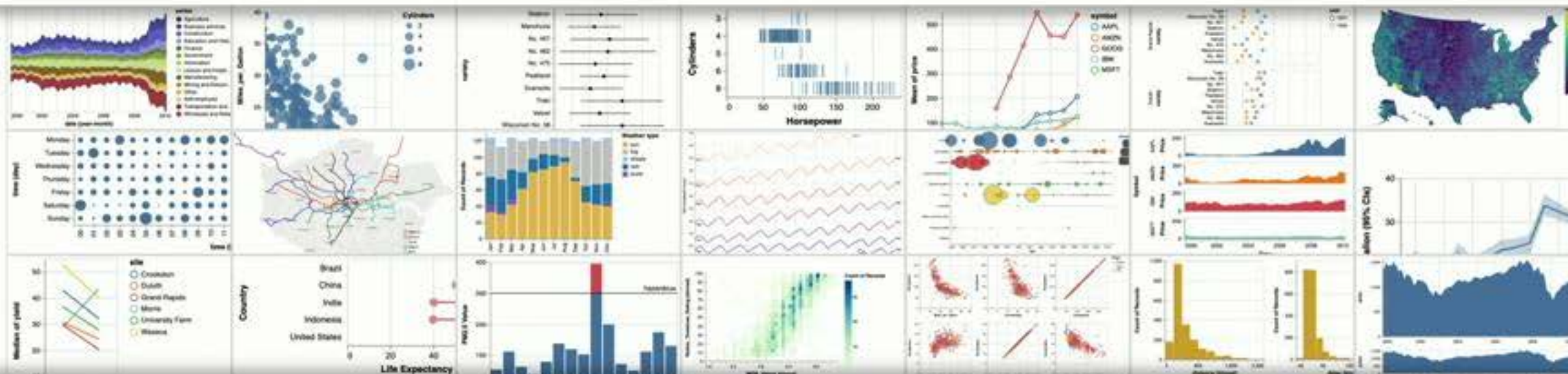
{
  "$schema": "https://vega.github.io/schema/vega-lite/v3.json",
  "title": "Seattle Weather, 2012-2015",
  "data": {
    "url": "data/seattle-weather.csv"
  },
  "vconcat": [
    {
      "width": 600,
      "height": 300,
      "mark": "point",
      "selection": {"brush": {"encodings": ["x"], "type": "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"}
      }
    }
  ]
}

```

Seattle Weather, 2012-2015



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 [introductory](#) documentation

out the

vega.github.io/vega-lite

Example

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

```
{
  "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')
```



Open in Google Colab

goo.gl/6ihGo2

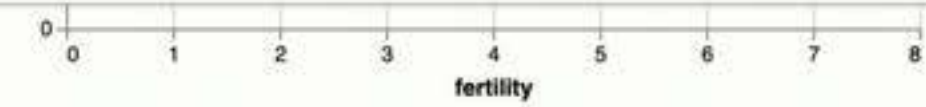
Similar bindings exist for R, Julia, Elm, Scala,...



JupyterLab interface showing a file browser on the left and a notebook editor on the right. The notebook editor displays a Vega chart and its corresponding Altair code.

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 | Python [conda env:altair]



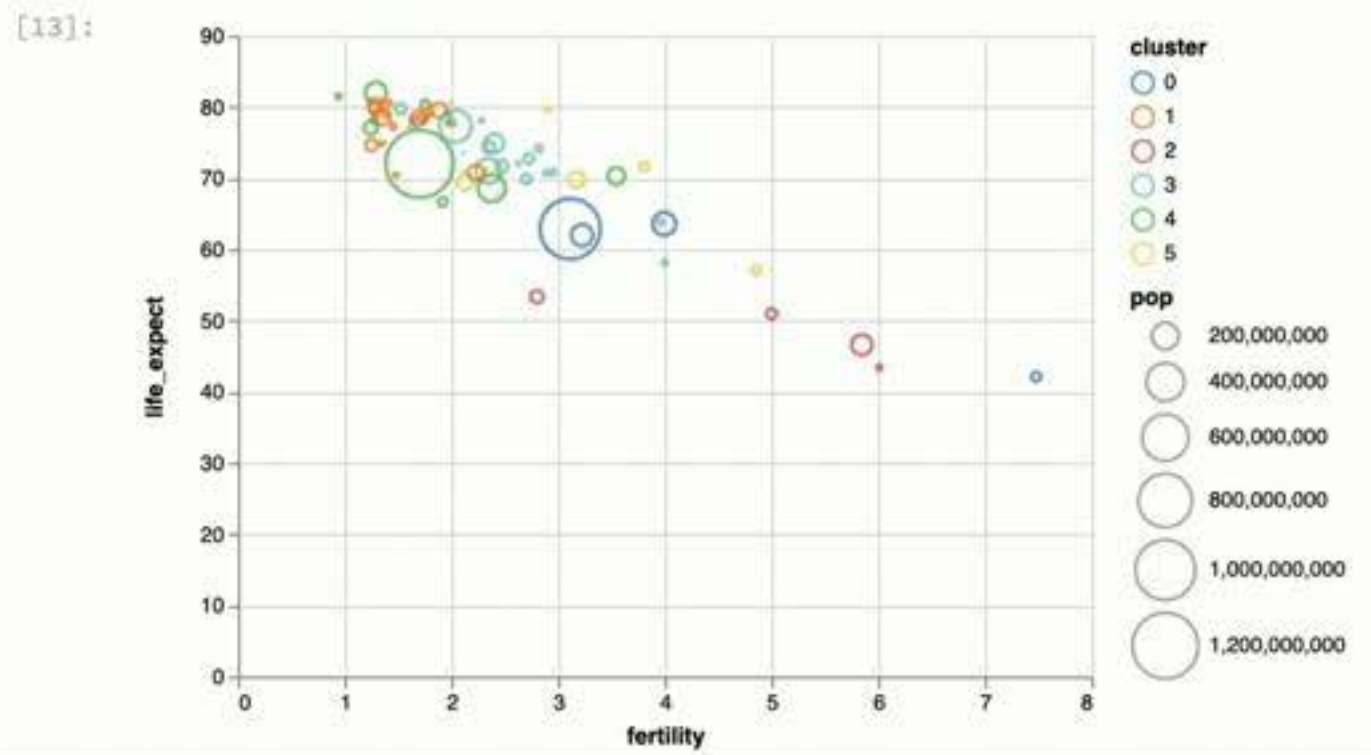
[Export as SVG](#) [Export as PNG](#) [View Source](#) [View Vega](#) [Open in Vega Editor](#)

Color and Opacity

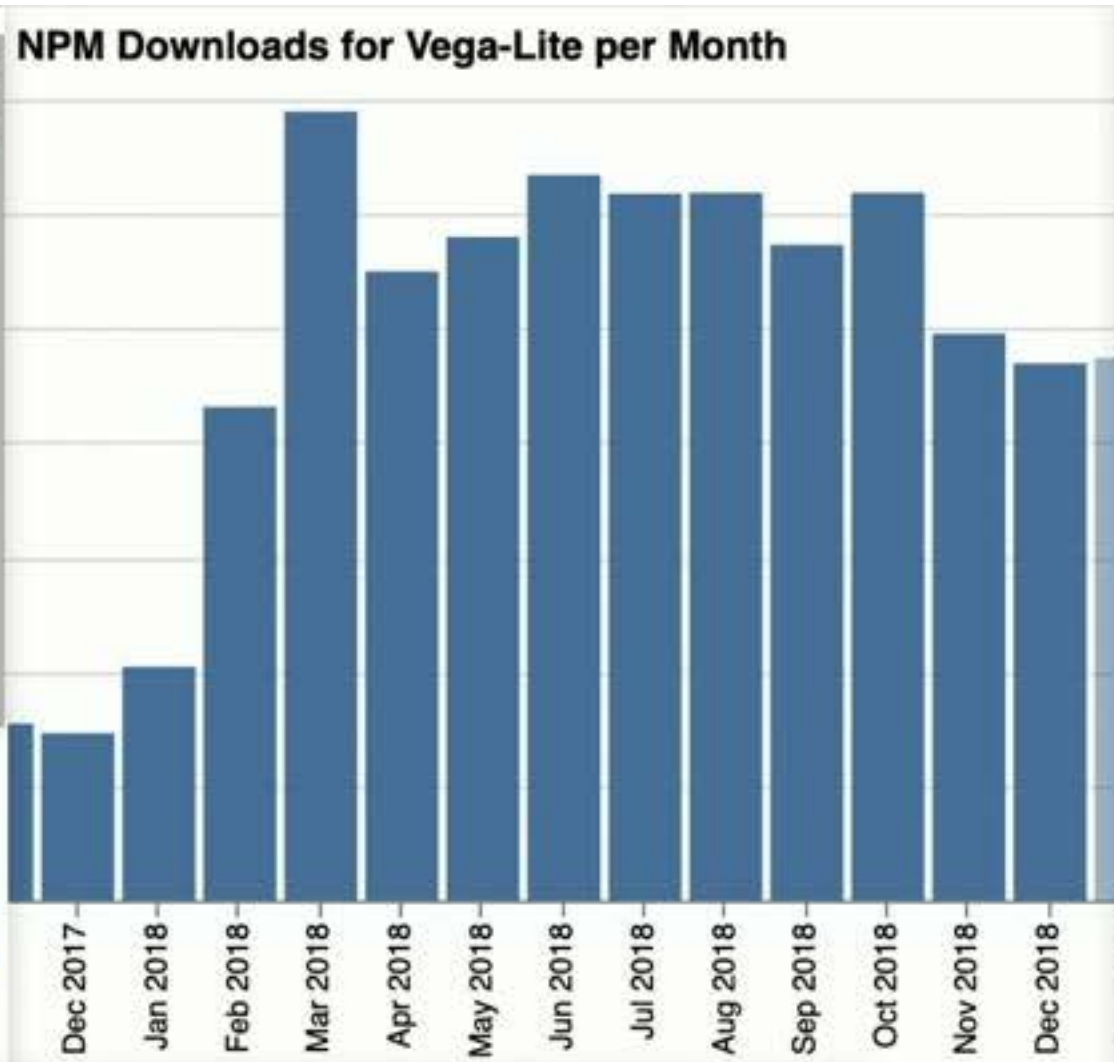
The `color` encoding channel sets a mark's color. The style of color encoding is highly dependent on the data type: nominal data will default to a multi-hued qualitative color scheme, whereas ordinal and quantitative data will use perceptually ordered color gradients.

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')
    )
```



Available as default plotting library in JupyterLab.



Vega-Lite: A Grammar of Interactive Graphics

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

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

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

Index Terms—Information visualization, interaction, systems, toolkits, declarative specification

1 INTRODUCTION

Grammars of graphics span a gamut of expressivity. Low-level grammars such as Protovis [3], D3 [4], and Vega [22] are useful for exploratory data visualization or as a basis for customized analysis tools, as their primitives offer fine-grained control. However, for exploratory visualization, higher-level grammars such as ggplot2 [27] and grammar-based systems such as Tableau (see Polaris [24]), are typically preferred as they favor conciseness over expressiveness. Analysts rapidly author partial specifications of visualizations; the grammar applies default values to resolve ambiguities, and synthesizes low-level details to produce visualizations.

High-level languages can also enable search and inference over the space of visualizations. For example, Wongsuphasawat et al. [30] introduced Vega-Lite to power the Voyager visualization browser. By providing a smaller surface area than the lower-level Vega language, Vega-Lite makes systematic construction and tinkering of data transformations and visual encodings more tractable.

However, existing high-level languages provide limited support for interactivity. An analyst can, at most, enable a predefined set of common techniques (linked selections, panning & zooming, etc.) or parameterize their visualization with dynamic query widgets [21]. For custom, direct manipulation interaction they must instead turn to imperative event handling callbacks. Recognizing that callbacks can be error-prone to author, and require complex static analysis to reason about, Satyanarayan et al. [23] recently formulated declarative interaction primitives for Vega. While these additions facilitate programmatic generation and retargeting of interactive visualizations, they remain low-level. Verbose specification impedes rapid authoring and hinders systematic exploration of alternative designs.

In this paper we extend Vega-Lite to enable concise, high-level specification of interactive data visualizations. To support expressive interaction methods, we first contribute an algebra to compose single-view Vega-Lite specifications into multi-view displays using layer, cross-filter, facet and repeat operators. Vega-Lite's compiler infers how input data should be reused across constituent views, and whether scale domains should be shared or remain independent.

Second, we contribute a high-level interaction grammar. With Vega-Lite, an interaction design is composed of *selections*: visual elements or data points that are chosen when input events occur. Selections parameterize visual encodings by serving as input data, defining scale extents, and providing predicate functions for testing or filtering items. For example, a rectangular “brush” is a common interaction technique for data visualization. In Vega-Lite, a brush is defined as a selection that holds two data points that correspond to its extent (e.g., captured when the mouse button is pressed and as it is dragged, respectively). Its predicate can be used to highlight visual elements that fall within the brushed region, and to materialize a dataset as input to other encodings. The selection can also serve as the scale domain for a secondary view, thereby constructing an overview + detail interaction.

For added expressivity, Vega-Lite provides a series of operators to transform a selection. Transformations can be triggered by input events as well, and manipulate selection points or predicate functions. For example, a brush's predicate will no longer receive a point from the selection

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

Altair is a declarative statistical visualization library for Python. Altair offers a powerful and concise visualization grammar that enables you to build a wide range of statistical visualizations quickly.

By declarative, we mean that you provide a high-level specification of what you want the visualization to include, in terms of data, graphical marks, and encoding channels, rather than having to specify how to implement it in terms of for-loops, low-level drawing commands, etc. The key idea is that you declare links between data fields and visual encoding channels, such as the x-axis, y-axis, color, etc. The rest of the plot details are handled automatically. Building on this declarative plotting idea, a surprising range of simple to sophisticated plots and visualizations can be created using a concise grammar.

Altair is based on [Vega-Lite](#), a high-level grammar of interactive graphics. Altair provides a friendly Python [API](#) *Visualization Programming Interface* that generates Vega-Lite specifications in [JSON](#) *Visual Grammar Object Notation* format. Environments such as Jupyter Notebooks, JupyterLab, and Colab can then take this specification and render it directly in the web browser. To learn more about the motivation and basic concepts behind Altair and Vega-Lite, watch the [Vega-Lite presentation video from DataCamp 2017](#).

This notebook will guide you through the basic process of creating visualizations in Altair. First, you will need to make sure you have the Altair package and its dependencies installed (for more, see the [Altair installation documentation](#)), or you are using a notebook environment that includes the dependencies pre-installed.

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
```

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.

Jeffrey Heer

U.S. Airports, 2008

Flight Routes in USA

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()
```

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

Make the chart.

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')

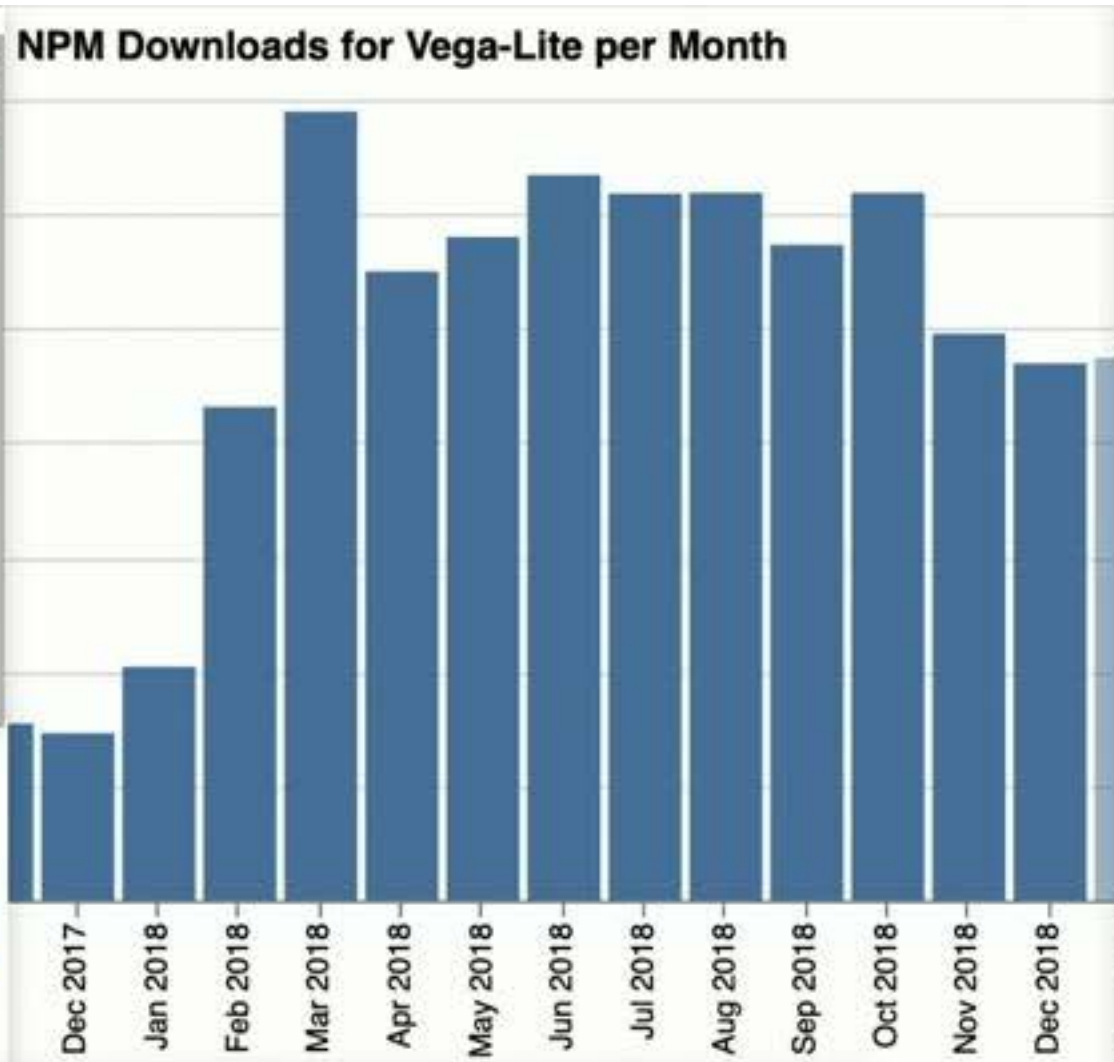
alt
  .Chart(weather)
  .mark_line()
  .encode(
    x=alt.X('date:T', timeUnit='monthdate'),
    y=alt.Y('temp_max:Q', aggregate='mean'),
    color='city:N')
```



Open in Google Colab

goo.gl/6ihGo2

Available as default plotting library in JupyterLab.



Vega-Lite: A Grammar of Interactive Graphics

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

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

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

Index Terms—Information visualization, interaction, systems, toolkits, declarative specification

1 INTRODUCTION

Grammars of graphics span a gamut of expressivity. Low-level grammars such as Protovis [3], D3 [4], and Vega [22] are useful for exploratory data visualization or as a basis for customized analysis tools, as their primitives offer fine-grained control. However, for exploratory visualization, higher-level grammars such as ggplot2 [27], and grammar-based systems such as Tableau (see Polaris [24]), are typically preferred as they favor concision over expressiveness. Analysts rapidly author partial specifications of visualizations; the grammar applies default values to resolve ambiguities, and synthesizes low-level details to produce visualizations.

High-level languages can also enable search and inference over the space of visualizations. For example, Wongsuphasawat et al. [30] introduced Vega-Lite to power the Voyager visualization browser. By providing a smaller surface area than the lower-level Vega language, Vega-Lite makes systematic construction and tinkering of data transformations and visual encodings more tractable.

However, existing high-level languages provide limited support for interactivity. An analyst can, at most, enable a predefined set of common techniques (linked selections, panning & zooming, etc.) or parameterize their visualization with dynamic query widgets [21]. For custom, direct manipulation interaction they must instead turn to imperative event handling callbacks. Recognizing that callbacks can be error-prone to author, and require complex static analysis to reason about, Satyanarayan et al. [23] recently formulated declarative interaction primitives for Vega. While these additions facilitate programmatic generation and retargeting of interactive visualizations, they remain low-level. Verbose specification impedes rapid authoring and hinders systematic exploration of alternative designs.

In this paper we extend Vega-Lite to enable concise, high-level specification of interactive data visualizations. To support expressive interaction methods, we first contribute an algebra to compose single-view Vega-Lite specifications into multi-view displays using *layer*, *concatenate*, *facet* and *repeat* operators. Vega-Lite's compiler infers how input data should be reused across constituent views, and whether scale domains should be shared or remain independent.

Second, we contribute a high-level interaction grammar. With Vega-Lite, an interaction design is composed of *selections*: visual elements or data points that are chosen when input events occur. Selections parameterize visual encodings by serving as input data, defining scale extents, and providing predicate functions for testing or filtering items. For example, a rectangular “brush” is a common interaction technique for data visualization. In Vega-Lite, a brush is defined as a selection that holds two data points that correspond to its extent (e.g., captured when the mouse button is pressed and as it is dragged, respectively). Its predicate can be used to highlight visual elements that fall within the brushed region, and to manipulate a dataset as input to other encodings. The selection can also serve as the scale domain for a secondary view, thereby constructing an overview + detail interaction.

For added expressivity, Vega-Lite provides a series of operators to transform a selection. Transformations can be triggered by input events as well, and manipulate selection points or predicate functions. For example, it needs to be able to add or remove a point from the selection.

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

Altair is a declarative statistical visualization library for Python. Altair offers a powerful and concise visualization grammar that enables you to build a wide range of statistical visualizations quickly.

By declarative, we mean that you provide a high-level specification of what you want the visualization to include, in terms of data, graphical marks, and encoding channels, rather than having to specify how to implement it in terms of for-loops, low-level drawing commands, etc. The key idea is that you declare links between data fields and visual encoding channels, such as the x-axis, y-axis, color, etc. The rest of the plot details are handled automatically. Building on this declarative plotting idea, a surprising range of simple to sophisticated plots and visualizations can be created using a concise grammar.

Altair is based on Vega-Lite, a high-level grammar of interactive graphics. Altair provides a friendly Python API, *Visualization Programming Interface* that generates Vega-Lite specifications in JSON (Vega-Spec Object Notation) format. Environments such as Jupyter Notebooks, JupyterLab, and Colab can then take this specification and render it directly in the web browser. To learn more about the motivation and basic concepts behind Altair and Vega-Lite, watch the Vega-Lite introduction video from DataCamp [20].

This notebook will guide you through the basic process of creating visualizations in Altair. First, you will need to make sure you have the Altair package and its dependencies installed (for more, see the [Altair installation documentation](#)), or you are using a notebook environment that includes the dependencies pre-installed.

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 Gernuska.

```
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()
```

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

Make the chart.

Data visualization tools drive interactivity and reproducibility in online publishing

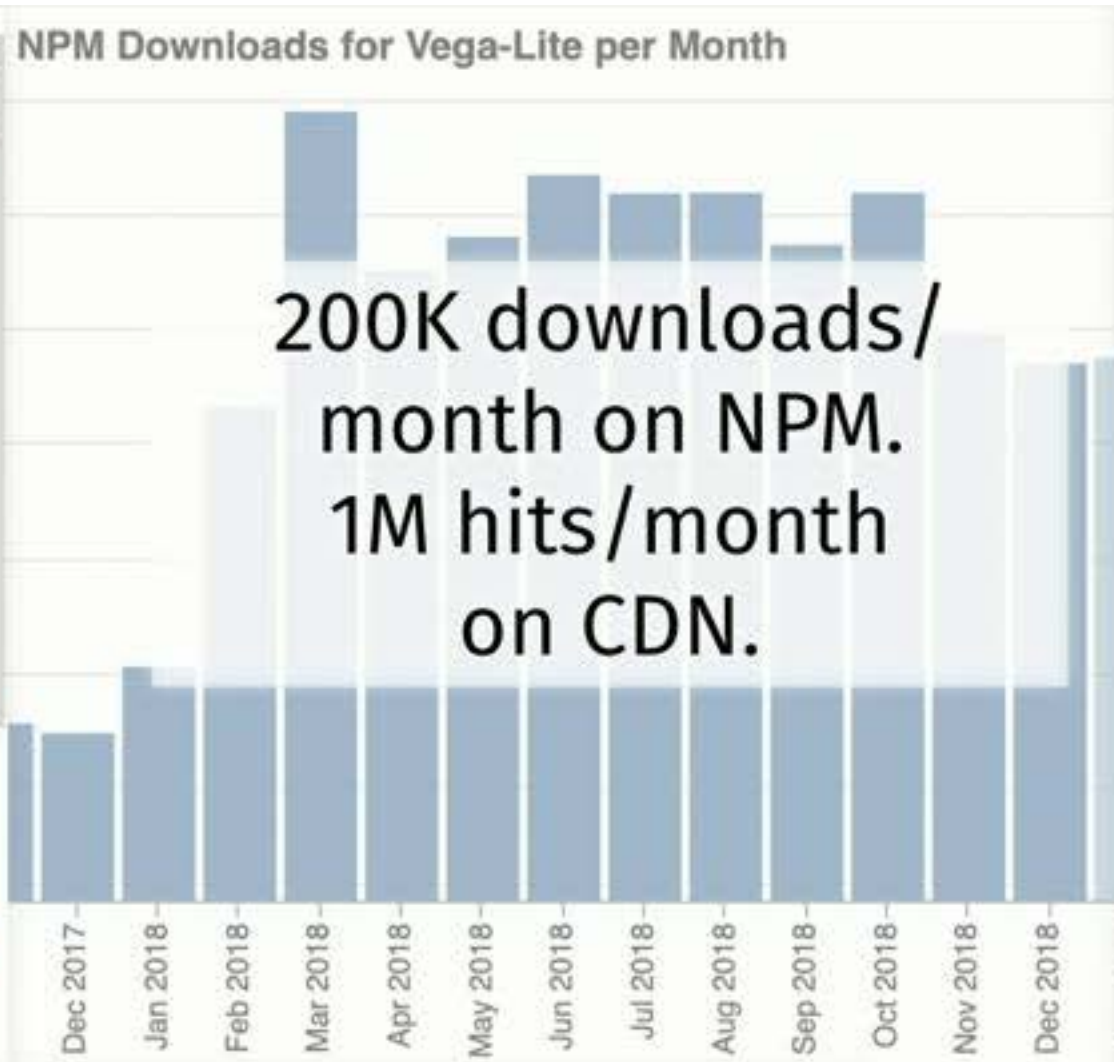
New tools for building interactive figures and software make scientific data more accessible, and reproducible.

Jeffrey H. Heer

U.S. Airports, 2008

Flight Routes in USA

Available as default plotting library in JupyterLab.



Vega-Lite: A Grammar of Interactive Graphics

Arvind Satyanarayan, Dominik Moritz, Karri Wingsughasawat, and Jeffrey Heer

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

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

Index Terms—Information visualization, interaction, systems, tactics, declarative specification

1 INTRODUCTION

Grammars of graphics open a portal of expressivity. Low-level grammars such as Protovis [8], D3.js [4], and Vega [22] are useful for exploratory data visualization or as a basis for customized analysis tools, as their primitives offer fine-grained control. However, for exploratory visualization, higher-level grammars such as ggplot2 [17], and grammar-based systems such as Tableau (see Polaris [24]), are typically preferred as they force constraints over expressions. Analysts rapidly author partial specifications of visualizations; the grammar applies default values to resolve ambiguities, and synthesizes low-level details to produce visualizations.

High-level languages can also enable search and reference over the space of visualizations. For example, Wingsughasawat et al. [30] introduced Vega-Lite to power the Voyager visualization browser. By providing a sparser surface area than the lower-level Vega language, Vega-Lite makes systematic exploration and linking of data transformations and visual encodings more tractable.

However, existing high-level languages provide limited support for interactivity. An analyst can, at most, enable a predefined set of common techniques (linked selections, panning & zooming, etc.) or parameterize their visualizations with dynamic query widgets [21]. In contrast, direct manipulation interactions rely on central hints or imperative event handling callbacks. Recognizing that callbacks can be time-consuming to author, and require complex state analysis to manage down, Satyanarayan et al. [27] recently formalized declarative interaction primitives for Vega. While these additions facilitate programmatic generation and retargeting of interactive visualizations, they remain low-level. Without specification impeding rapid authoring and hindering systematic exploration of alternative designs.

In this paper we extend Vega-Lite to enable concise, high-level specifications of interactive data visualizations. To support expressive interaction methods, we first contribute an algebra to compose single-view Vega-Lite specifications into multi-view displays using layer, containment, layer and event operations. Vega-Lite's compiler infers how input data should be reused across containment views, and whether scale domains should be shared or remain independent.

Second, we contribute a high-level interaction grammar. With Vega-Lite, an interaction design is composed of selections: visual encodings or data points that are chosen when input events occur. Selections parameterize visual encodings by serving as input data, defining scale domains, and providing predicate functions for testing or filtering items. For example, a rectangular "brush" is a common interaction technique for data visualizations. In Vega-Lite, a brush is defined as a selection that binds two data points that correspond to its extent (e.g., captured when the mouse button is pressed and as it is dragged, respectively). Its predicate can be used to highlight visual elements that fall within the brushed region, and to reconfigure a dataset to input to other encodings. The operation can also serve as the scale domain for a secondary view, thereby constructing an overview + detail interaction.

For added expressivity, Vega-Lite provides a series of operators to manage a selection. Transitions can be triggered by input events as well, and composite selection points or predicate functions. For example, a brush's transition, which can reconfigure a brush from the selection

Notebooks

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

Introduction to Altair / Vega-Lite

Altair is a declarative statistical visualization library for Python. Altair offers a powerful and concise visualization grammar that enables you to build a wide range of statistical visualizations quickly.

By declarative, we mean that you provide a high-level specification of what you want the visualization to exhibit, in terms of data, graphical marks, and encoding channels, rather than having to specify how to implement it in terms of low-level plotting statements, etc. The key idea is that you declare links between data fields and visual encoding channels, such as the marks, scales, axes, etc. The rest of the plot details are handled automatically, building on the declarative plotting idea, a surprising range of simple to sophisticated plots and visualizations can be created using a concise grammar.

Altair is based on Vega-Lite, a high-level grammar of interactive graphics. Altair provides a friendly Python API, supporting Python's idiomatic style that generates Vega-Lite specifications in JSON (also called Domain/Specification format). Environments such as Jupyter Notebooks, JupyterLab, and Colab can then take this specification and render it directly in the web browser. To learn more about the motivation and basic concepts behind Altair and Vega-Lite, watch the Vega-Lite presentation video from DataCamp [20].

This notebook will guide you through the basic process of creating visualizations in Altair. First, you will need to make sure you have the Altair package and its dependencies installed (for more, see the [DataCamp installation instructions](#)), or you are using a notebook environment that includes the dependencies pre-installed.

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 West

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 Gernuska.

```
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('inputs/mhhi_by_county.csv')
```

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

Unnamed: 0	name	geoid	b19013001
0	Anchorage Municipality, Alaska	05000US02020	85634.0
1	Fairbanks North Star Borough, Alaska	05000US02090	77326.0
2	Matanuska-Sustina Borough, Alaska	05000US02170	69332.0
3	Baldwin County, Alabama	05000US01003	56732.0
4	Calhoun County, Alabama	05000US01015	41687.0

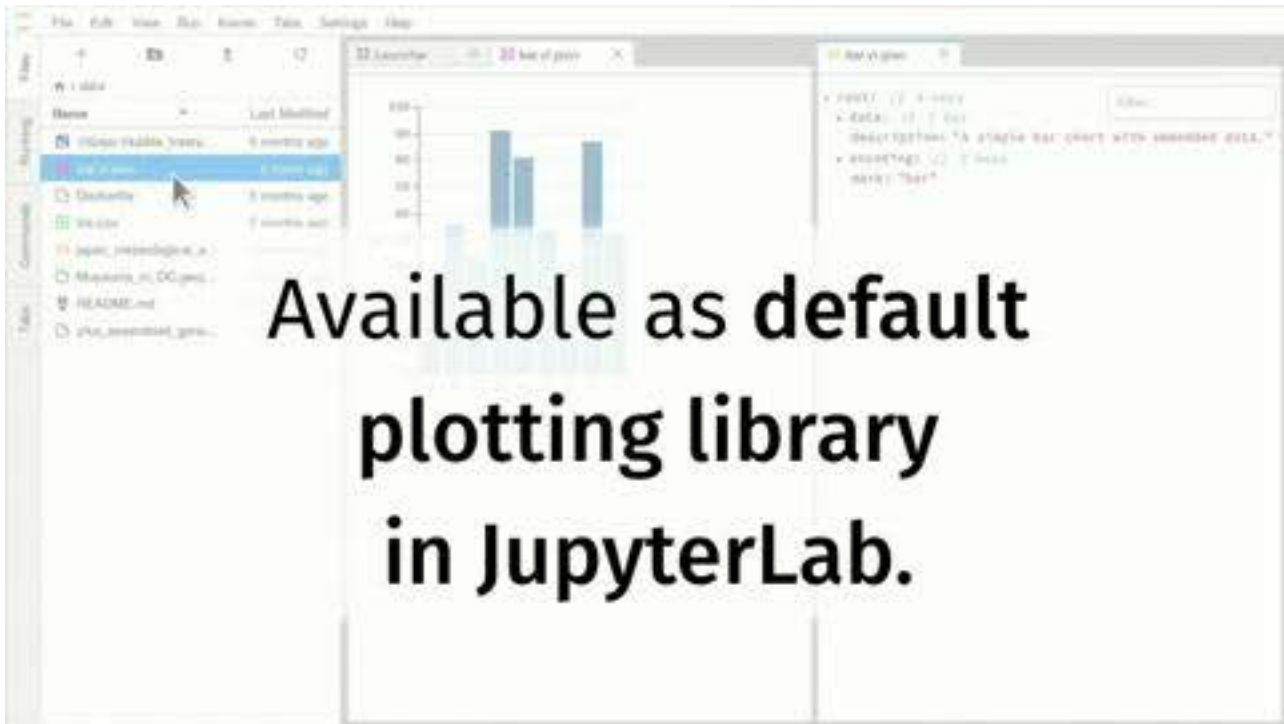
Make the chart.

Data visualization tools drive interactivity and reproducibility in online publishing

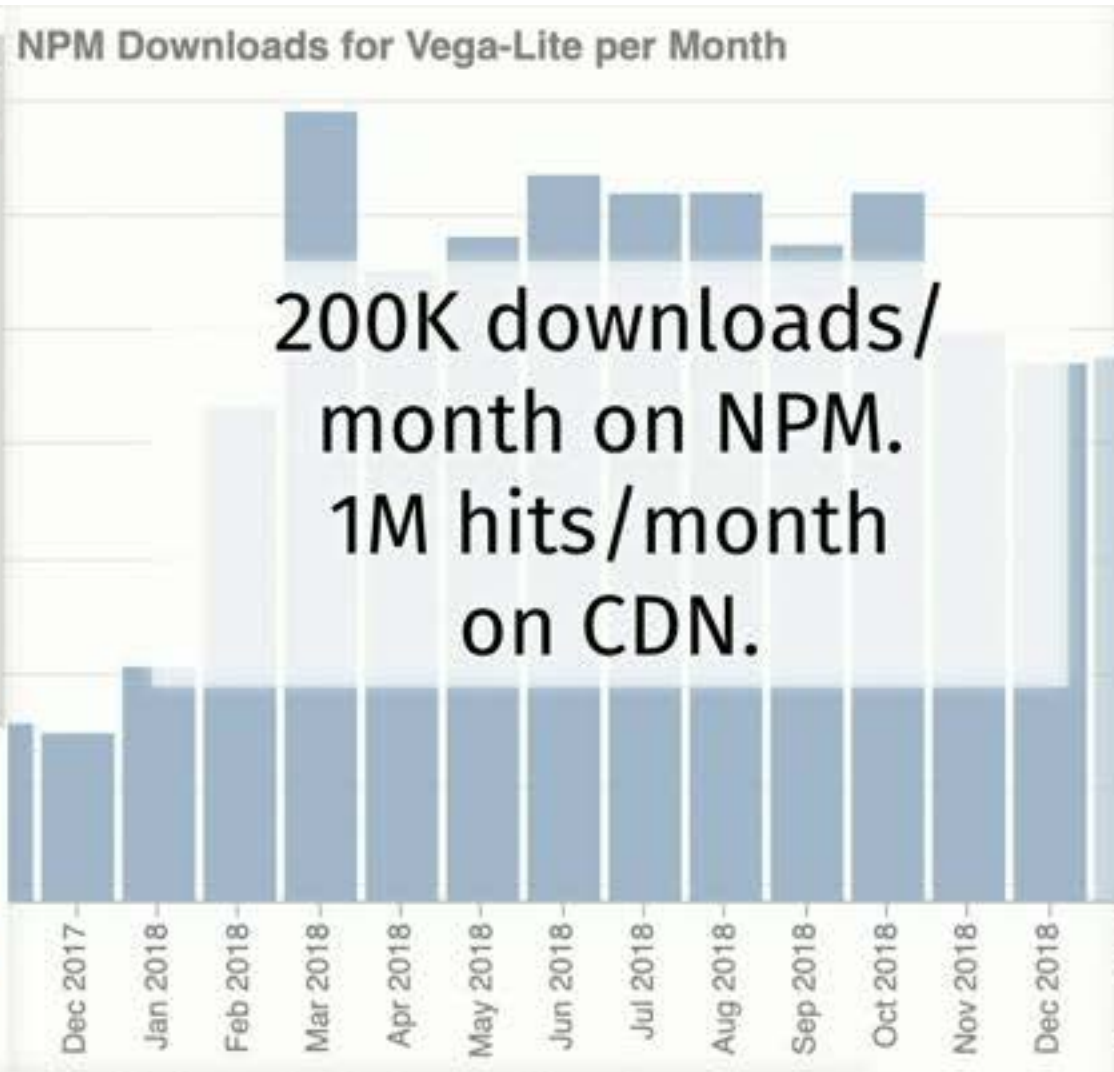
New tools for building interactive figures and software make scientific data more accessible and reproducible.

U.S. Airports, 2008

Flight Routes in USA



Available as default plotting library in JupyterLab.



Vega-Lite: A Grammar of Interactive Graphics

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

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

and grammar-based systems such as Tableau (see Polaris [24]), are typically performed in their own containers over experiments. Analysis rapidly author partial specifications of visualizations; the grammar applies default values to resolve ambiguities, and synthesizes low-level details to produce visualizations.

High-level languages can also enable search and inference over the space of visualizations. For example, Wongsuphasawat et al. [30] introduced Vega-Lite to power the Vizgo visualization browser. By providing a spatial surface area above the lower-level Vega language, Vega-Lite makes systematic enumeration and ranking of data transformations and visual encodings more tractable.

However, existing high-level languages provide limited support for interactivity. An analyst can, at most, enable a predefined set of common technique-linked selections, panning & zooming, etc., or parameterize their visualizations with dynamic query widgets [21]. In contrast, direct manipulation interactions rely on manual hints to interpret event handling (clicks). Recognizing that callbacks can be time-prohibitive to author, and support complex state analysis to make them doable, Satyanarayan et al. [27] recently formalized declarative interaction primitives for Vega. While these additions facilitate programmatic generation and reinteracting of interactive visualizations, they remain

interaction primitives, not full containers, an algorithm to compress single-view Vega-Lite specifications into multi-view displays using lens, zoom, pan, and other operations. Vega-Lite's compiler infers how input data should be mapped across component views, and whether scale domains should be shared or remain independent.

Second, we envision a high-level interactive grammar. With Vega-Lite, an interaction design is composed of declarative visual encodings or data points that are chosen when input events occur. Selections parameterize visual encodings by serving as input data defining scale domains, and providing predicate functions for filtering or filtering events. For example, a rectangular "brush" is a common interaction technique for data visualization. In Vega-Lite, a brush is defined as a selection that binds two data points that correspond to an event (e.g., captured when the mouse button is pressed) and an id or dragged, respectively. To predicate can be used to highlight visual elements that fall within the brush's region, and to minimize a dataset to help in other encodings. The selection can also serve as the wide domain for a secondary view, thereby connecting an overview + detail interaction.

For added expressivity, Vega-Lite provides a series of operators to compose a selection. Transitions can be triggered by input events as well, and aggregate selection points or predicate functions. For example, a brush's transition, with an associated action from the collection

Notebooks

- Introduction [Open in CodeSandbox](#)
- Marks and Encodings [Open in CodeSandbox](#)
- Homework: Basic Chart Building [Open in CodeSandbox](#)
- Data Transformations [Open in CodeSandbox](#)
- Visual Data Analysis [Open in CodeSandbox](#)
- Homework: EDA [Open in CodeSandbox](#)
- Scales, Axes, Legends [Open in CodeSandbox](#)
- Layering and Composition [Open in CodeSandbox](#)
- Homework: Multi-View Composition [Open in CodeSandbox](#)
- Selections [Open in CodeSandbox](#)
- Visual Data Analysis 2 [Open in CodeSandbox](#)
- Homework: Interactive Dashboard [Open in CodeSandbox](#)

Introduction to Altair / Vega-Lite

Vega is a declarative grammar visualization library for Python. Vega offers a powerful and flexible visualization grammar that enables you to build a wide range of statistical visualizations quickly.

By declarative, we mean that you provide a high-level specification of what you want the visualization to include, in terms of data, graphical marks, and encoding channels, rather than having to specify how to implement it in terms of for-loops, low-level drawing statements, etc. The key idea is that you declare links between data fields and visual encoding channels, such as We want, year, count, etc. The rest of the job details are handled automatically. Building on this declarative parsing idea, a surprising range of scripts to sophisticated plots and visualizations can be created using a concise grammar.

Altair is based on Vega-Lite, a high-level grammar of interactive graphics. Altair provides a friendly Python API, accessible through JupyterLab, and CodeSandbox. You can also use the Vega-Lite specification and render it directly in the web browser. To learn more about the motivation and basic concepts behind Altair and Vega-Lite, watch the Vega-Lite presentation video from DataCamp.

This notebook will guide you through the basic process of creating visualizations in Altair. First, you will need to make sure you have the Altair package and its dependencies installed (for more, see the [DataCamp documentation](#)), or you are using a notebook environment that includes the dependencies pre-installed.

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 Washi

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 Gernuska.

```
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()
```

Unnamed: 0	name	geoid	b19015001
0	Anchorage Municipality, Alaska	05000US02020	85634.0
1	Fairbanks North Star Borough, Alaska	05000US02090	77328.0
2	Matanuska-Susitna Borough, Alaska	05000US02170	69332.0
3	Baldwin County, Alabama	05000US01003	56732.0
4	Calhoun County, Alabama	05000US01015	41687.0

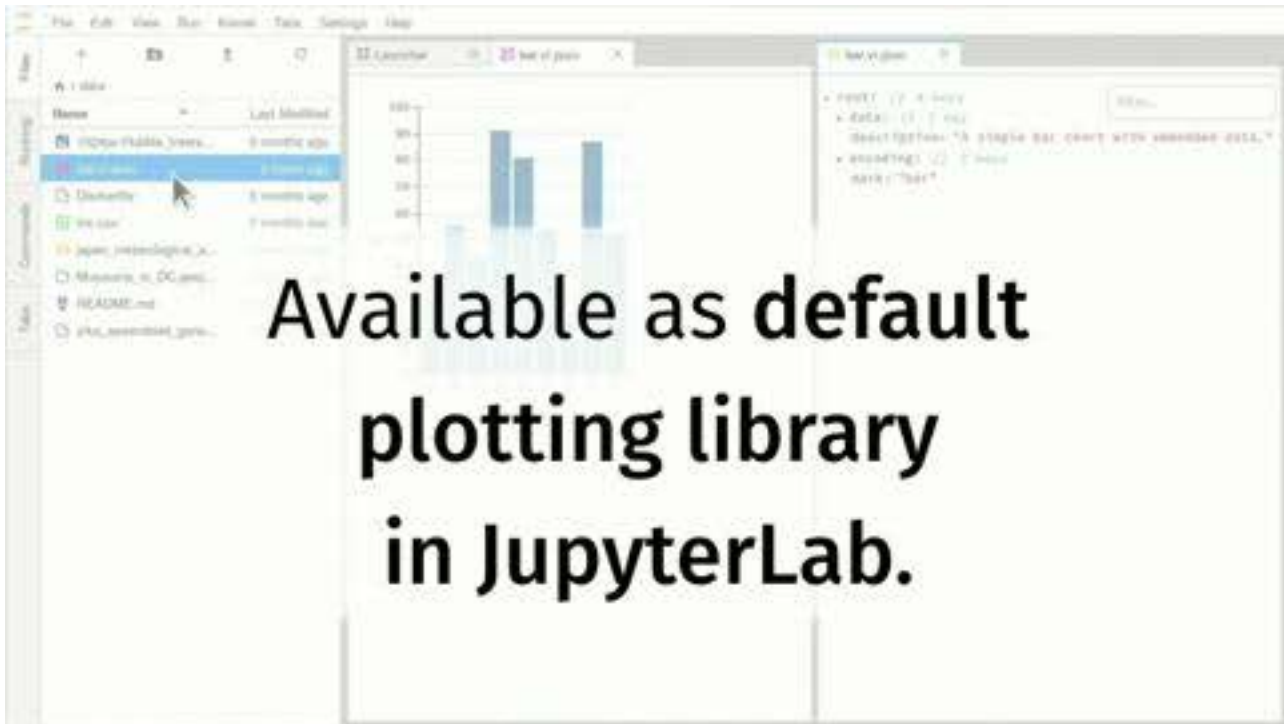
Make the chart.

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.

U.S. Airports, 2008

Flight Routes in USA



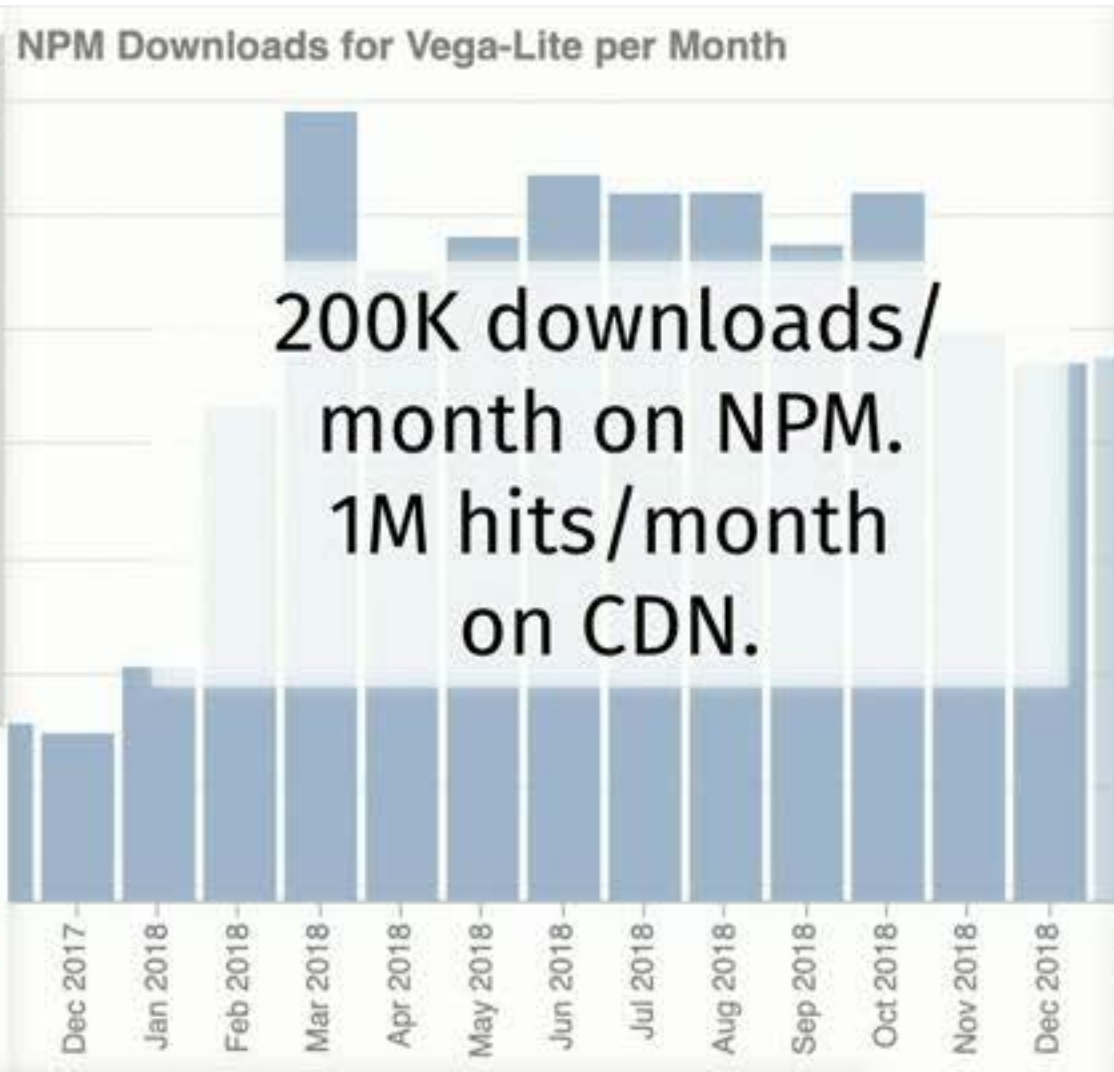
Available as default plotting library in JupyterLab.



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



Used by the LA Times



"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

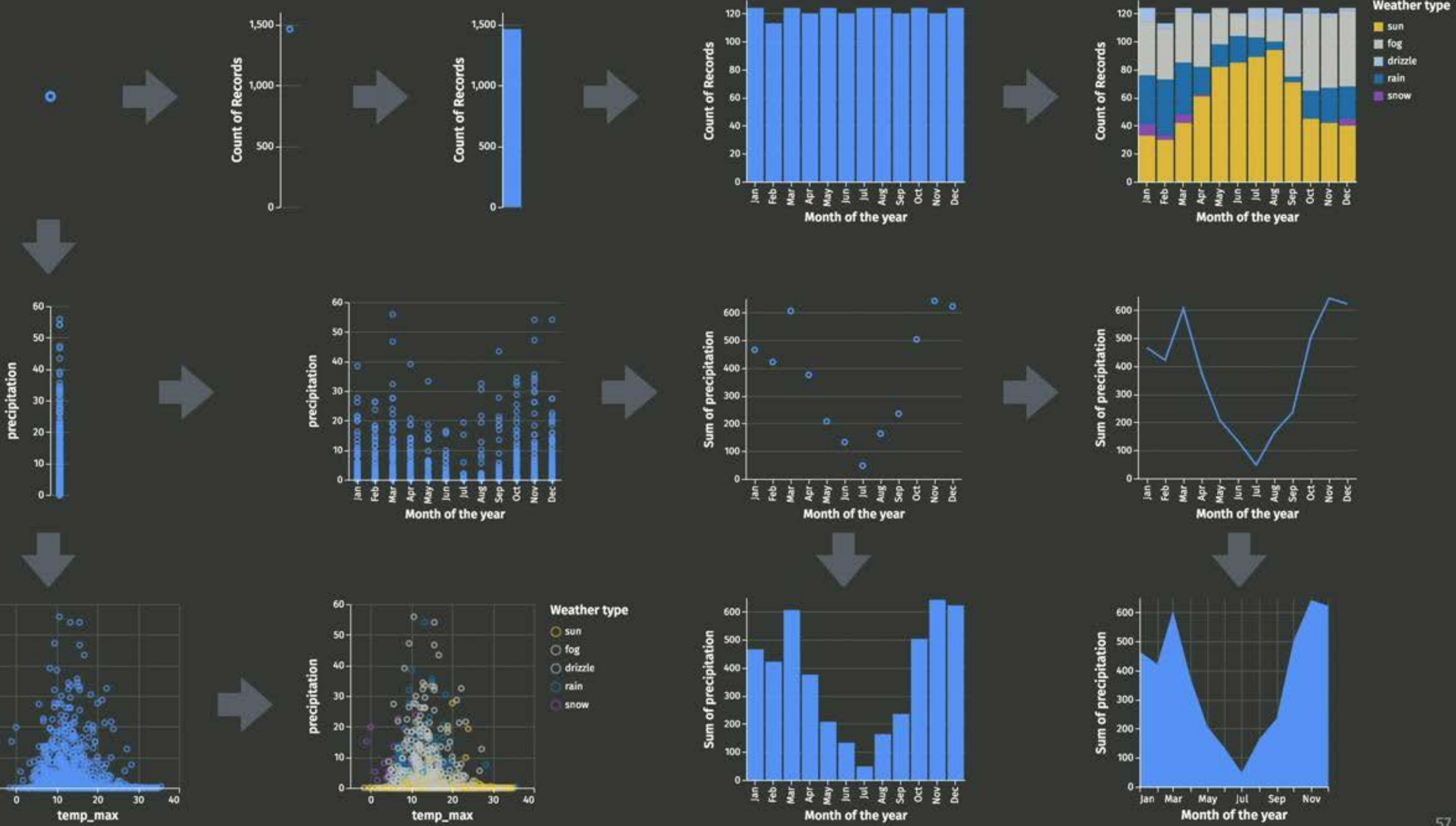


Designed for programmatic generation

Declarative: specification decoupled from execution

High-level, domain-specific abstractions

Composable building blocks





Draco

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

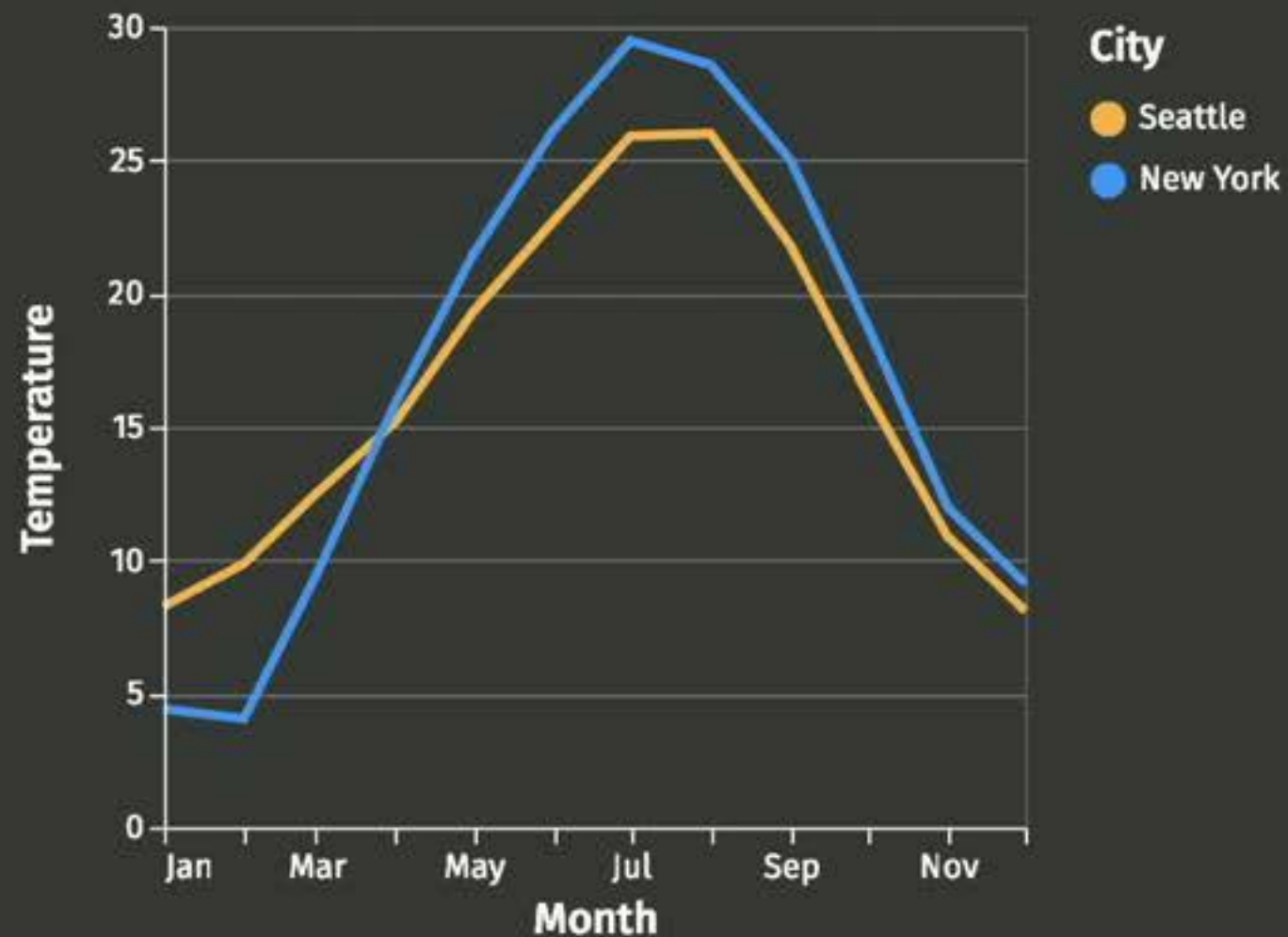
IEEE Infovis 2018. **Best Paper Award**

Formal model of
visual encodings
as sets of facts.

Design knowledge
as constraints.

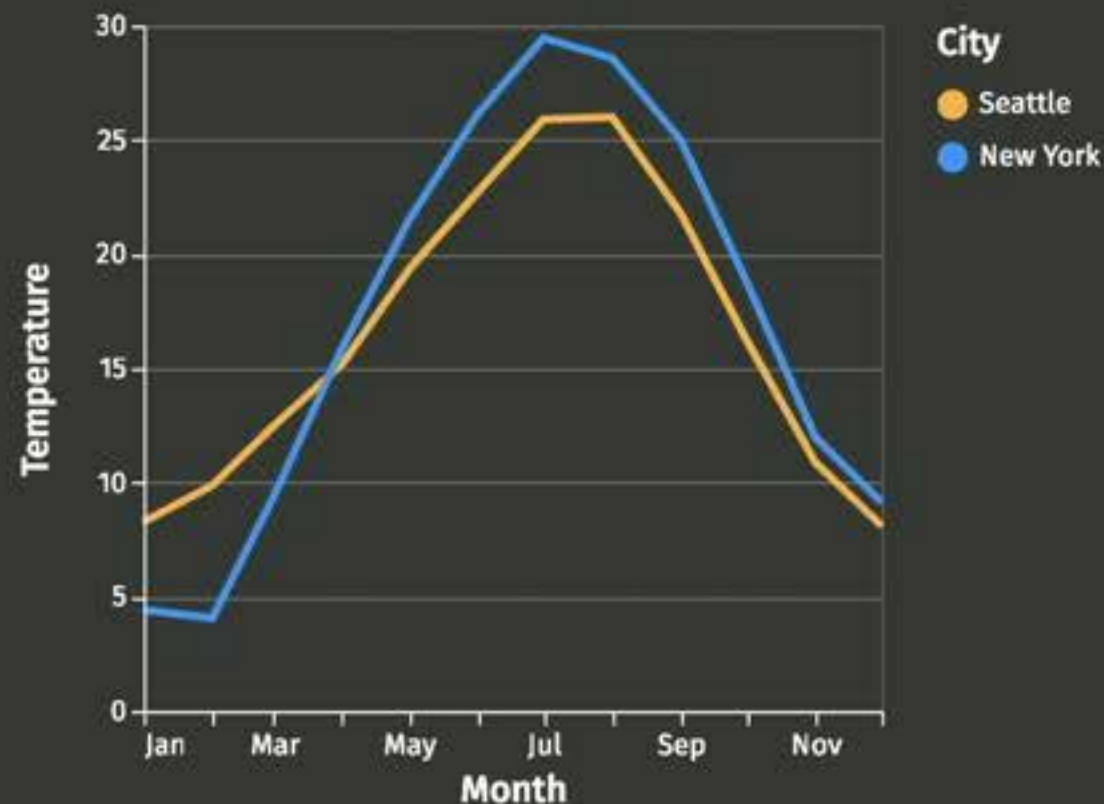


Draco: Encodings as Logical Facts



```
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").

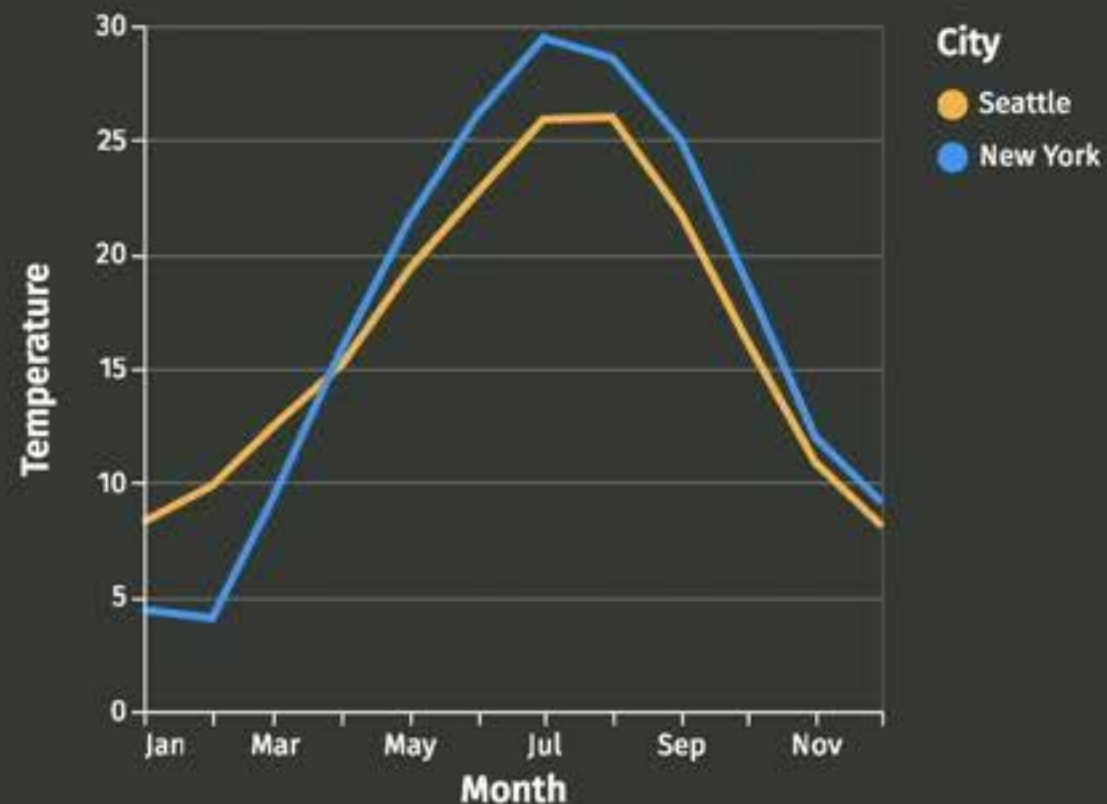
mark(line).

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").
```

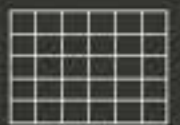
```
mark(line).
```

```
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



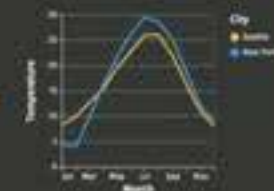
Select Data Fields

City
Date
Temperature

Transform Data

MEAN(Temperature)
BY Month of Date, City

Design Encoding



Data

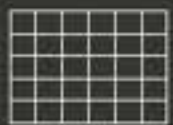
Data Fields

Transformed Data

Visualization

What are the properties of the fields?

Weather



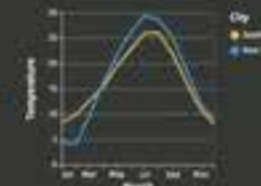
Select Data Fields

City
Date
Temperature

Transform Data

MEAN(Temperature)
BY Month of Date, City

Design Encoding



Data

Data Fields

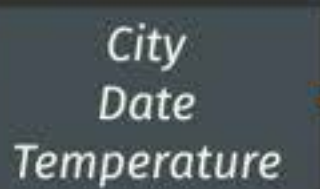
Transformed Data

Visualization

What are the properties of the fields?



Data

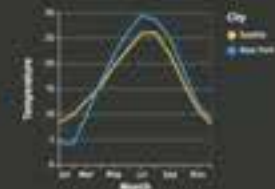


Data Fields



MEAN(Temperature)
BY Month of Date, City

Transformed Data



Visualization



Visualization Always has a Context

What are the properties of the fields?



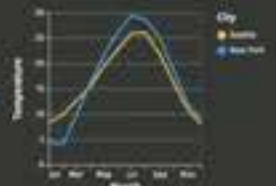
Select Data Fields

City
Date
Temperature

Transform Data

MEAN(Temperature)
BY Month of Date, City

Design Encoding



Data

Data Fields

Transformed Data

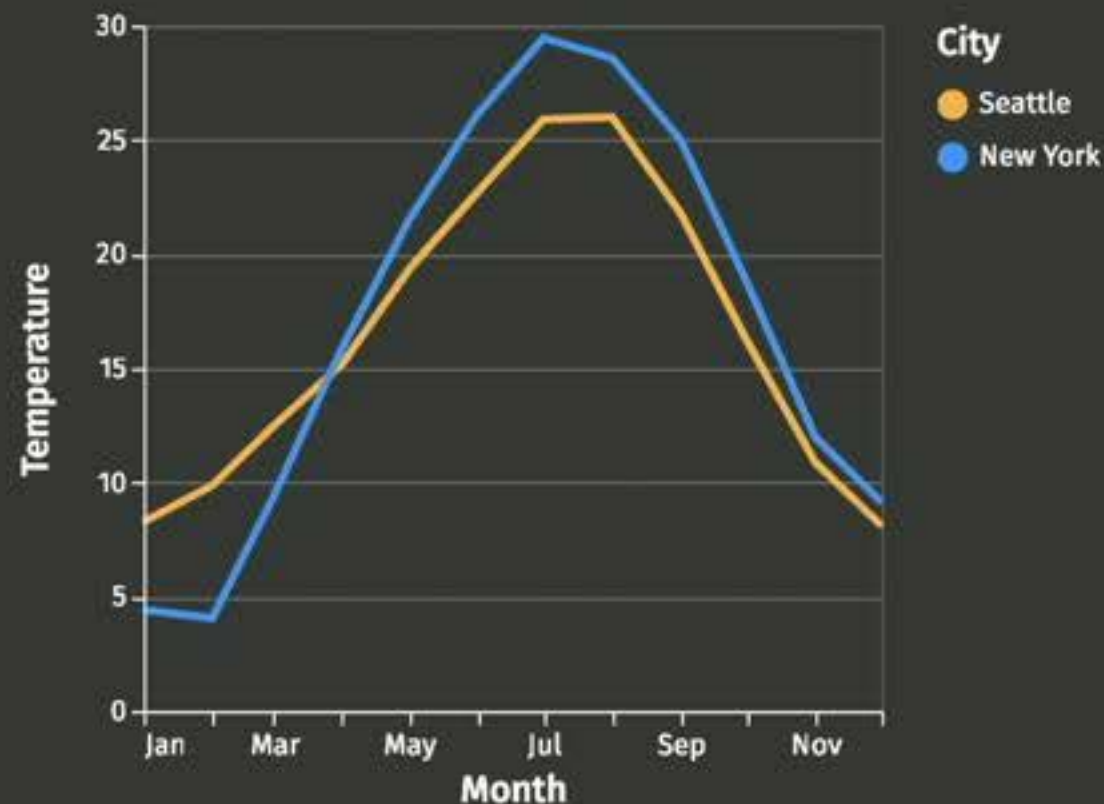
Visualization

Seattle is gray and cold all the time, right?



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



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.

“Prefer specifications with fewer encodings.”

“Prefer not to use aggregation.”

“Prevent overlapping marks.”

total ~150 soft constraints

Attribute domains

Integrity

Preferences

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).
```



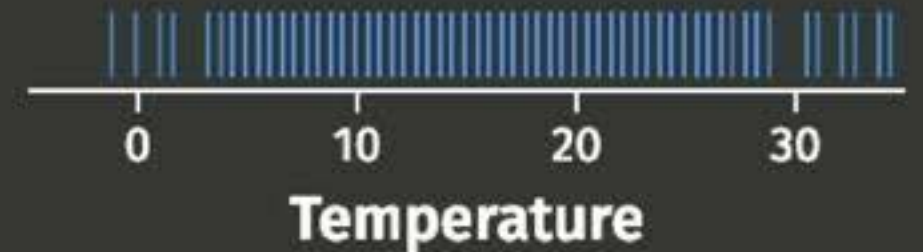
Attribute domains

Integrity

Preferences

constraint
solver

```
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).
```



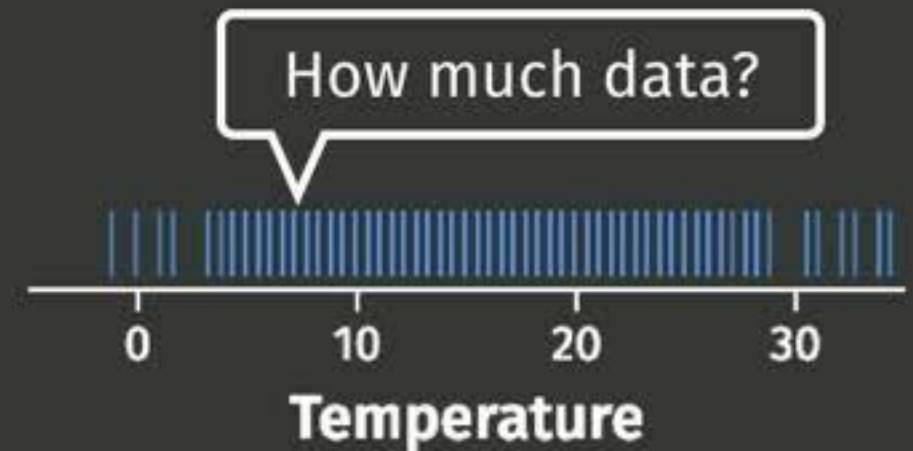
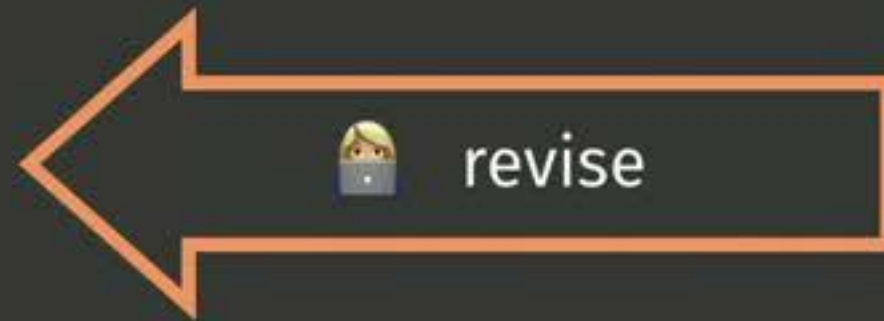
Attribute domains

Integrity

Preferences

constraint
solver

```
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 **bin**ned temperature."



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

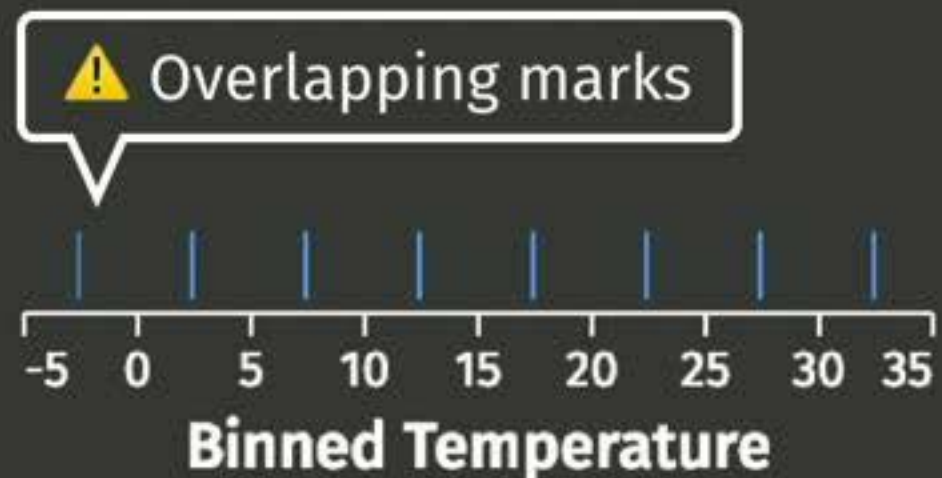


Attribute domains

Integrity.

Preferences

constraint
solver



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

Visualization Recommendation with Draco

"I want a visualization of the **bin**ned temperature."



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



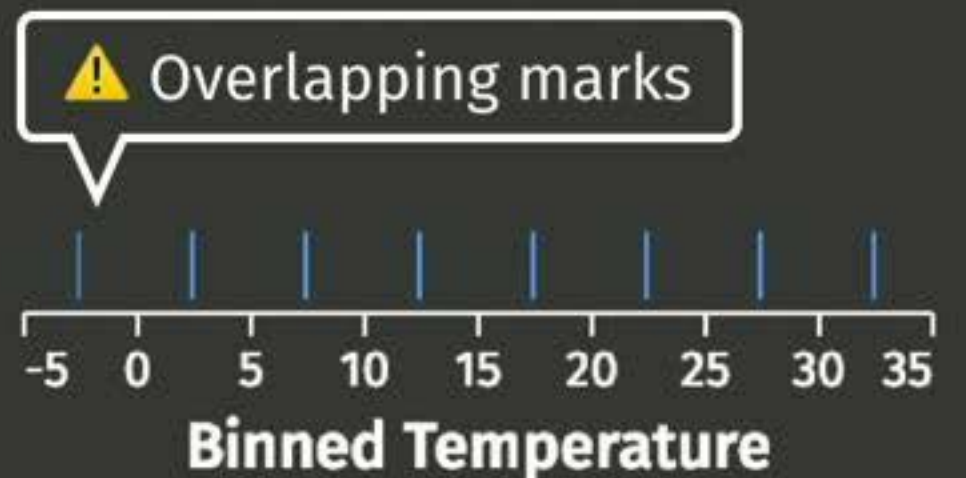
Attribute domains

Integrity.

Preferences

constraint
solver

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



```
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).
```



Attribute domains

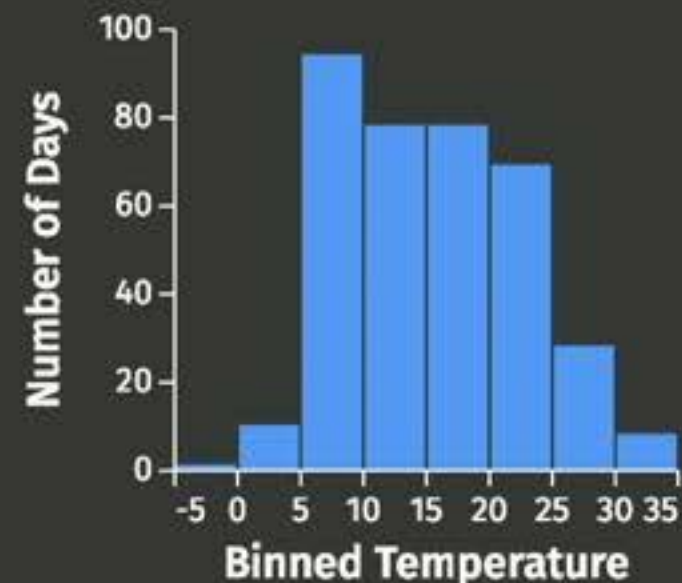
Integrity.

Preferences

constraint
solver

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

$$2+3 < 6$$



```
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.

Design knowledge
as constraints.

Attribute domains

Integrity

Preferences

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

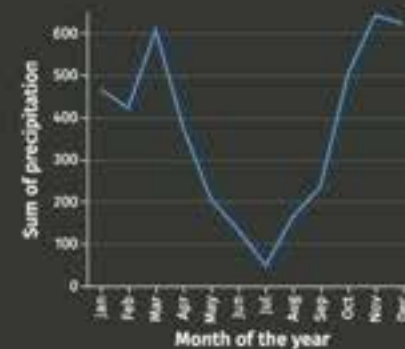
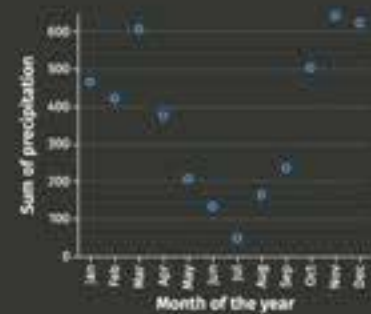
(**Visualization** → **Score**) → Pairs where score is significantly different



Score: 0.5 (bad)



Score: 1.7 (good)



Score: 1.1 (okay)



Score: 2.5 (great)



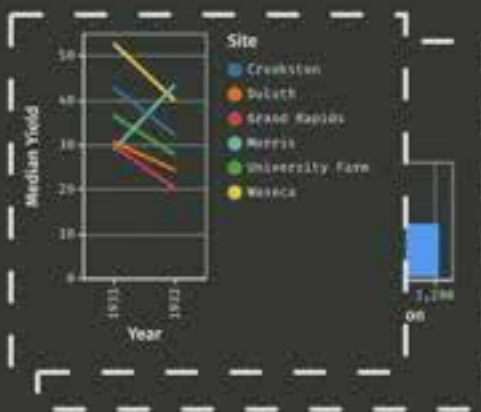
Pairs are Ordinal

We can combine the results of multiple experiments with different measures.

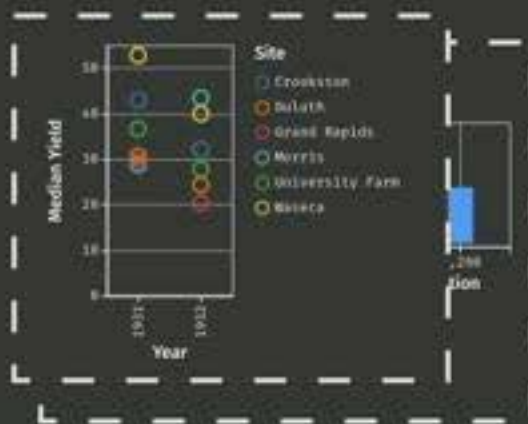
Draco: Learn Trade-Offs from Data

Training Data

Pairs of
Ranked Visualizations



V



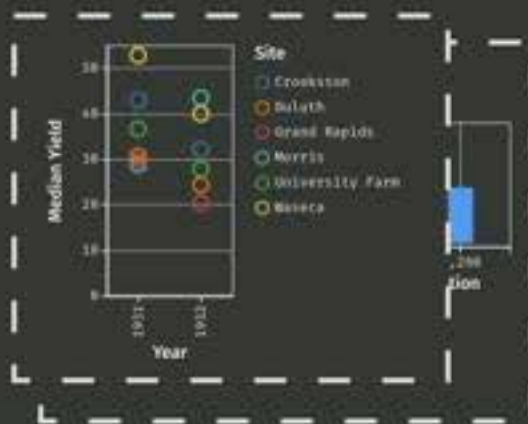
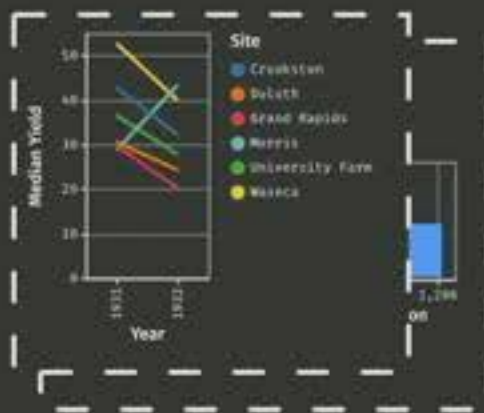
Draco: Learn Trade-Offs from Data

Training Data

Pairs of
Ranked Visualizations

Features

Violations of
Soft Constraints



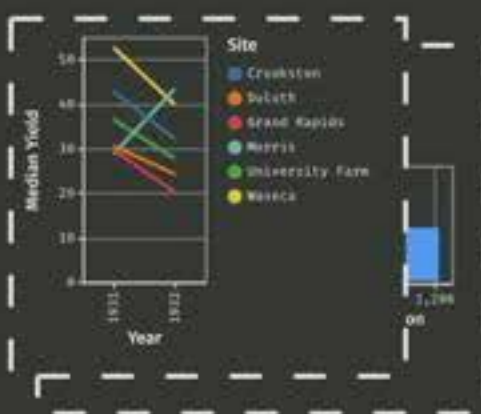
Draco: Learn Trade-Offs from Data

Training Data

Pairs of
Ranked Visualizations

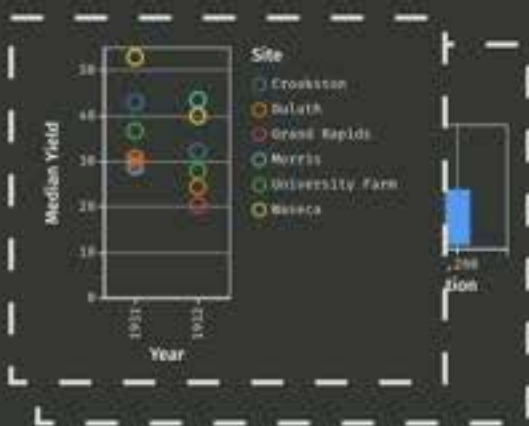
Features

Violations of
Soft Constraints



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

V



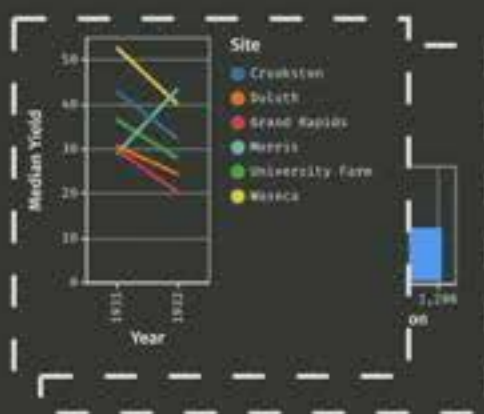
👎 negative example
Feature Vector
 $[v_1, v_2, \dots, v_k]$

v_i : the number of
violations of constraint i

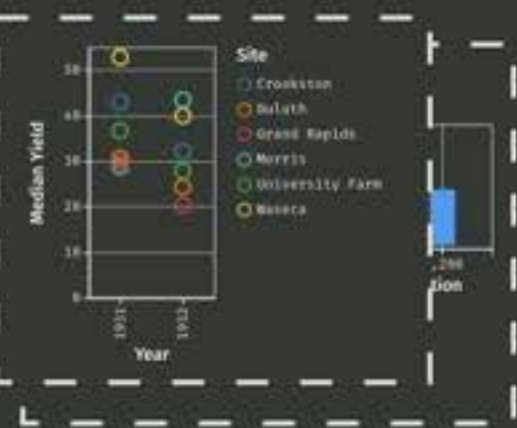
Draco: Learn Trade-Offs from Data

Training Data

Pairs of
Ranked Visualizations



V



Features

Violations of
Soft Constraints

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

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

v_i : the number of
violations of constraint i

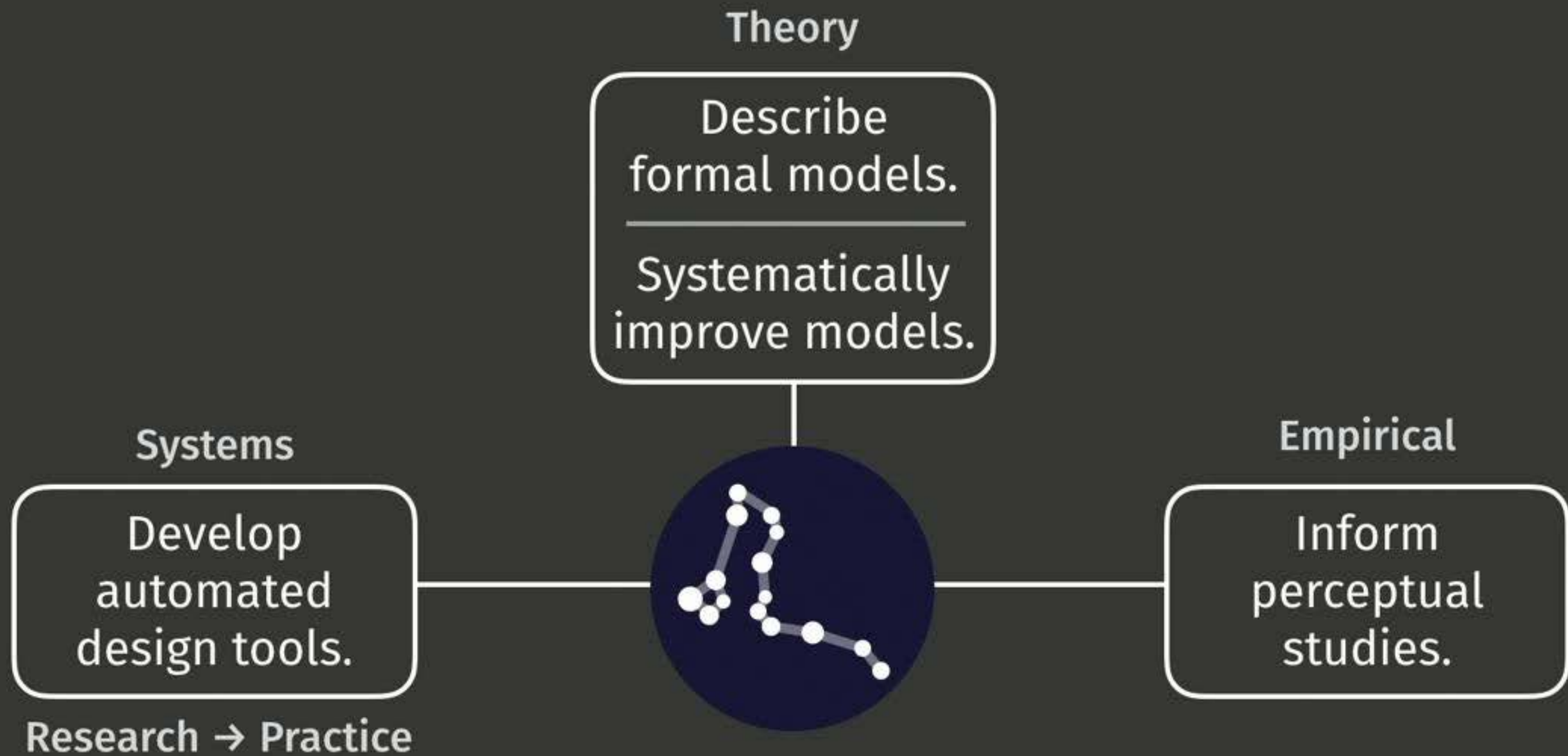
Learning Algorithm

Learning to Rank
with Linear SVM

w is the weight vector
of the soft constraints

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

Draco at the Core of Visualization Research



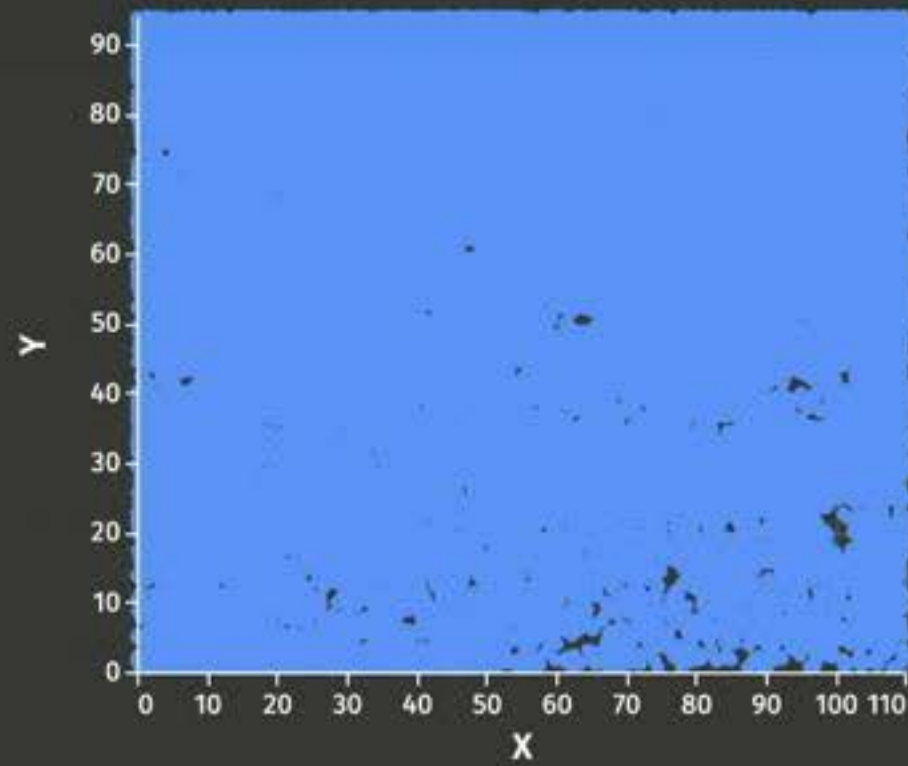
Infovis 2018. **Best Paper**

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

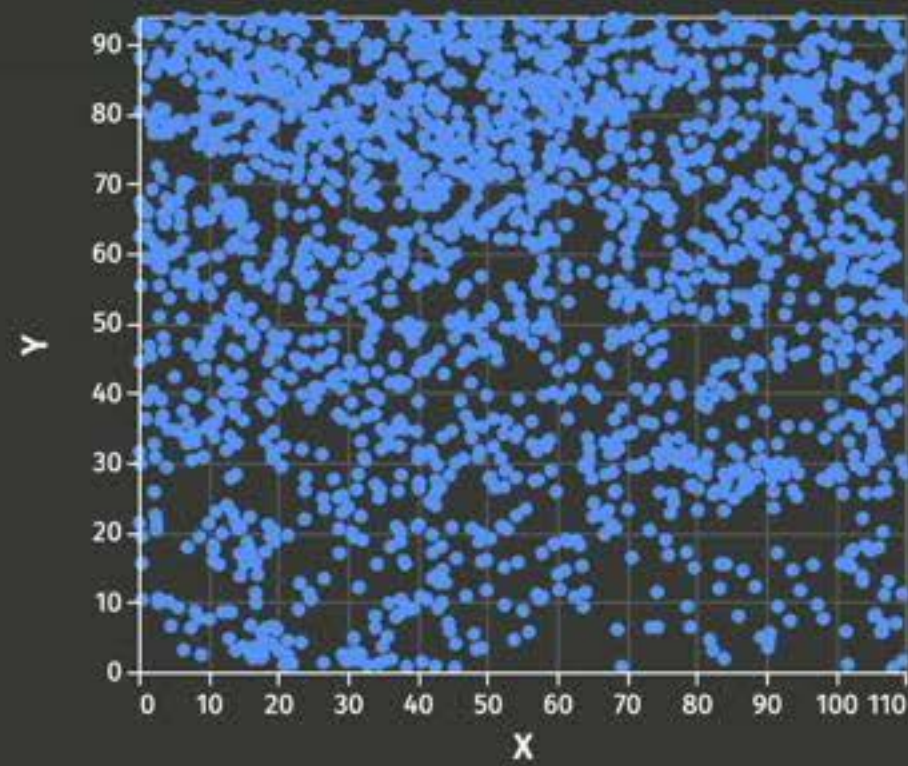
How can we visualize and
interact with **billion+ record**
datasets in real-time?

How can we visualize and
interact with **billion+ record**
datasets in real-time?

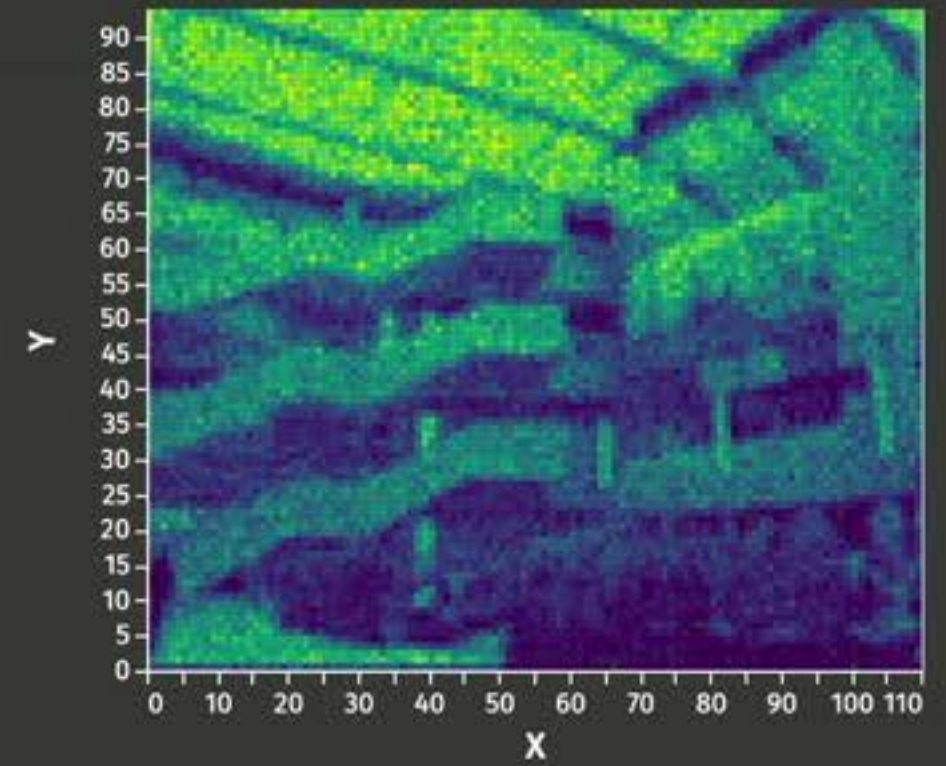
How to Visualize a Billion+ Records



Data

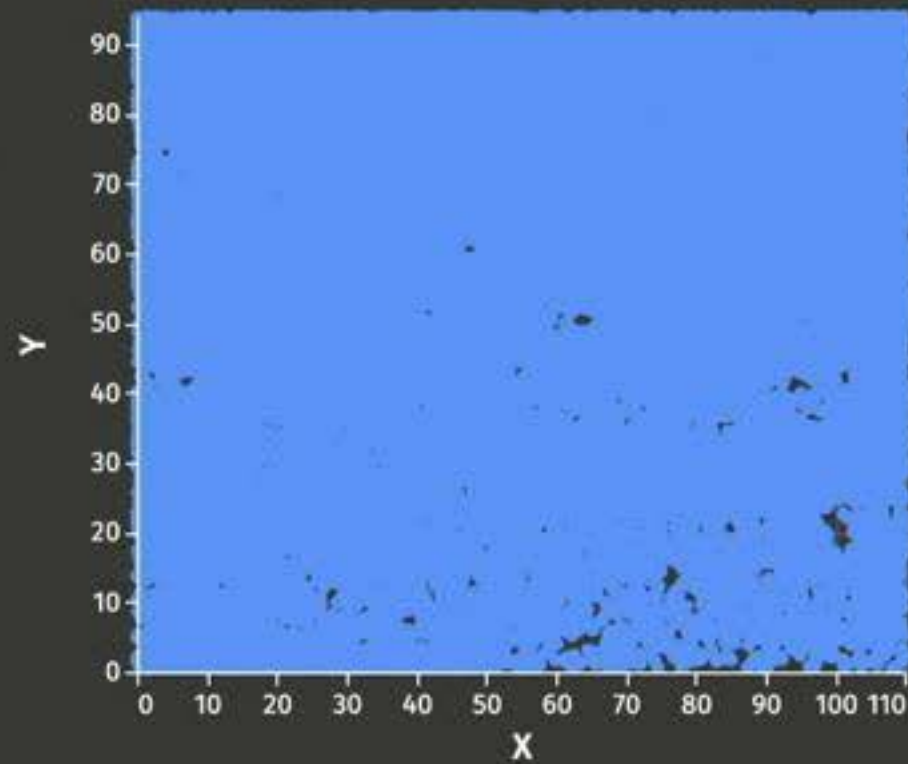


Sampling

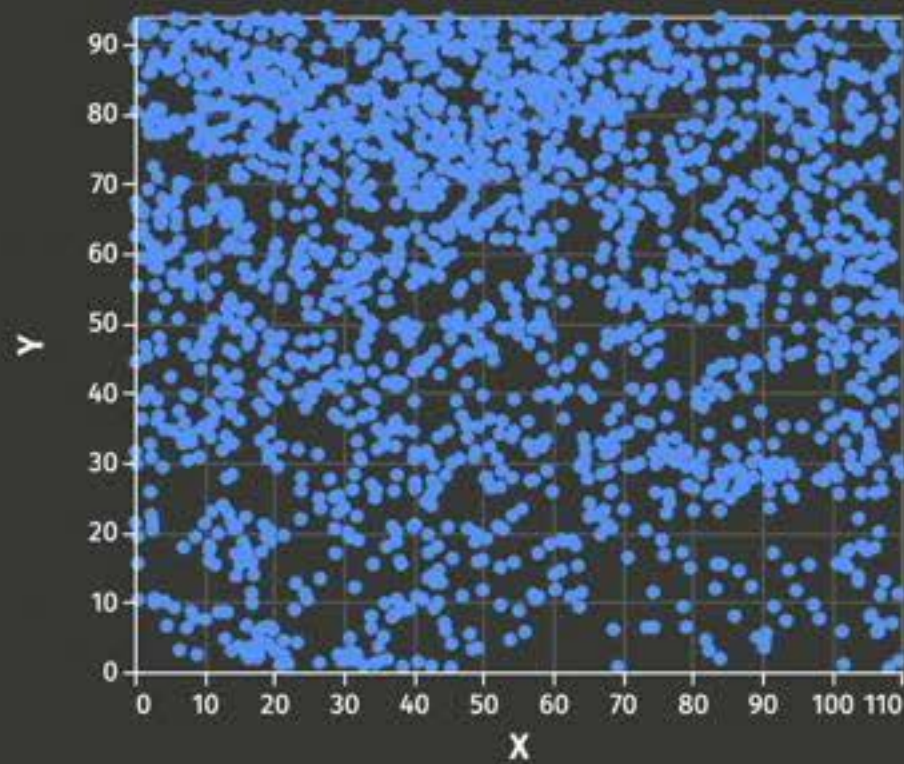


Binned Aggregation

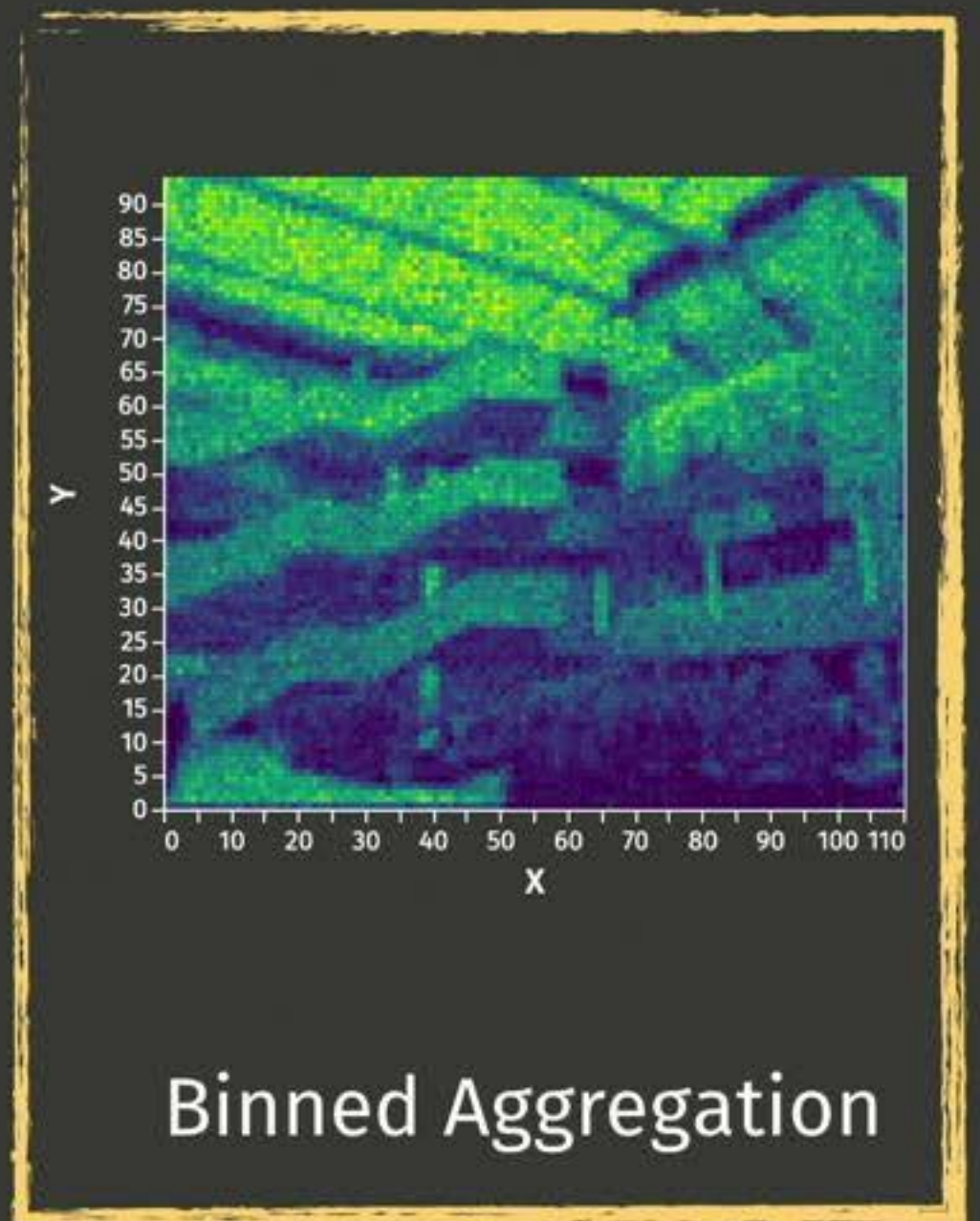
How to Visualize a Billion+ Records



Data



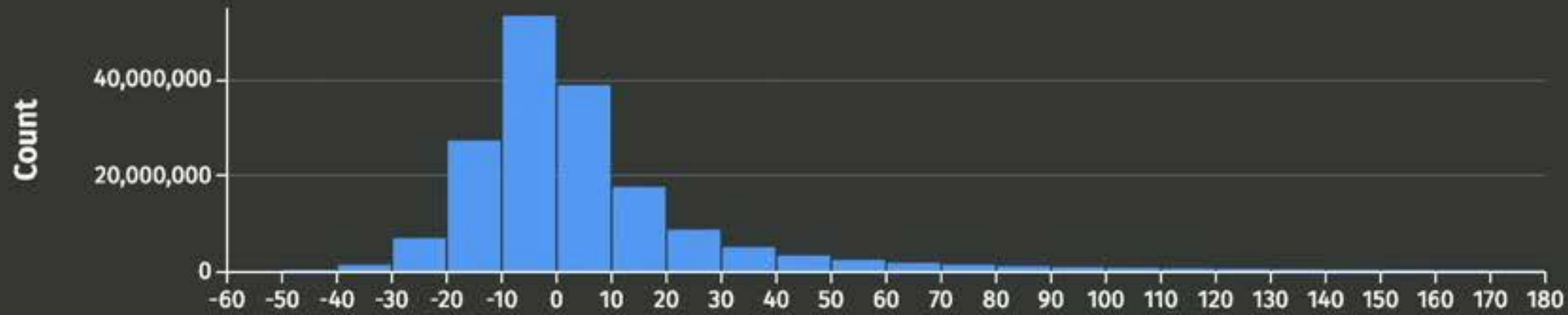
Sampling



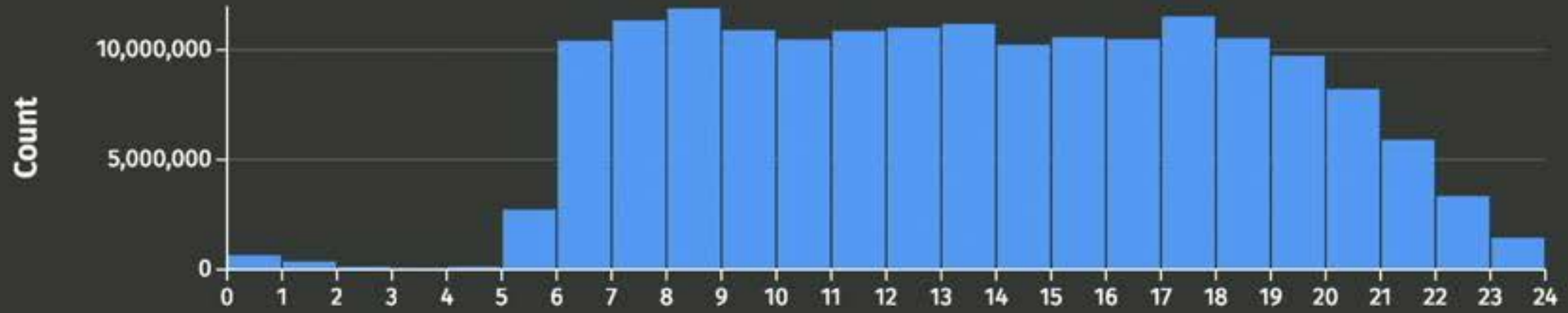
Binned Aggregation

Decouple the visual complexity from the raw data through aggregation.

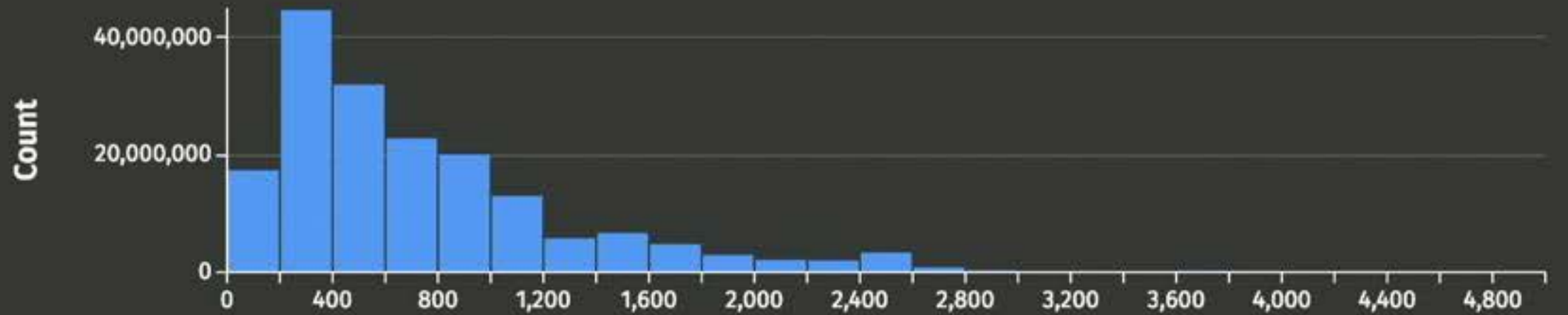
Arrival Delay in Minutes



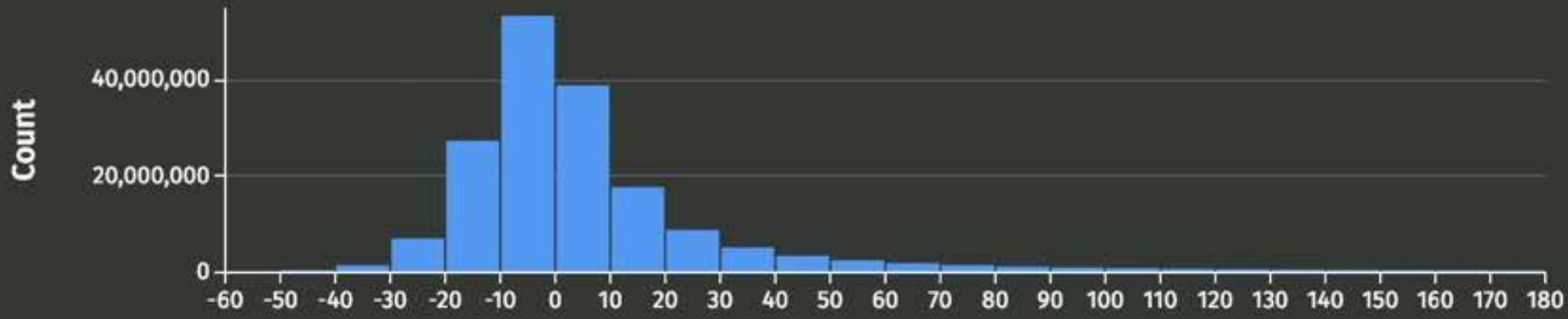
Departure Time



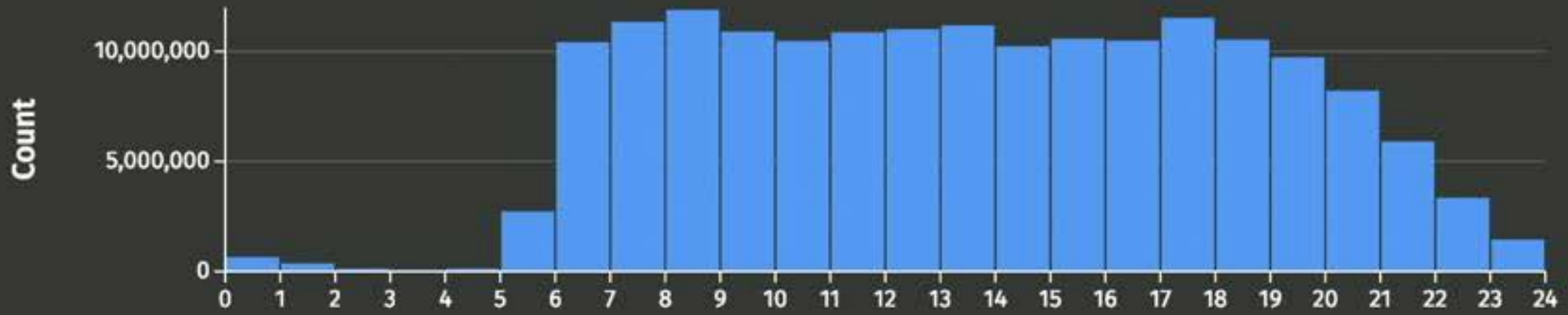
Distance in Miles



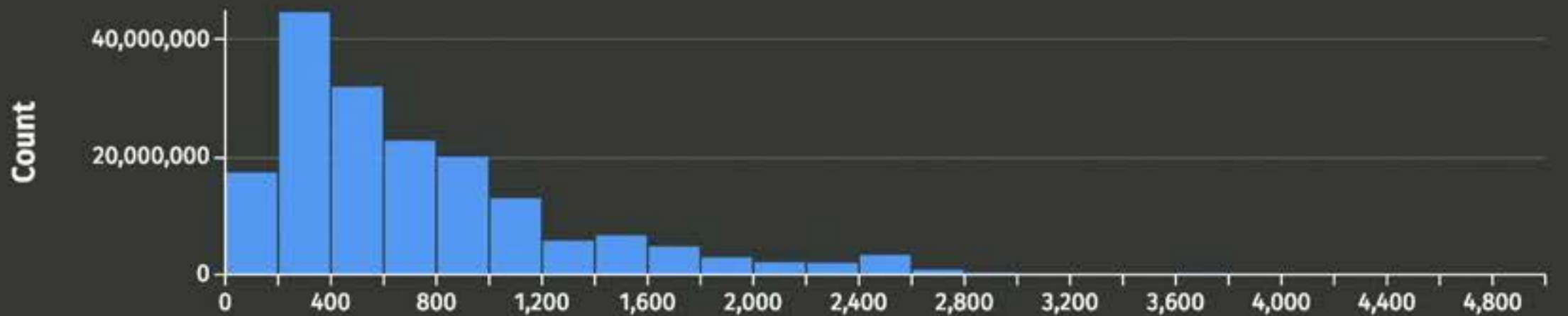
Arrival Delay in Minutes



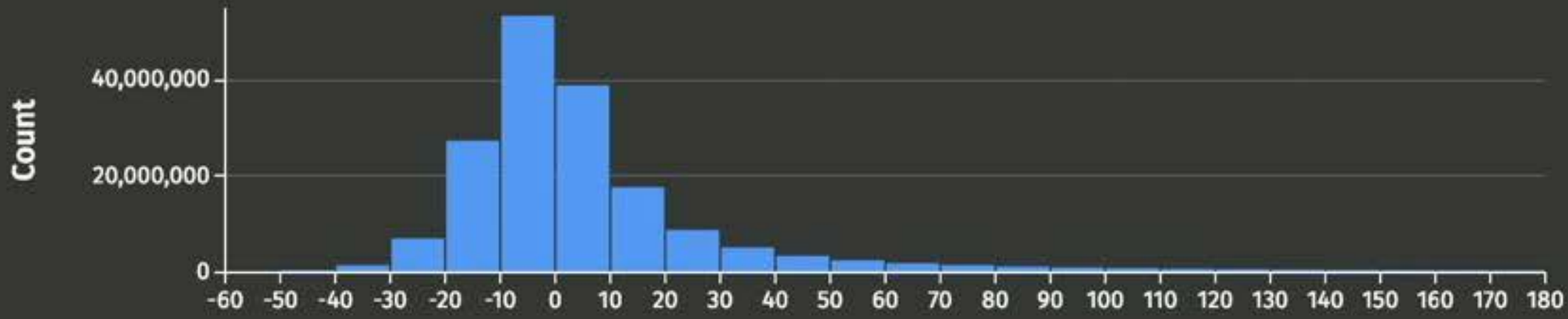
Departure Time



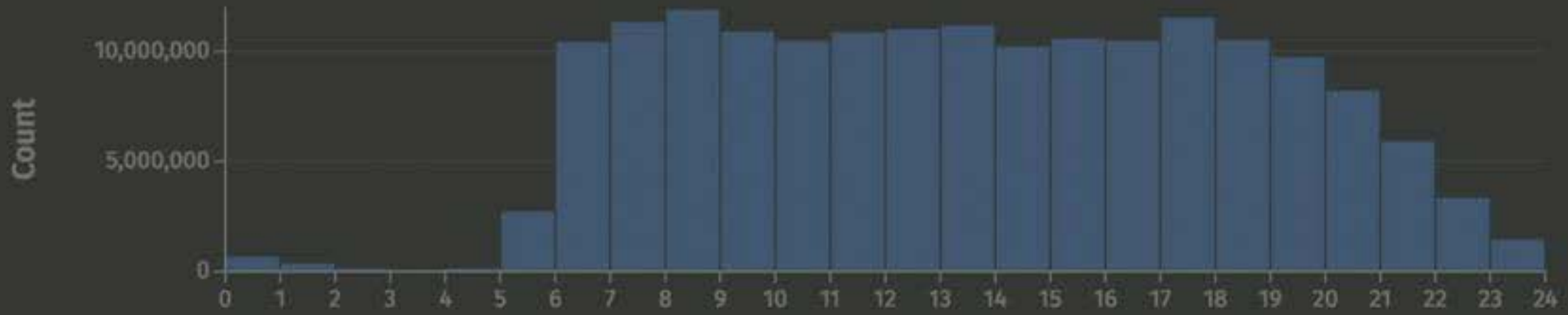
Distance in Miles



Arrival Delay in Minutes



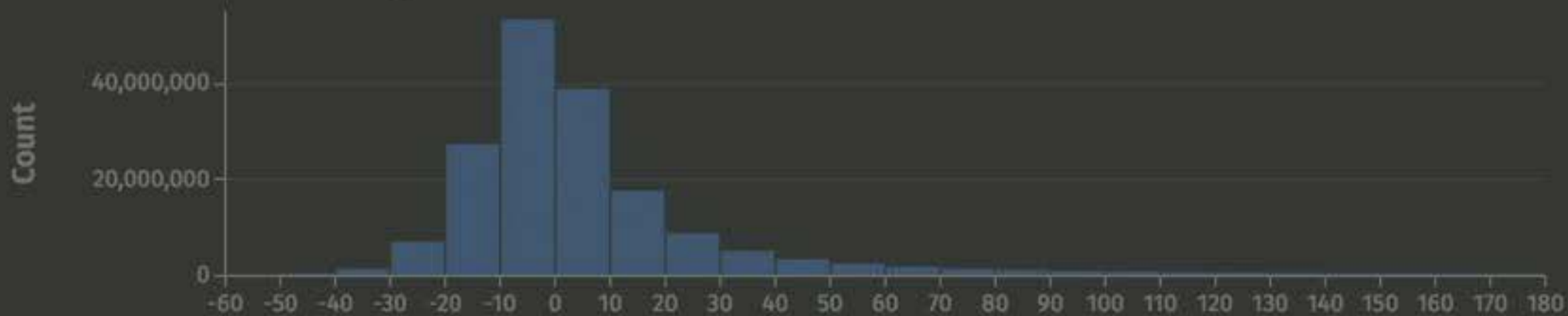
Departure Time



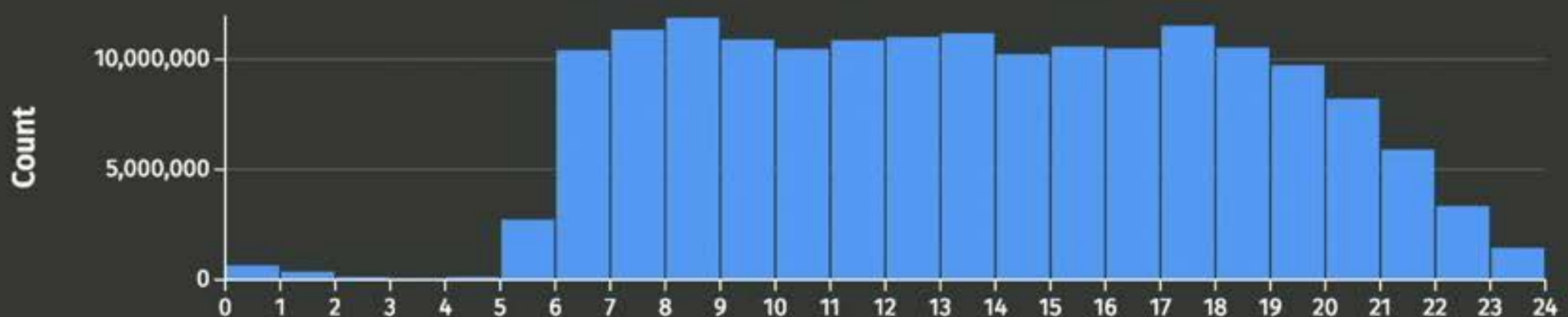
Distance In Miles



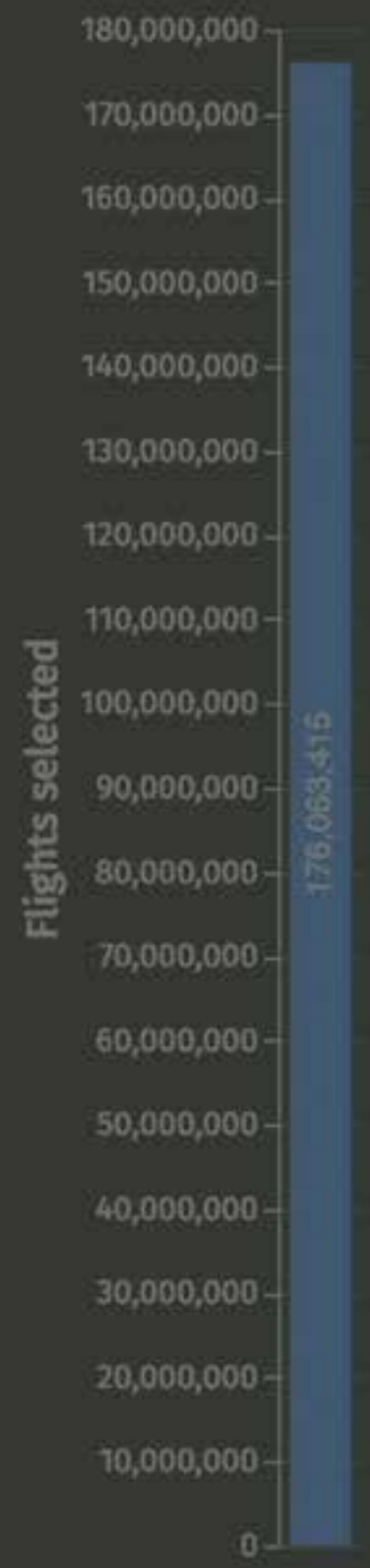
Arrival Delay in Minutes



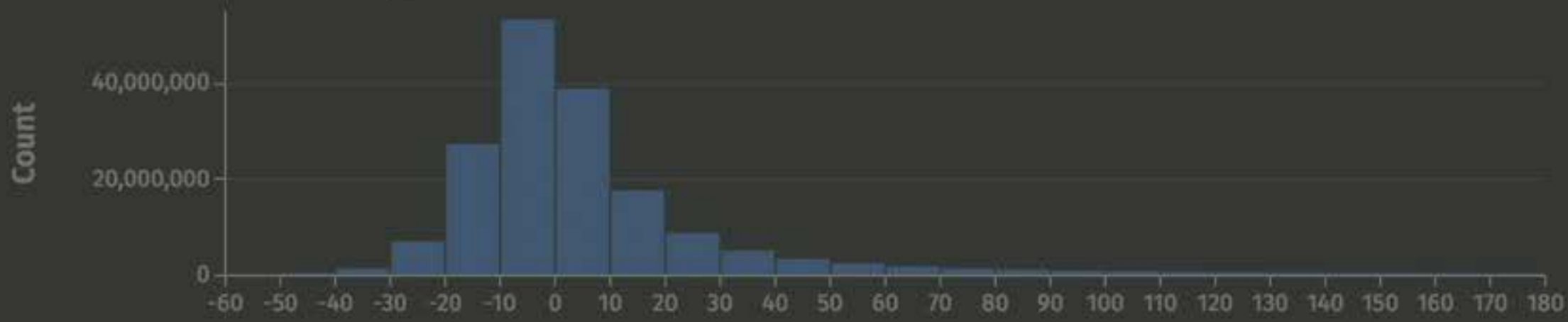
Departure Time



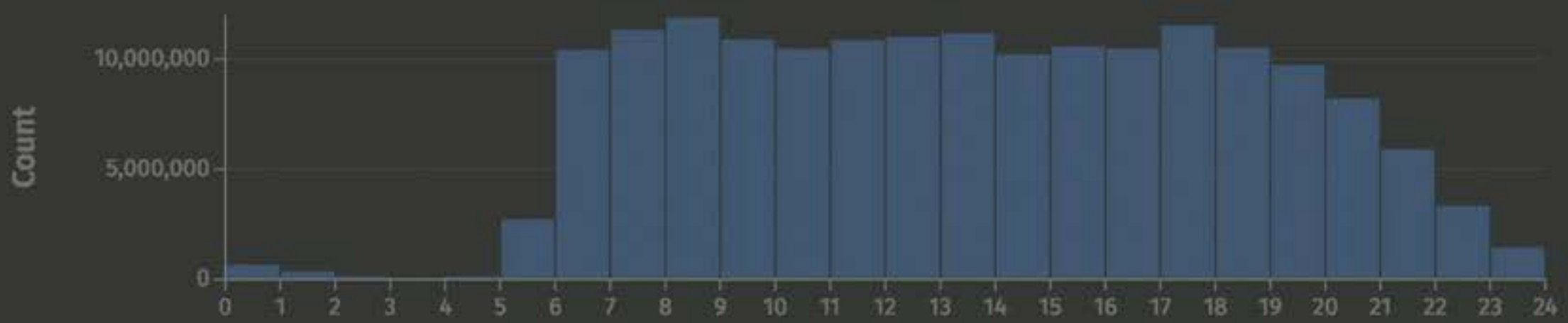
Distance In Miles



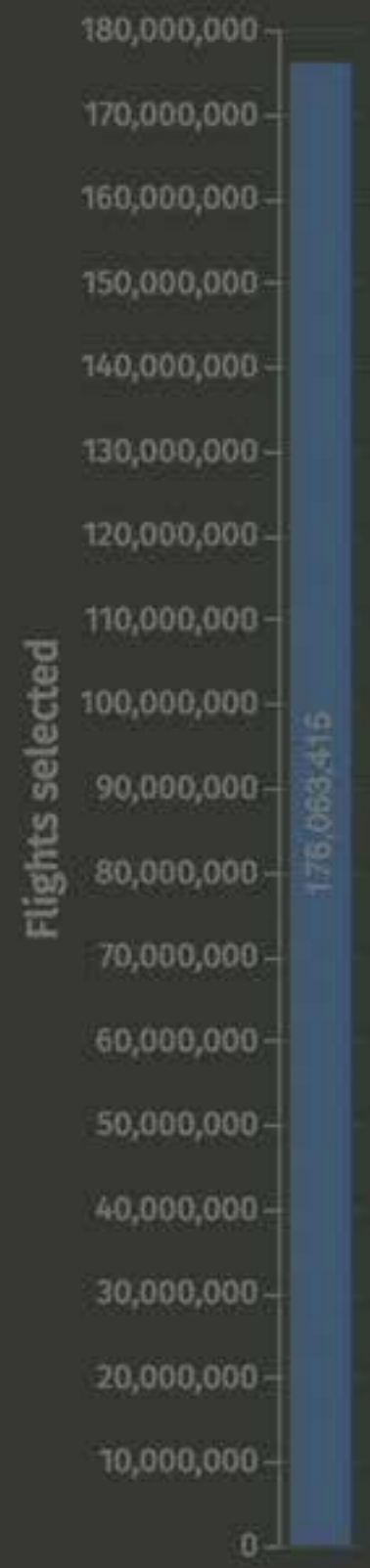
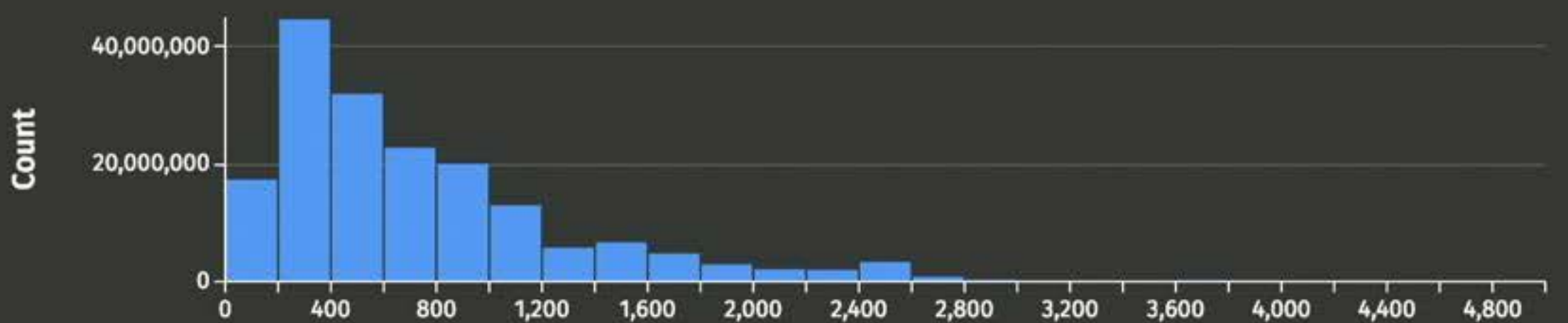
Arrival Delay in Minutes



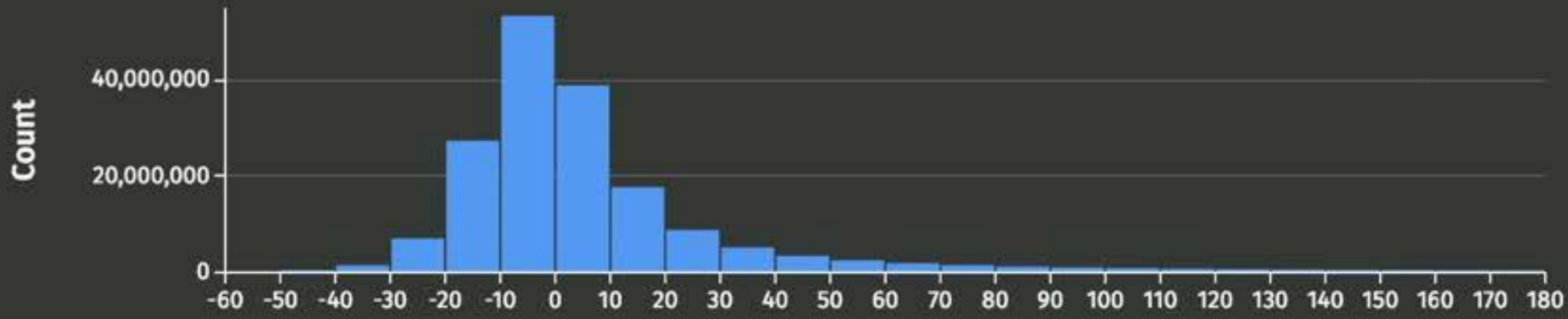
Departure Time



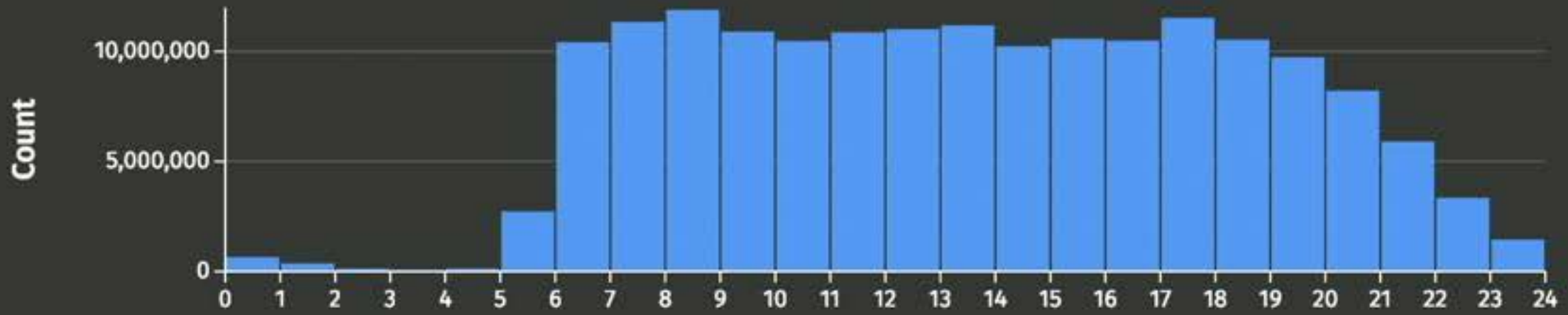
Distance in Miles



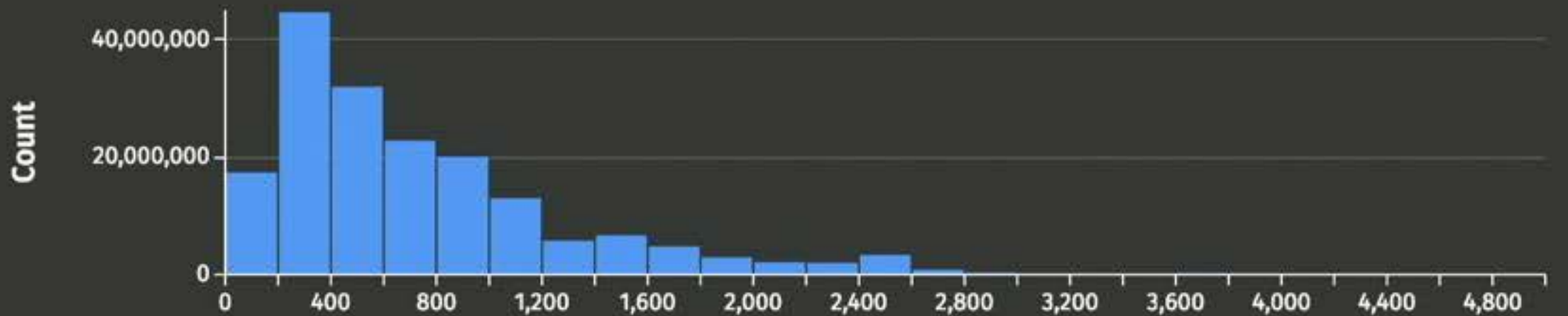
Arrival Delay in Minutes



Departure Time



Distance in Miles



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

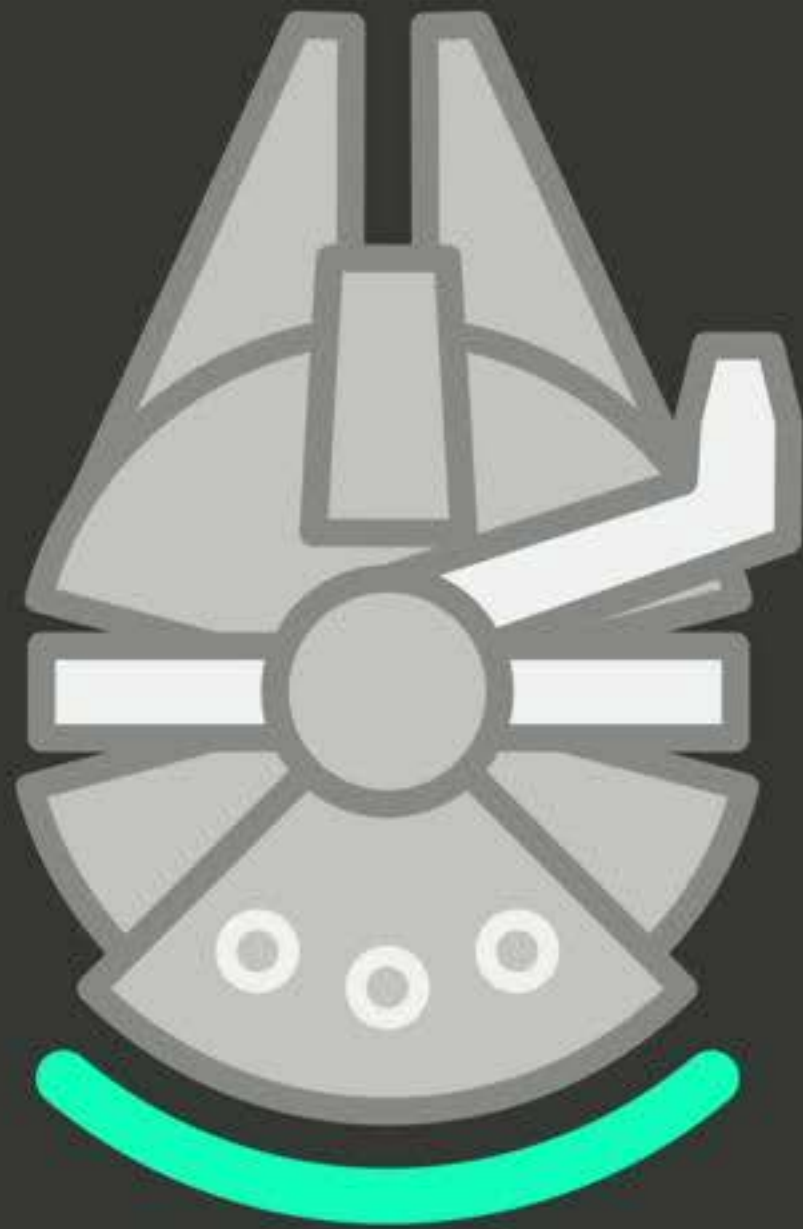
How do we interact with
billion+record datasets in real-time?

How do we interact with billion+record datasets in real-time?

Delays reduce engagement and lead to fewer observations.

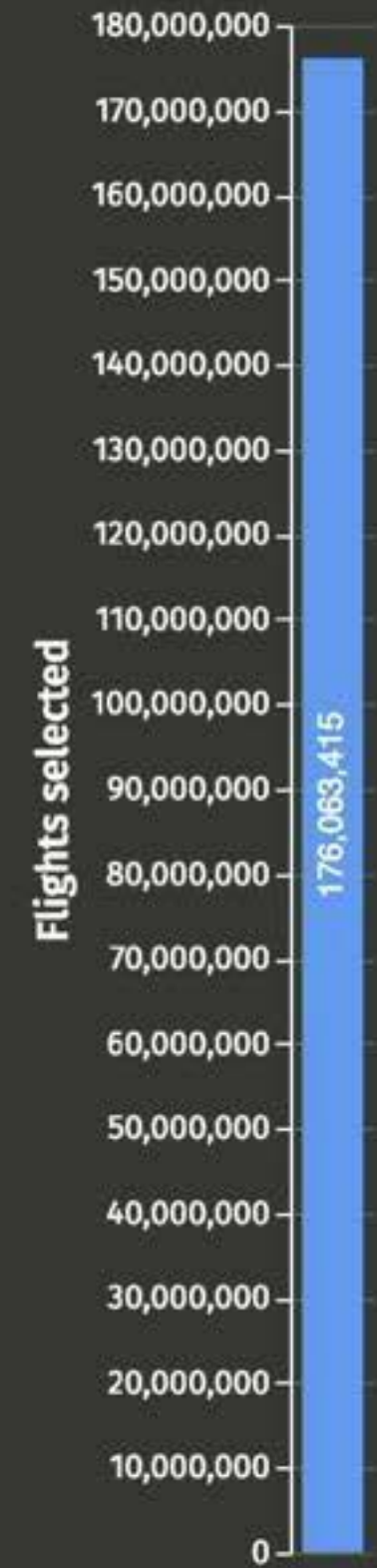
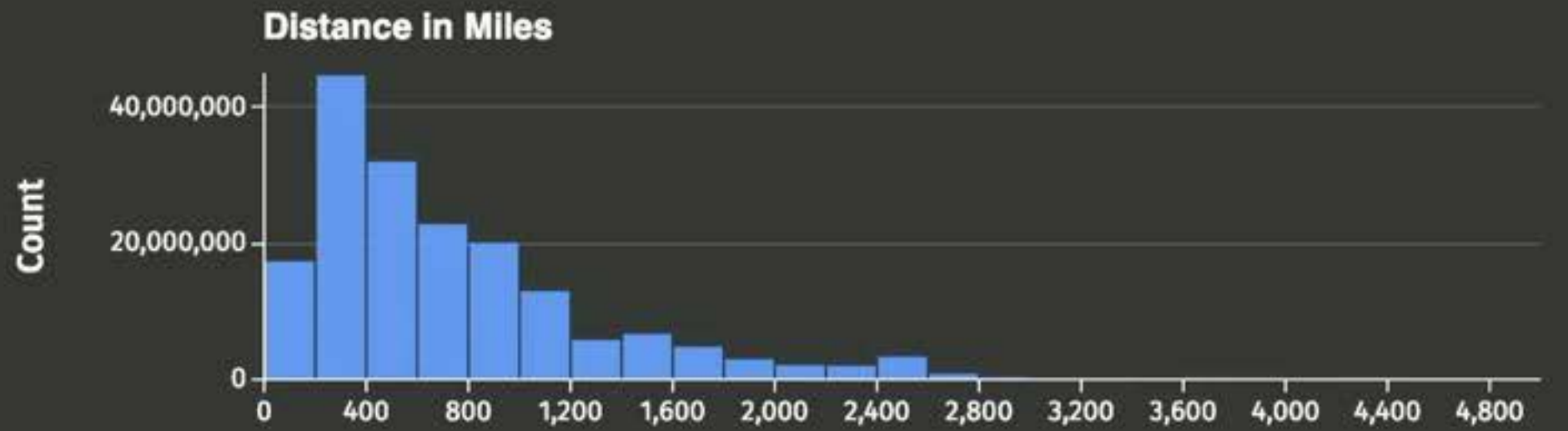
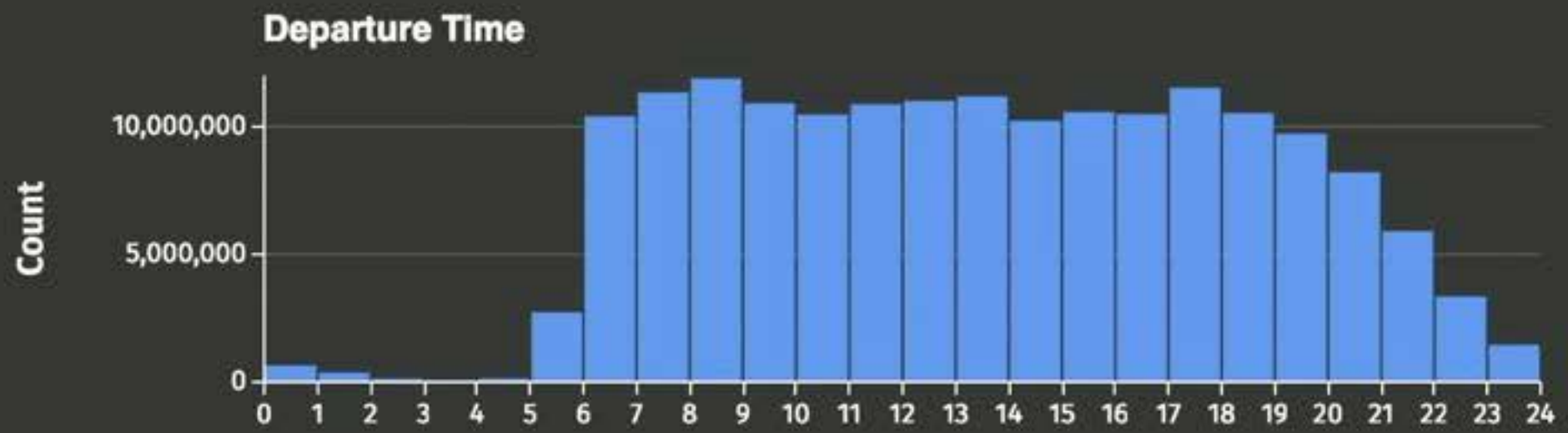
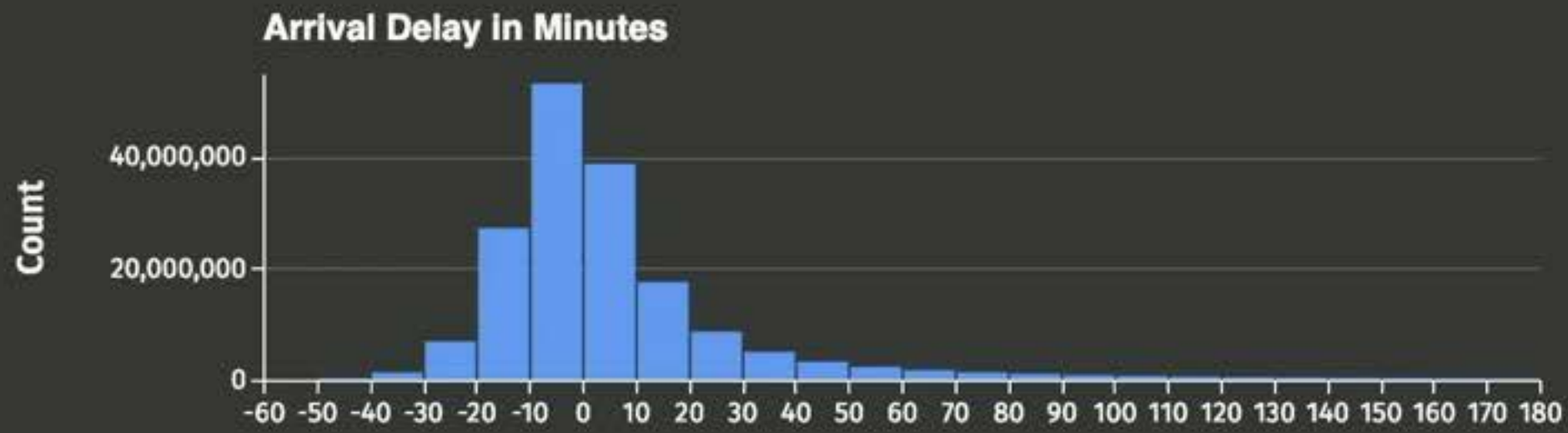
The Effect of Interactive Latency. Liu, Heer. IEEE Infovis 2014.

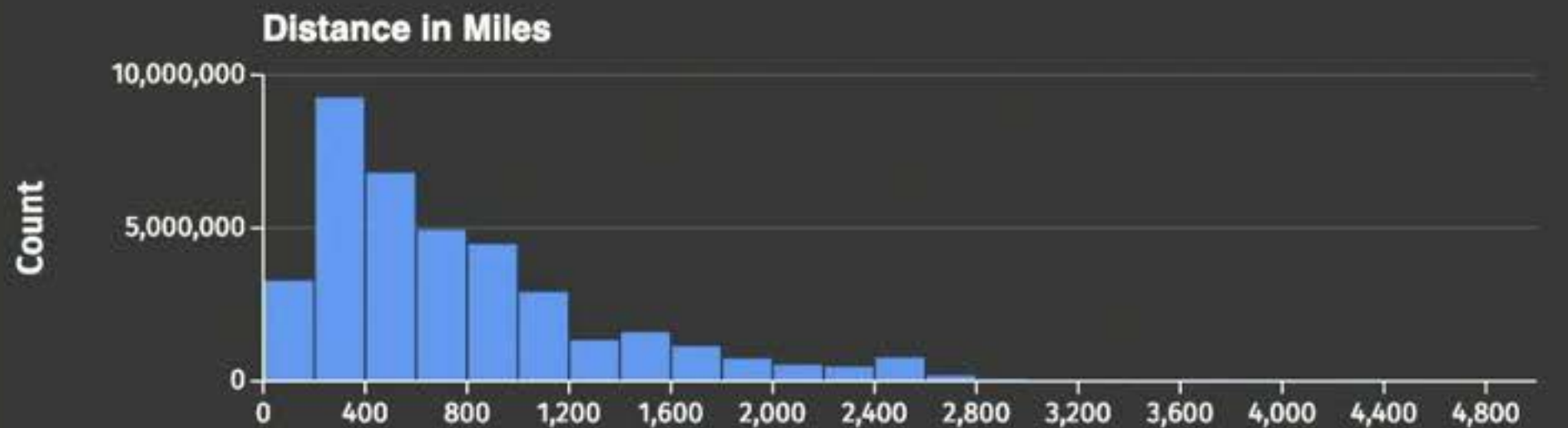
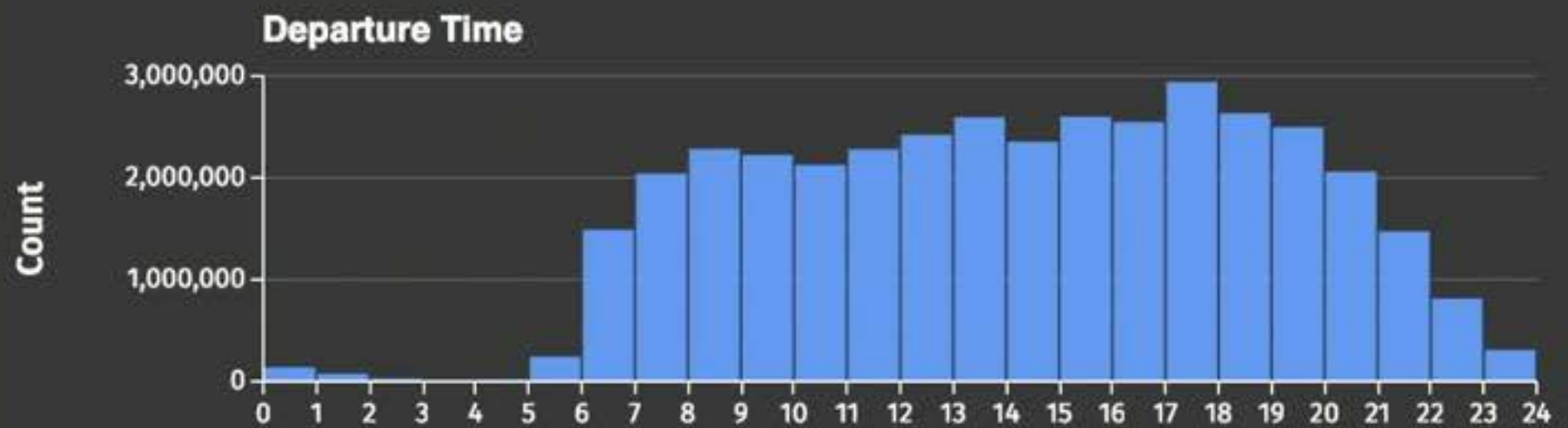
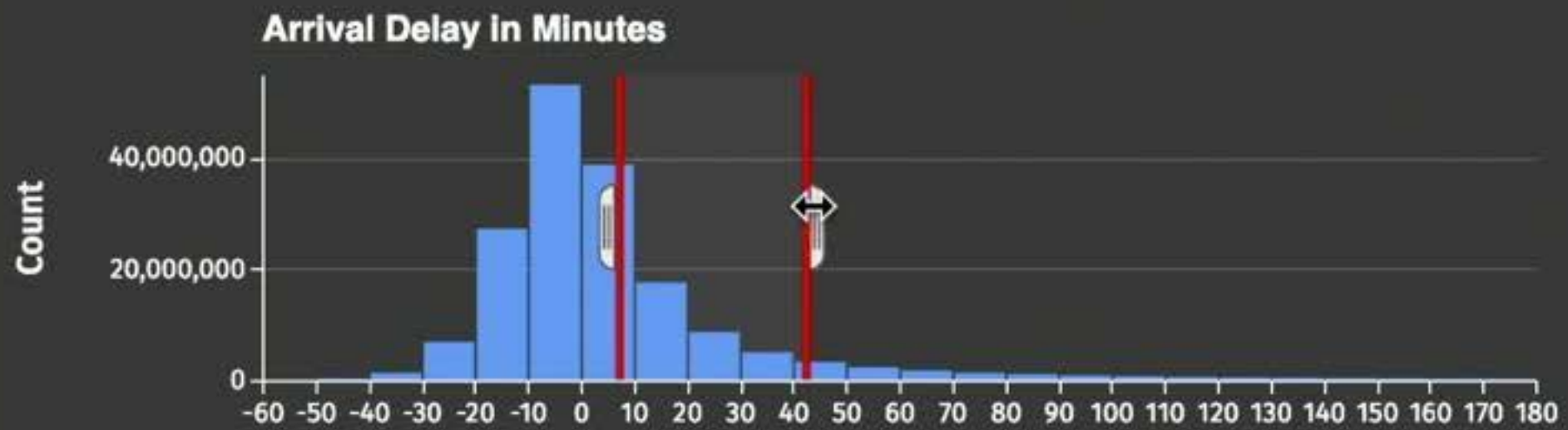
Delays may bias analysts towards convenient data.

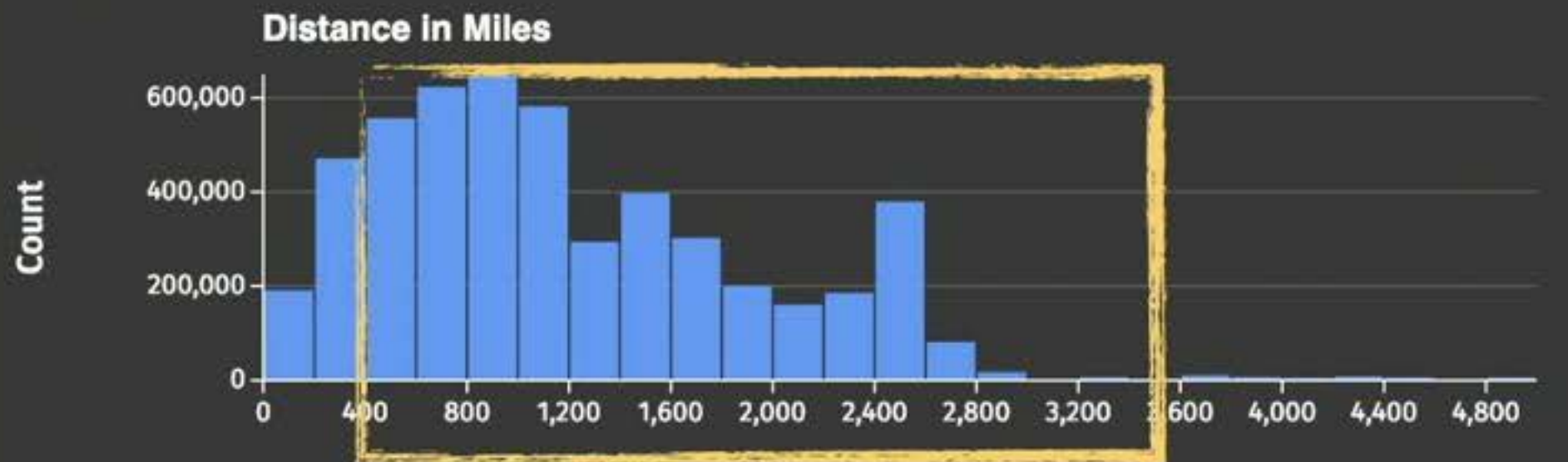
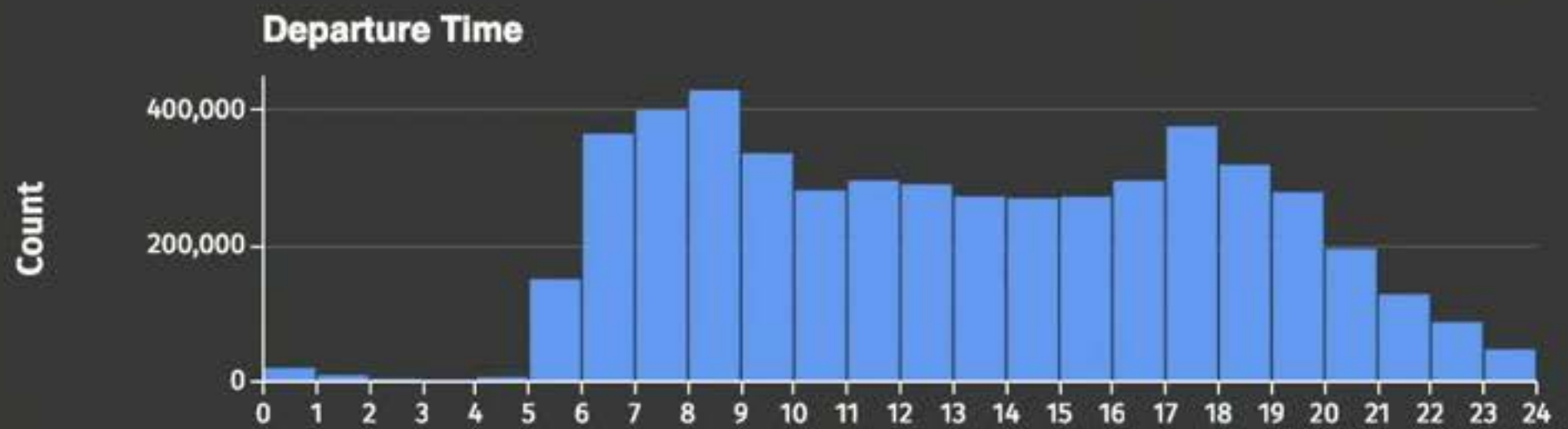
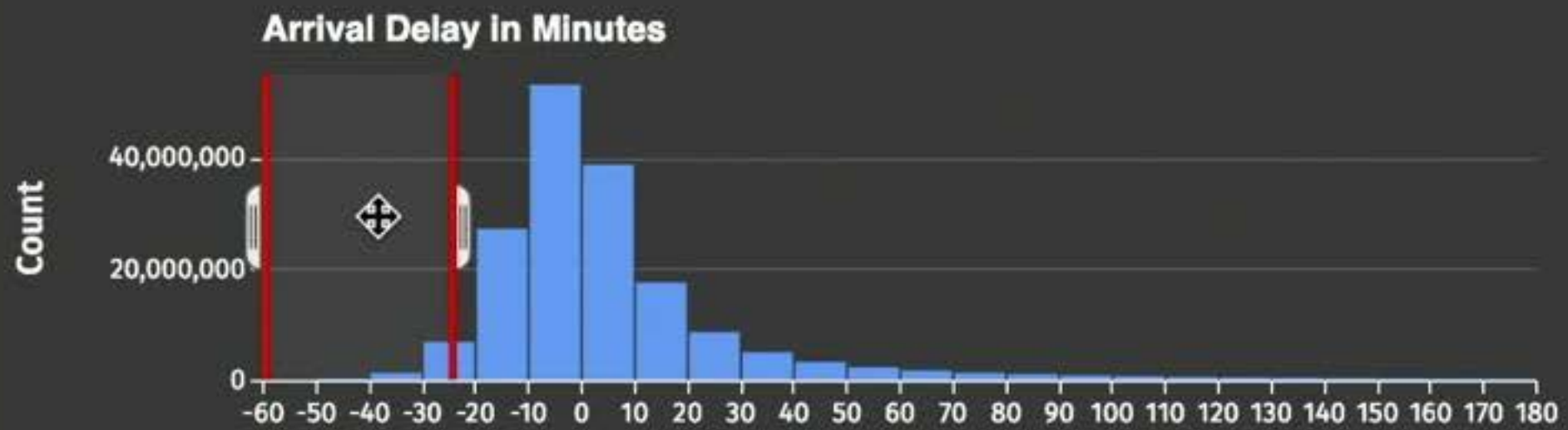


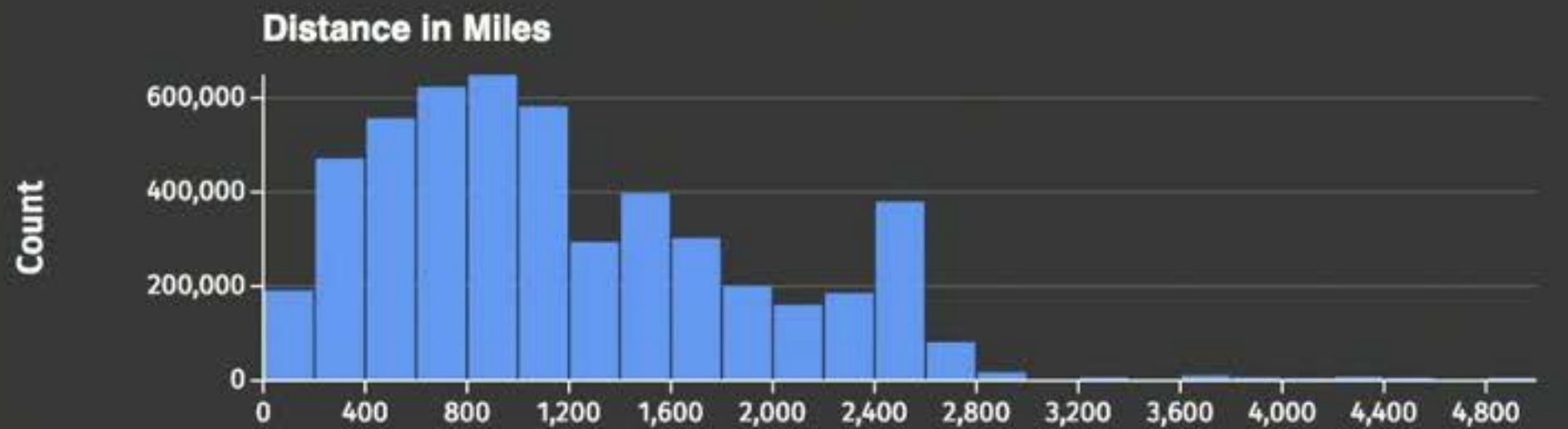
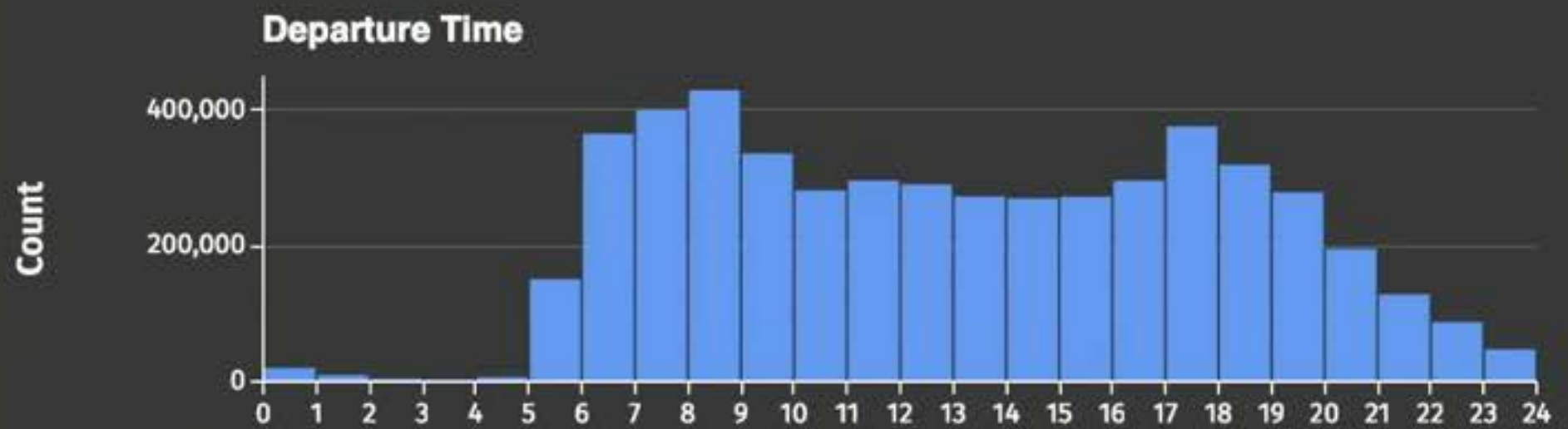
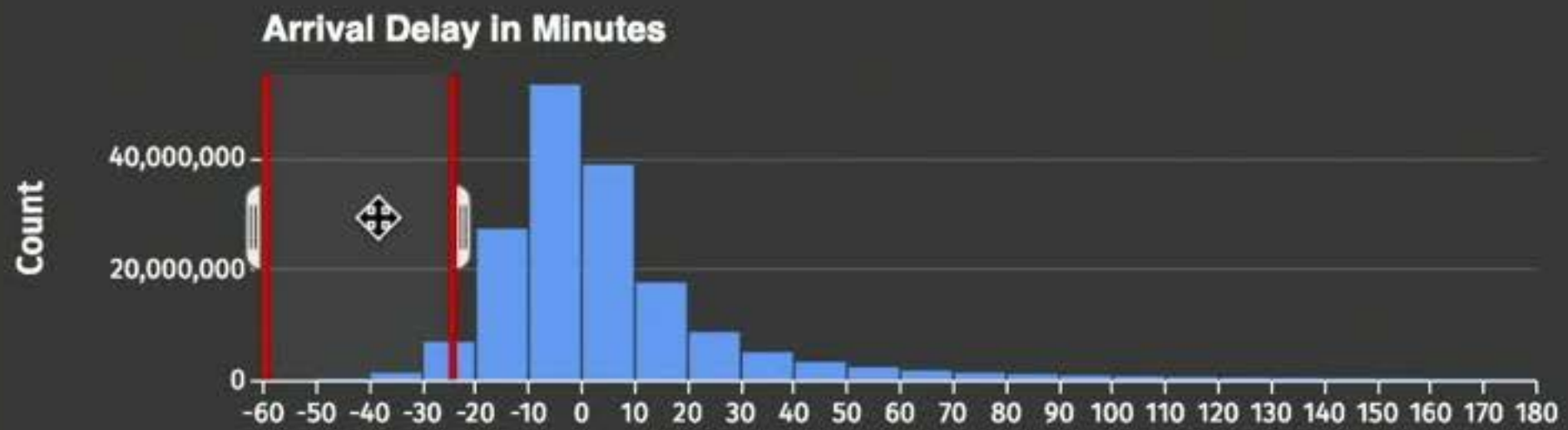
Falcon

uwdata.github.io/falcon

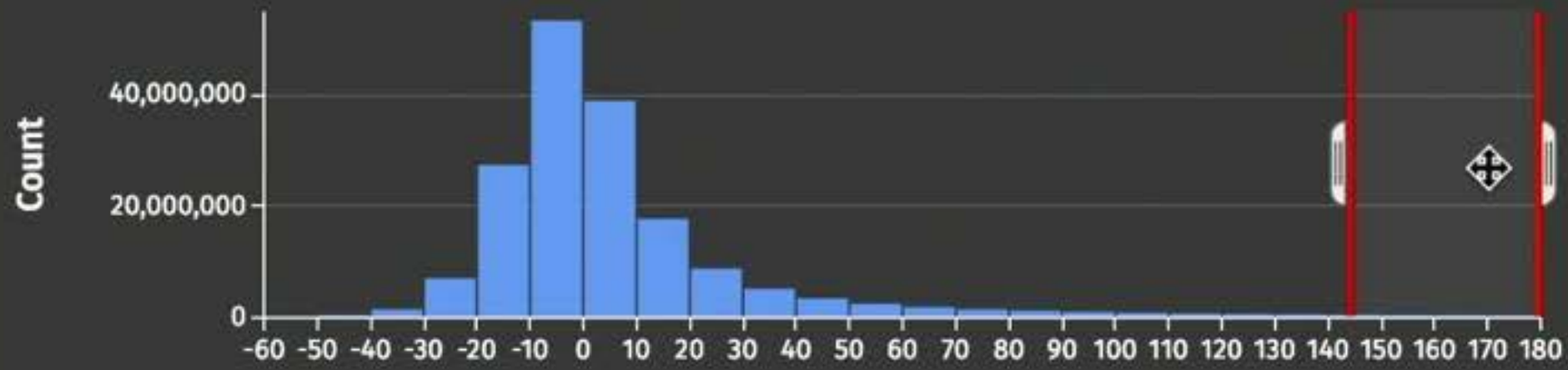




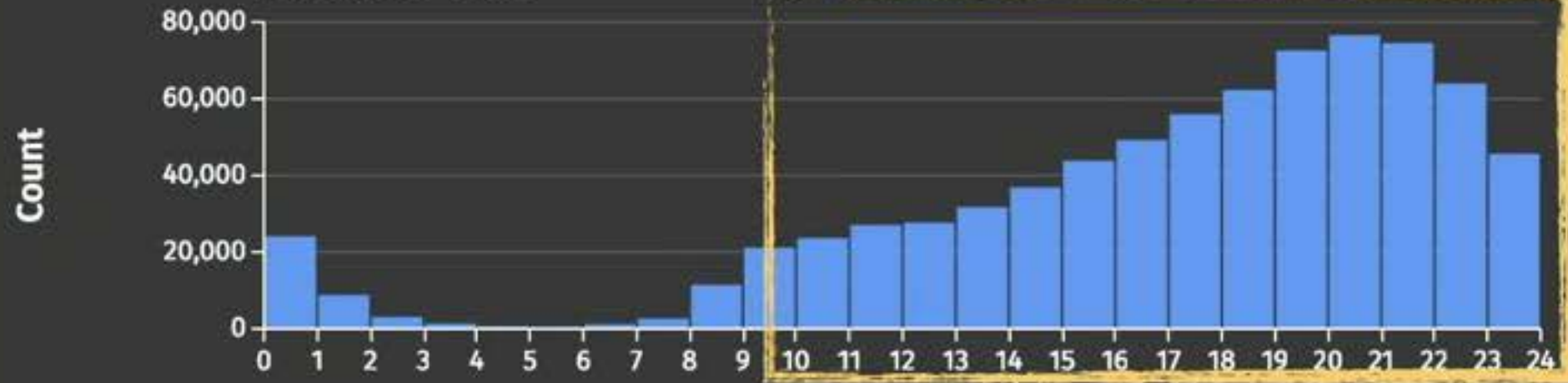




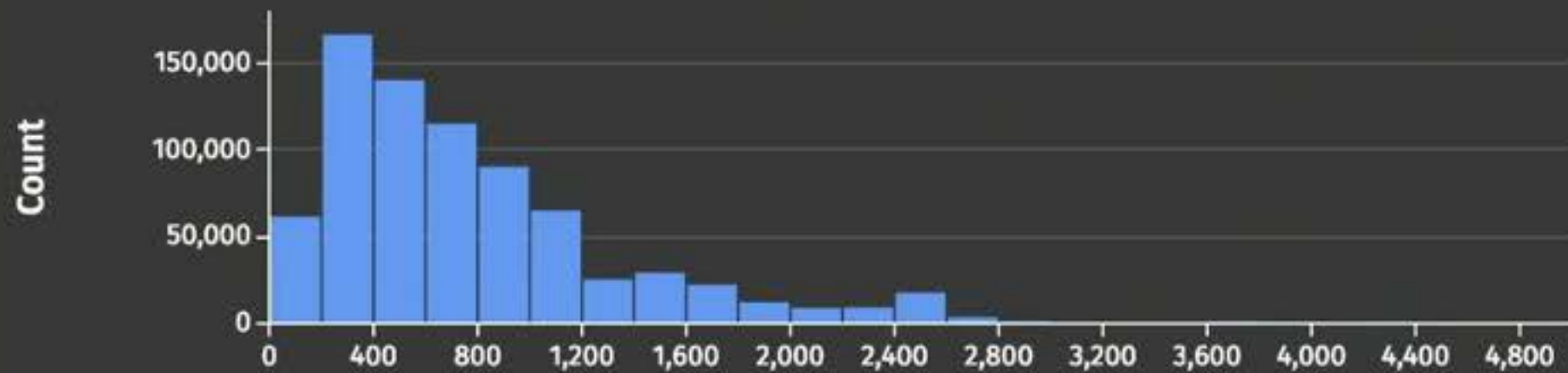
Arrival Delay in Minutes

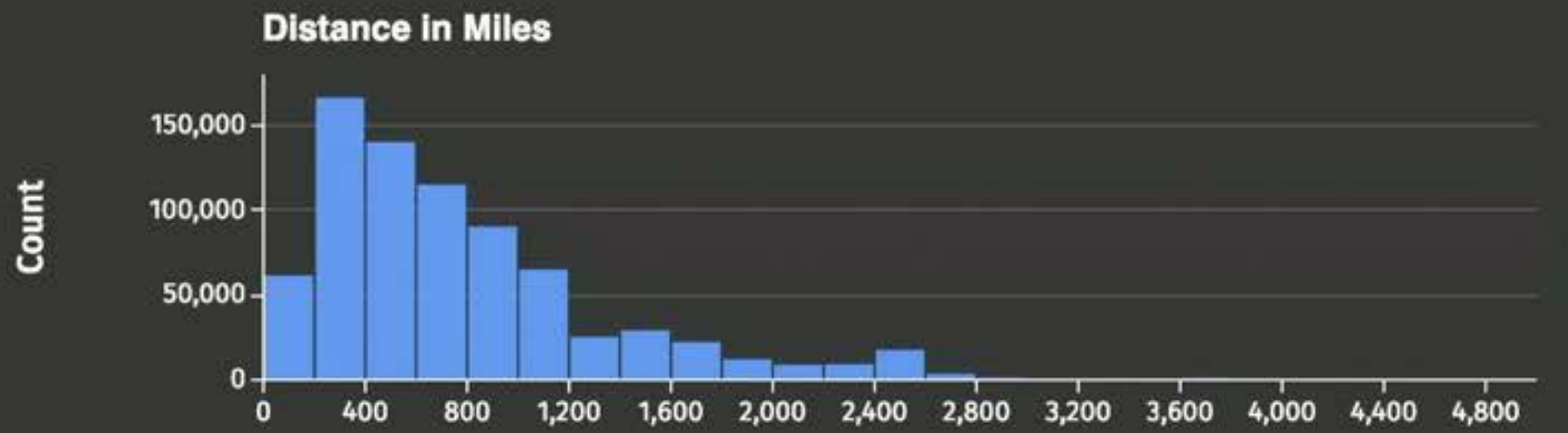
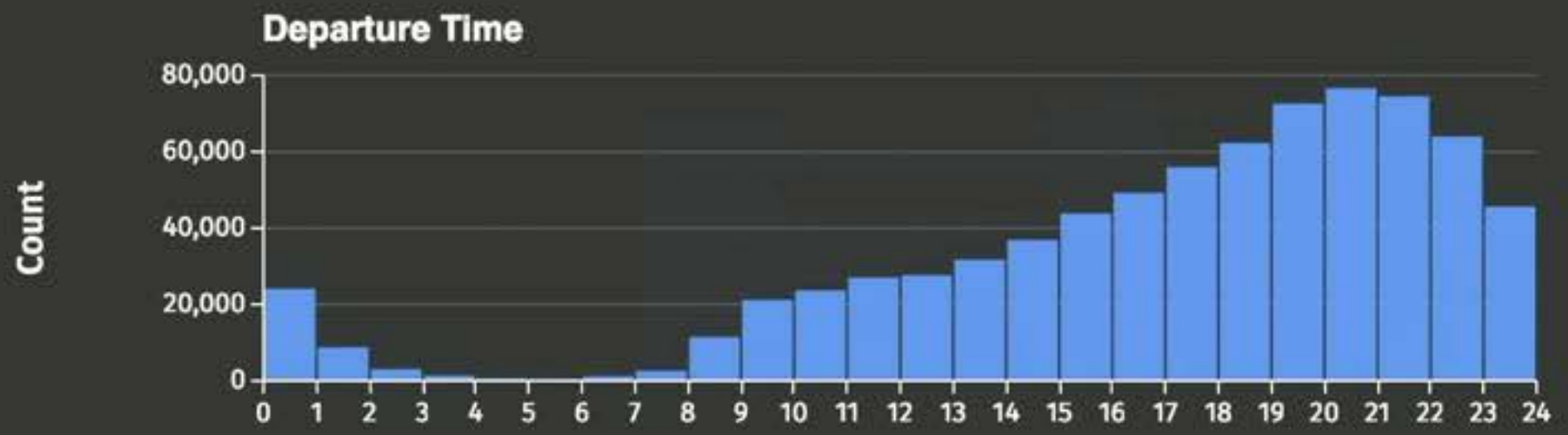
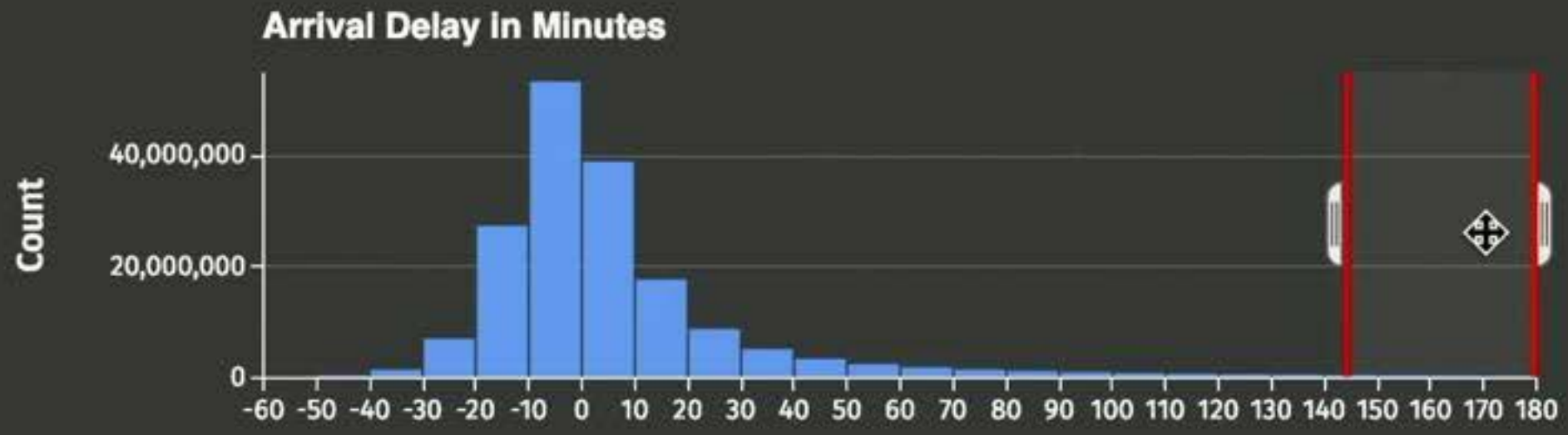


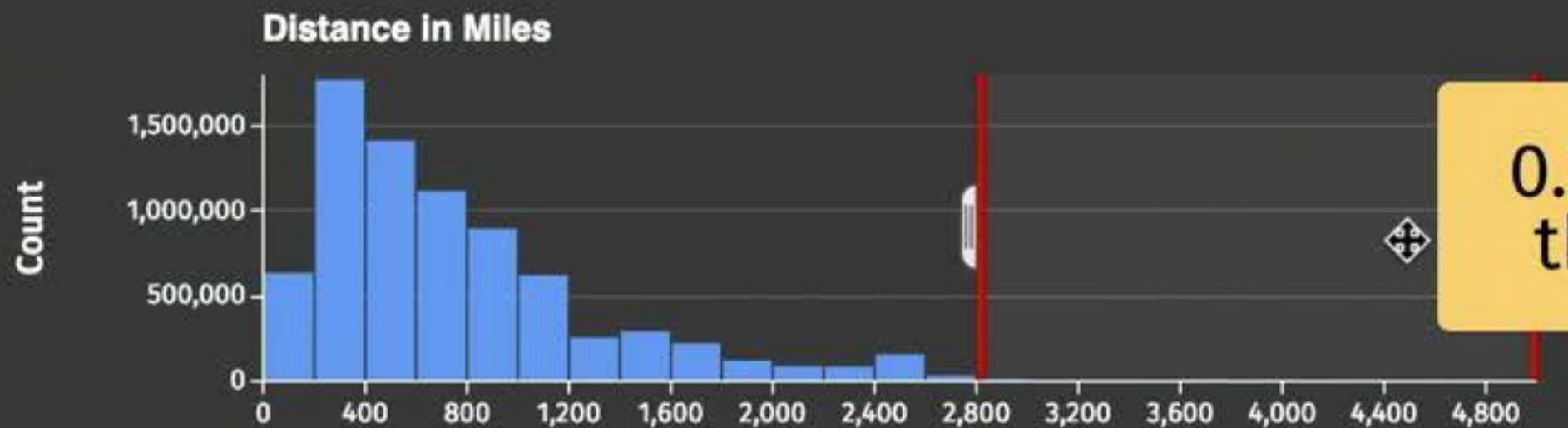
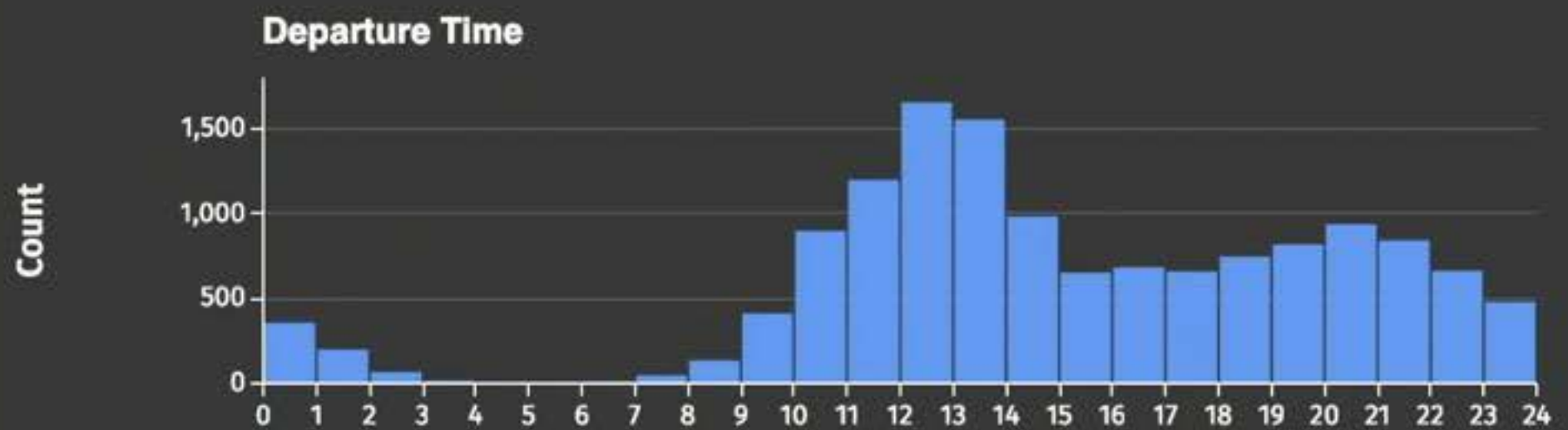
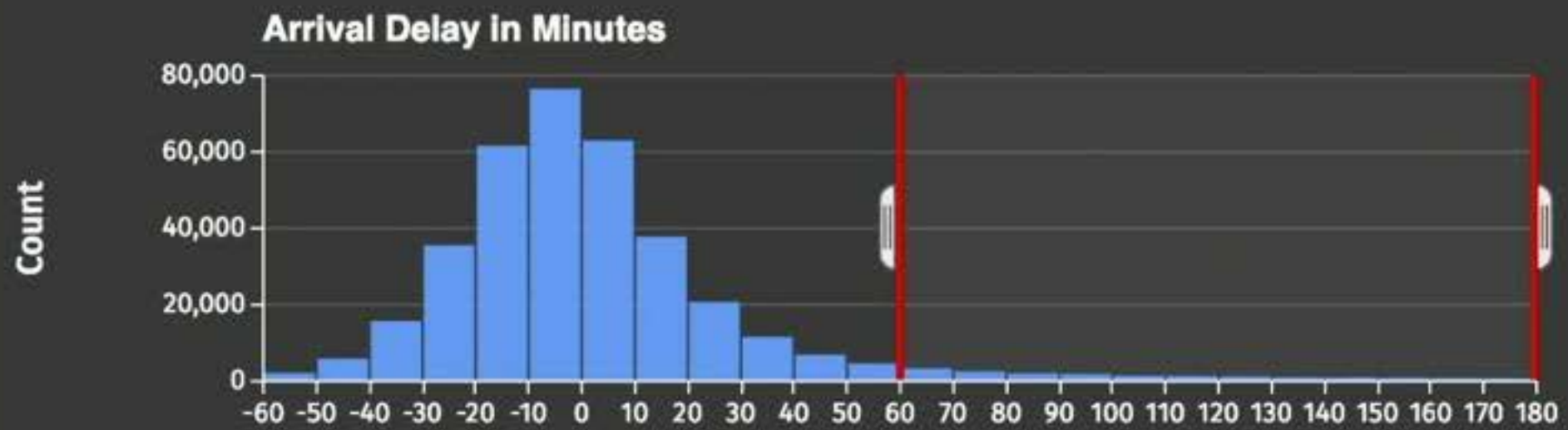
Departure Time



Distance in Miles

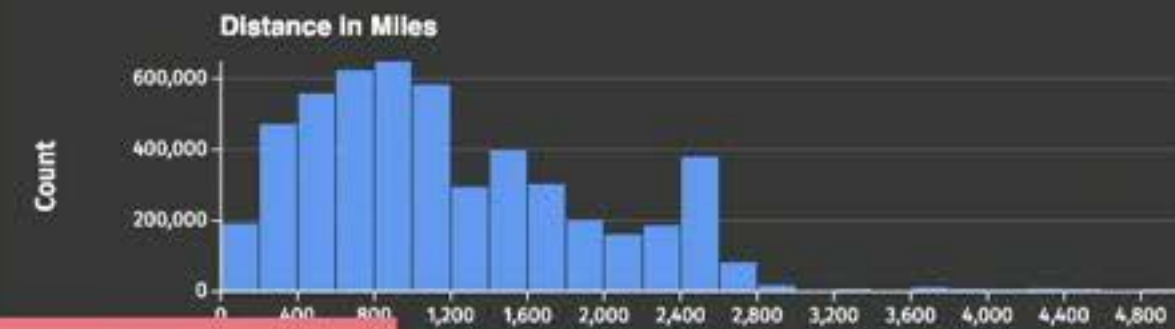
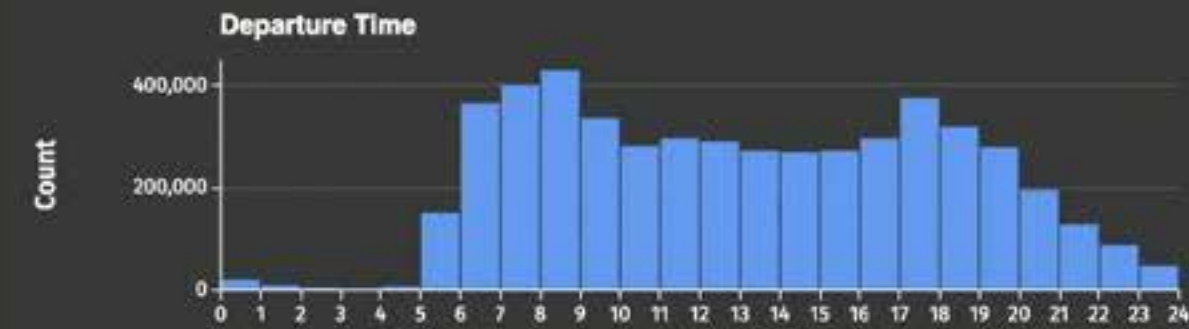
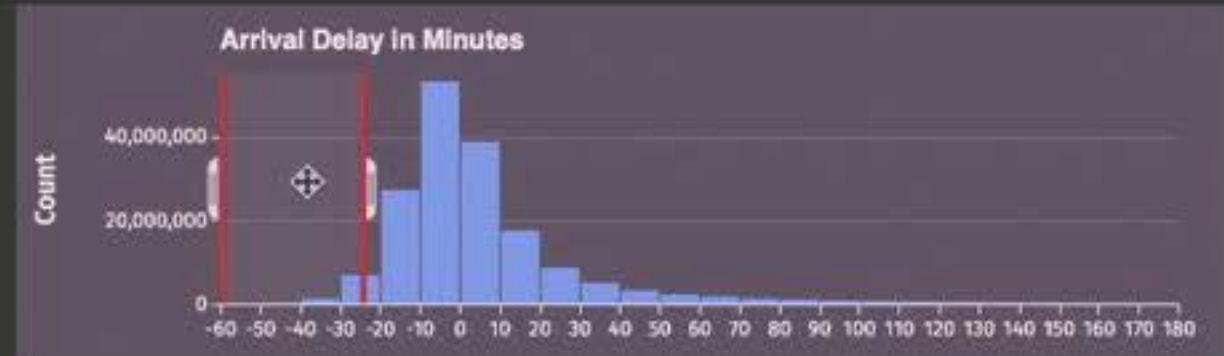




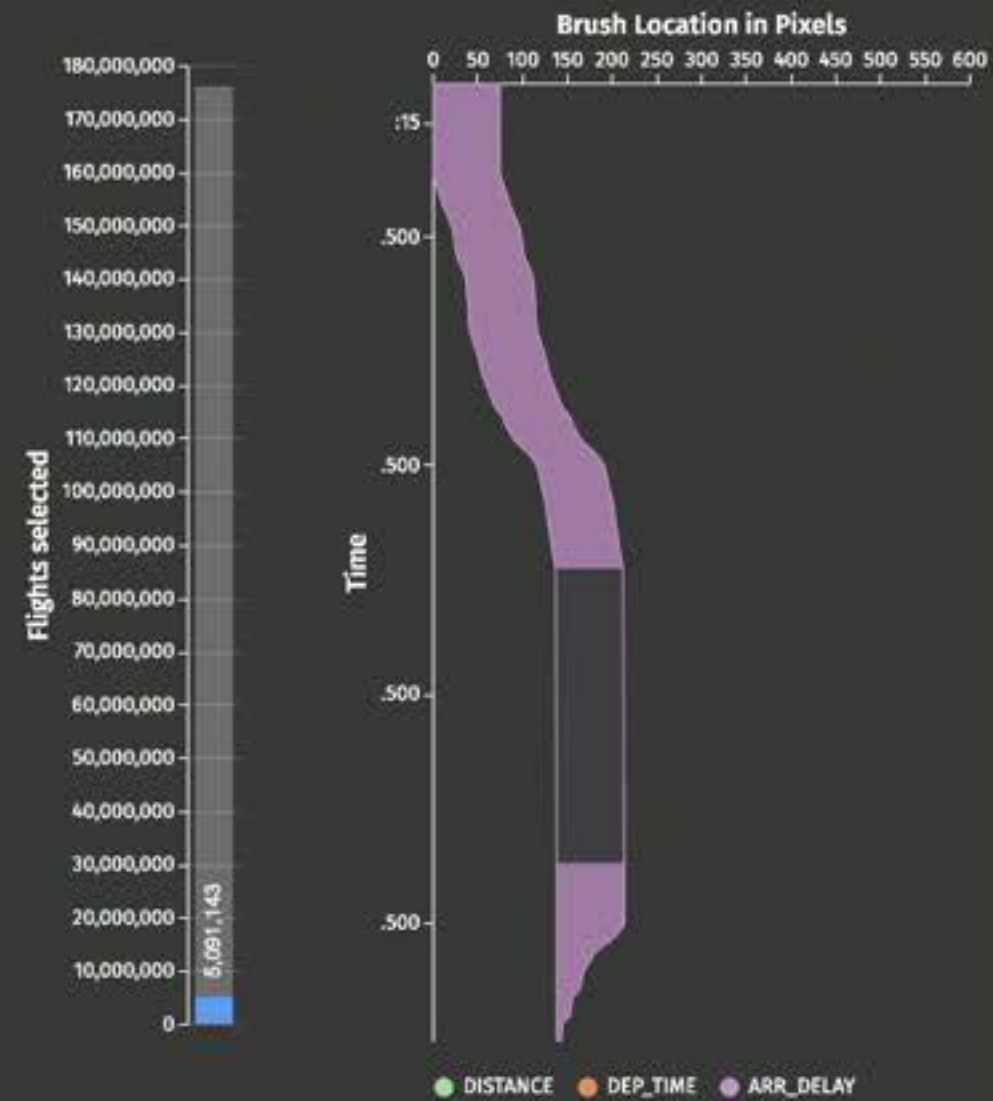


How can Falcon be real-time?

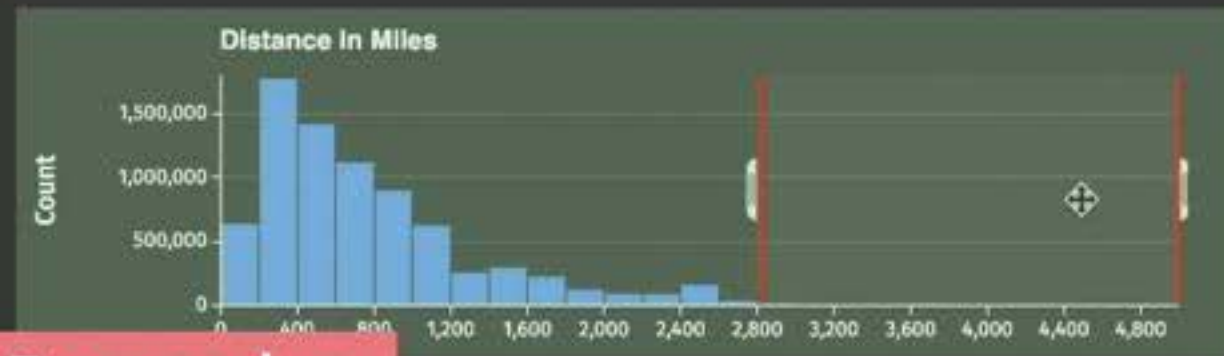
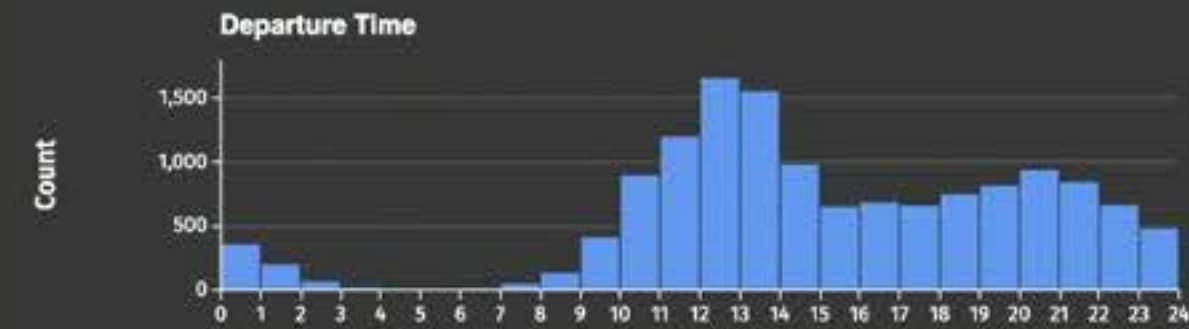
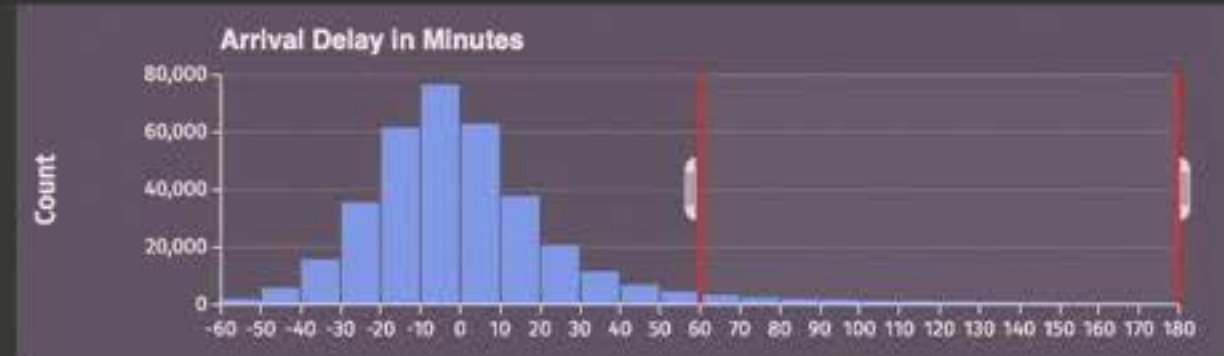
Falcon Interaction Log



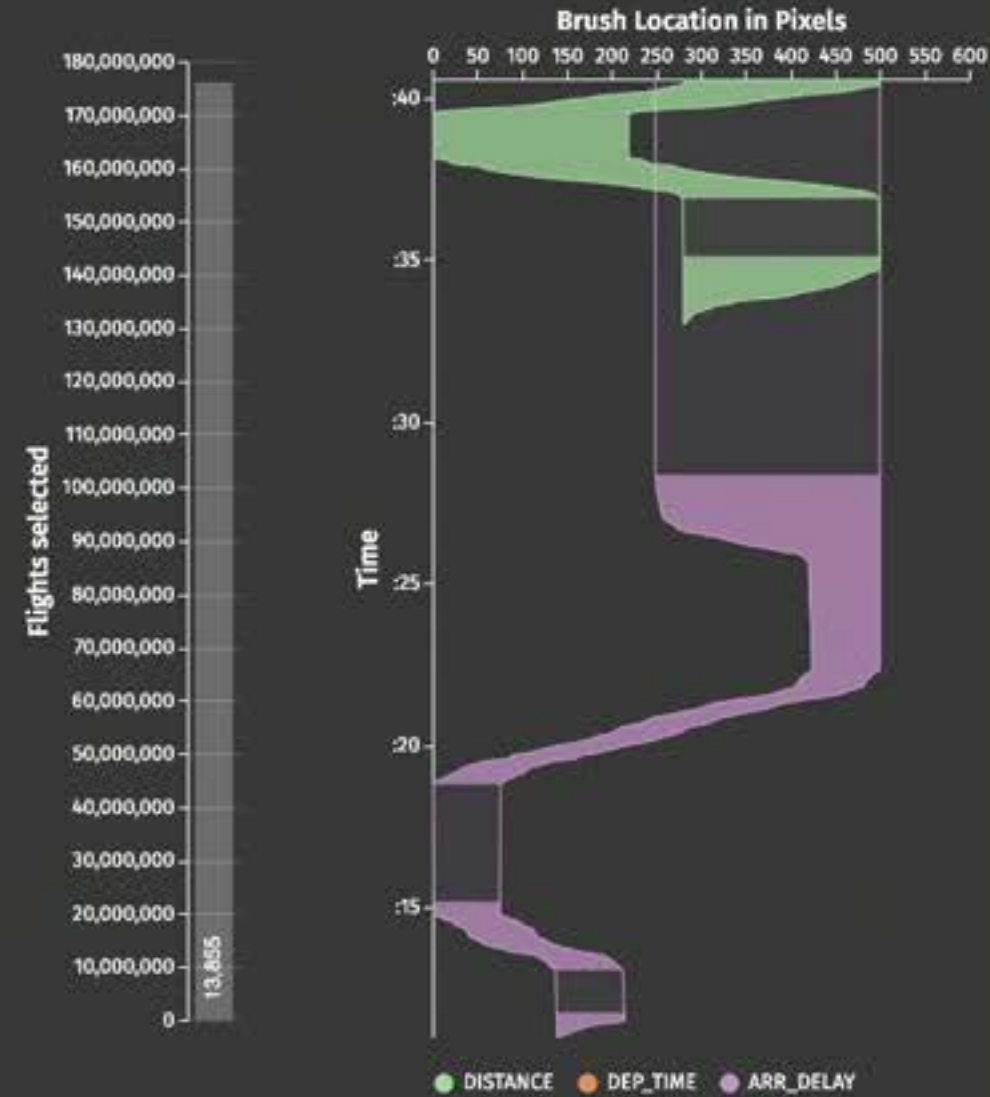
5x speedup



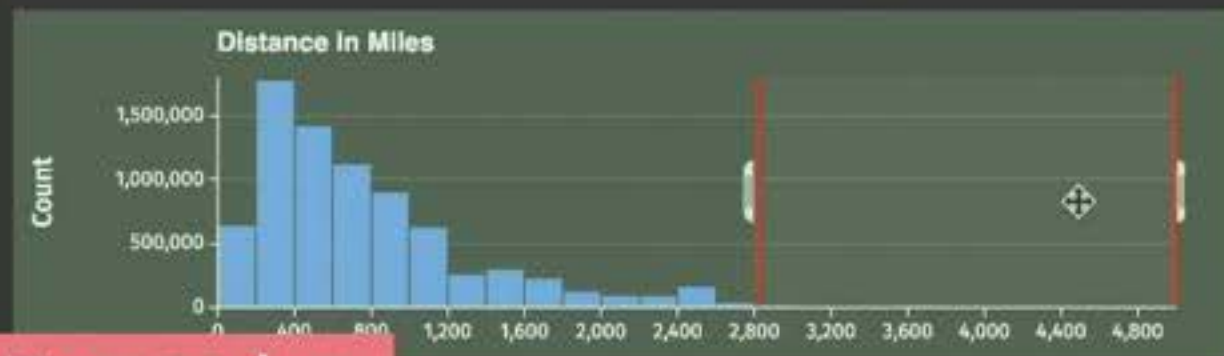
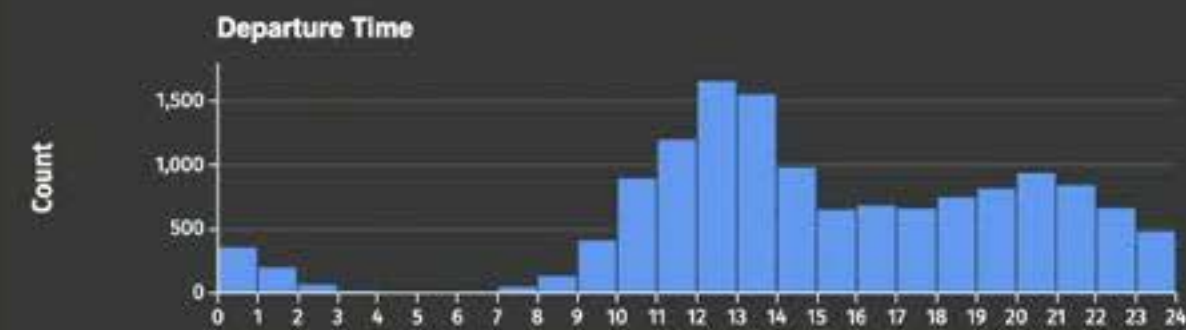
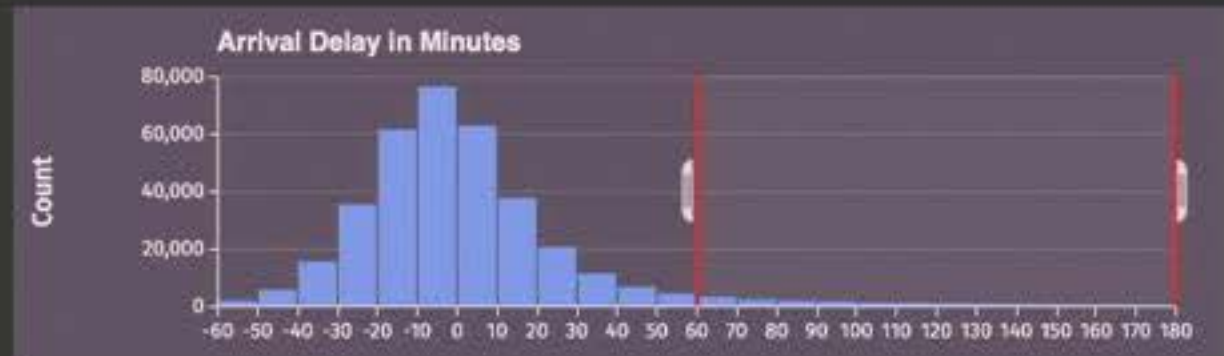
Falcon Interaction Log



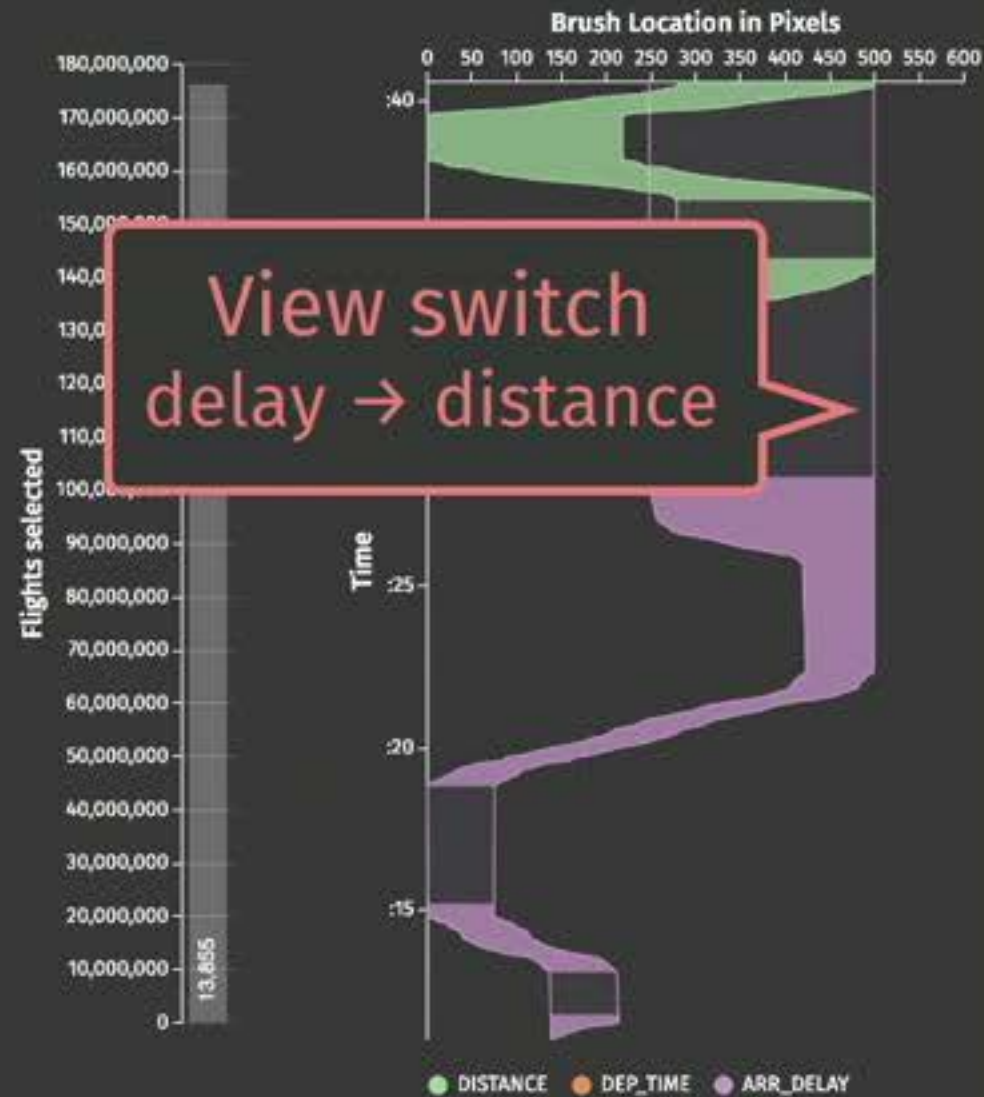
5x speedup



Falcon Interaction Log



5x speedup



Brushing interactions

- 👁️ Brushing is more common and people are sensitive to latencies.
- 💡 Prioritize **brushing** latency over **view switching** latency.

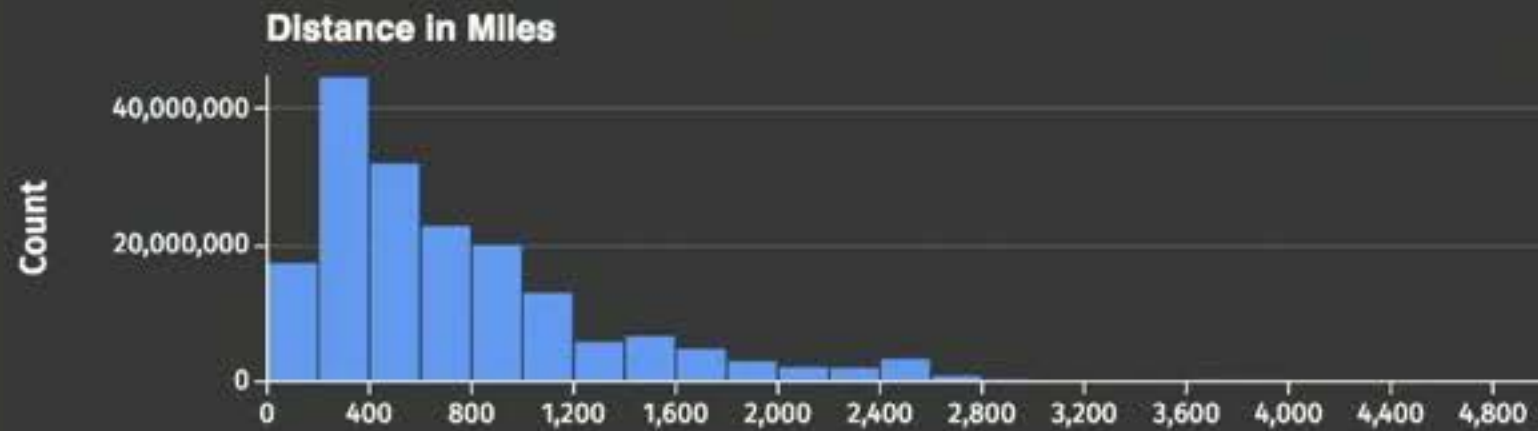
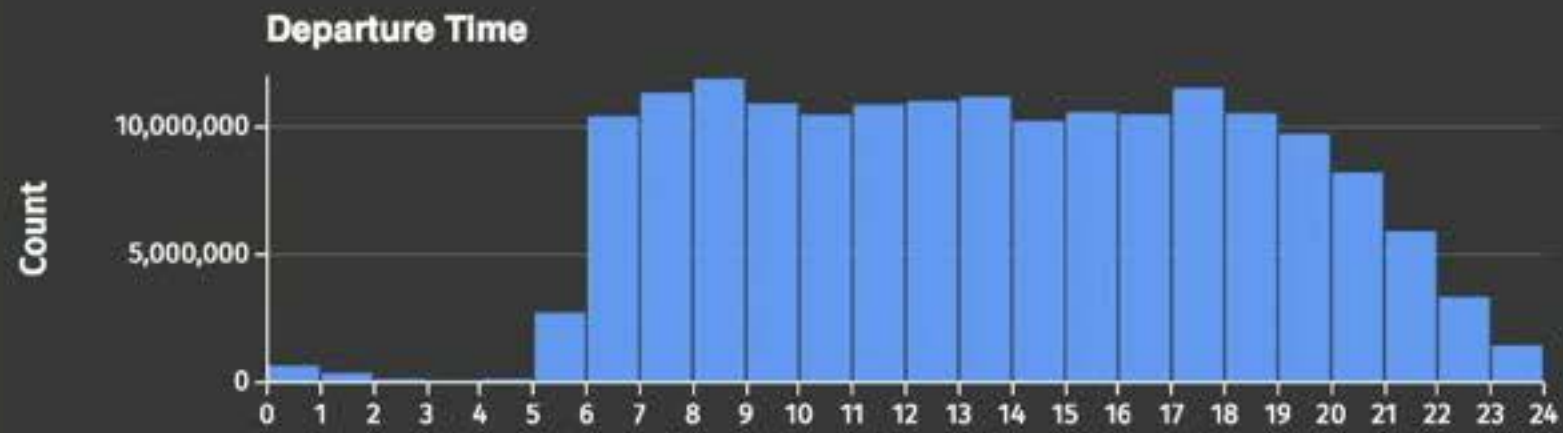
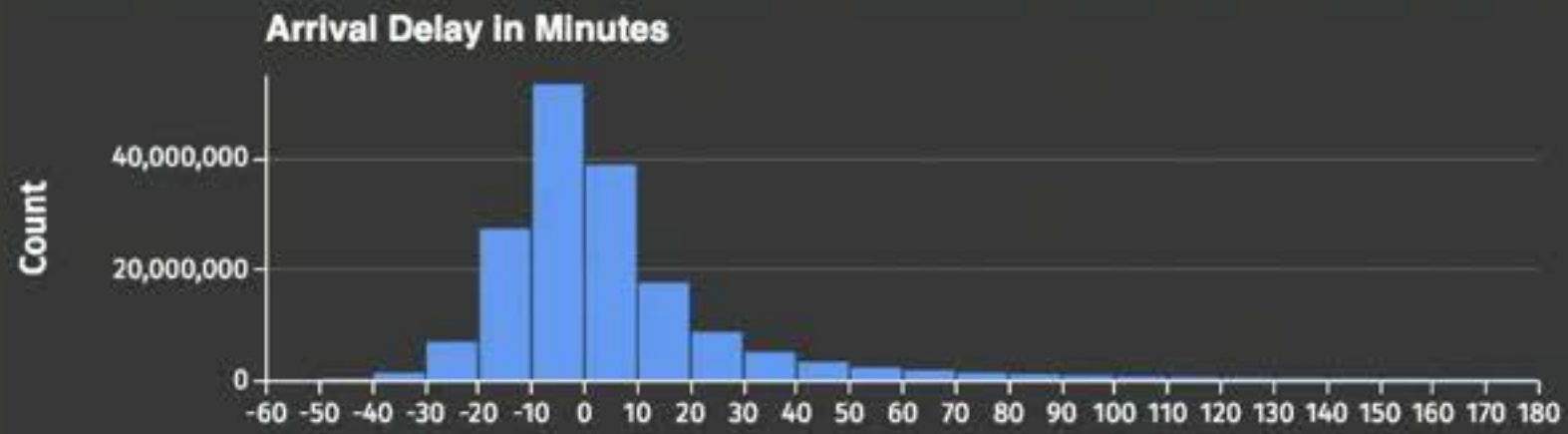
Key Idea:

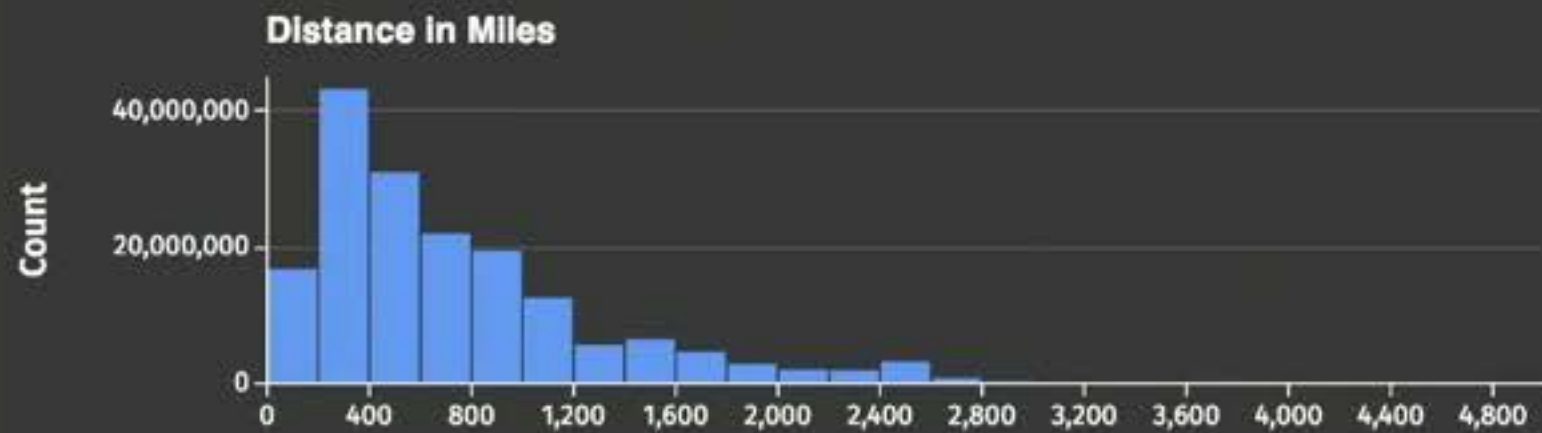
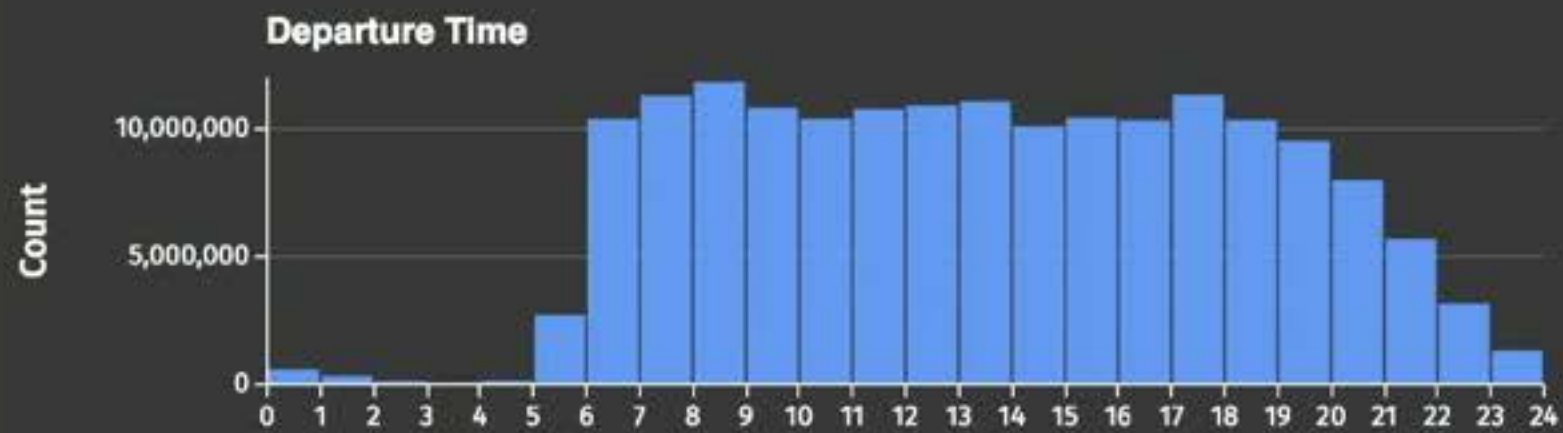
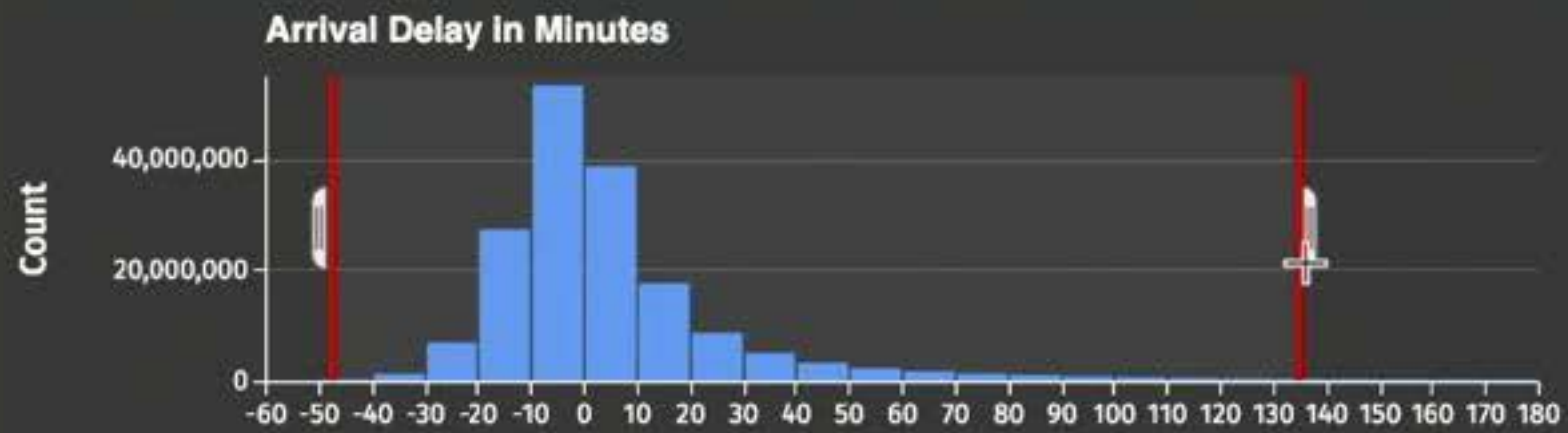
User-centered prefetching and indexing to support **all brushing interactions with one view.**

Re-compute if the user switches the view.



brushes in the precomputed view



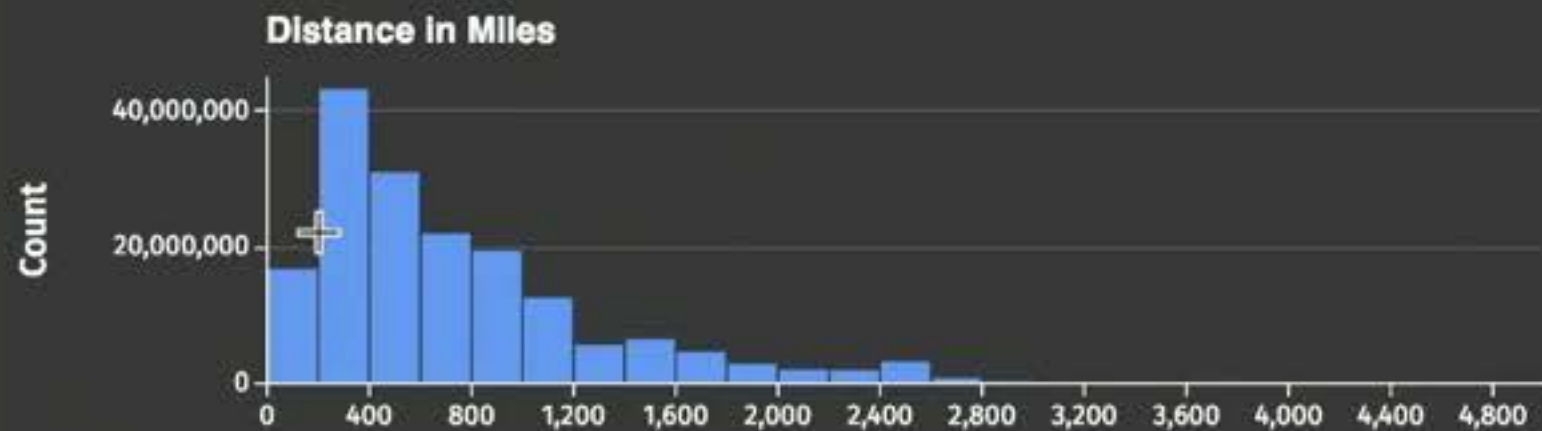
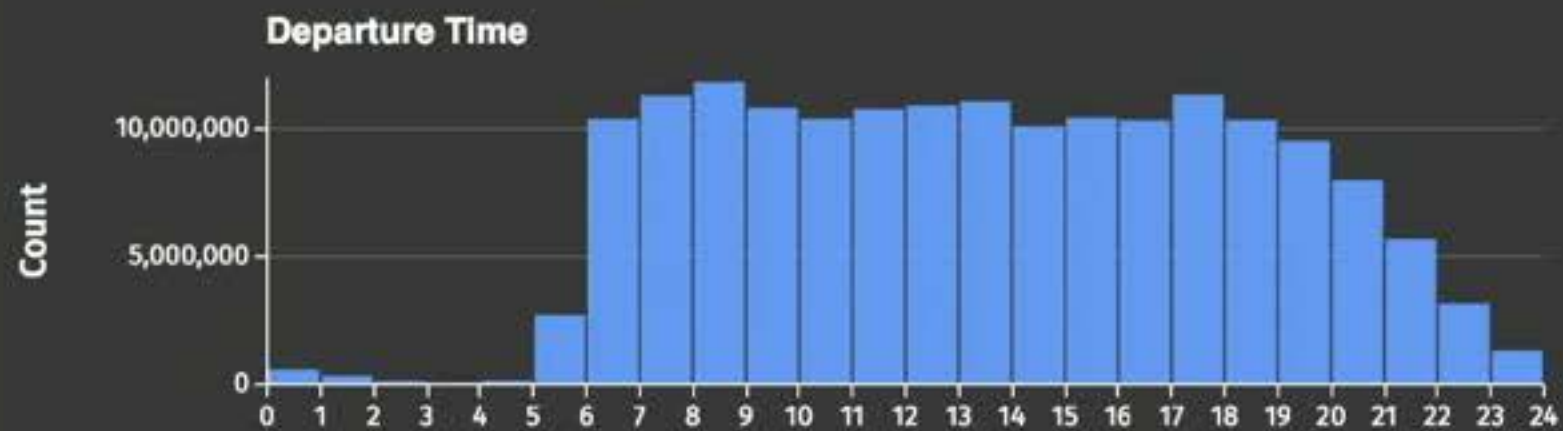
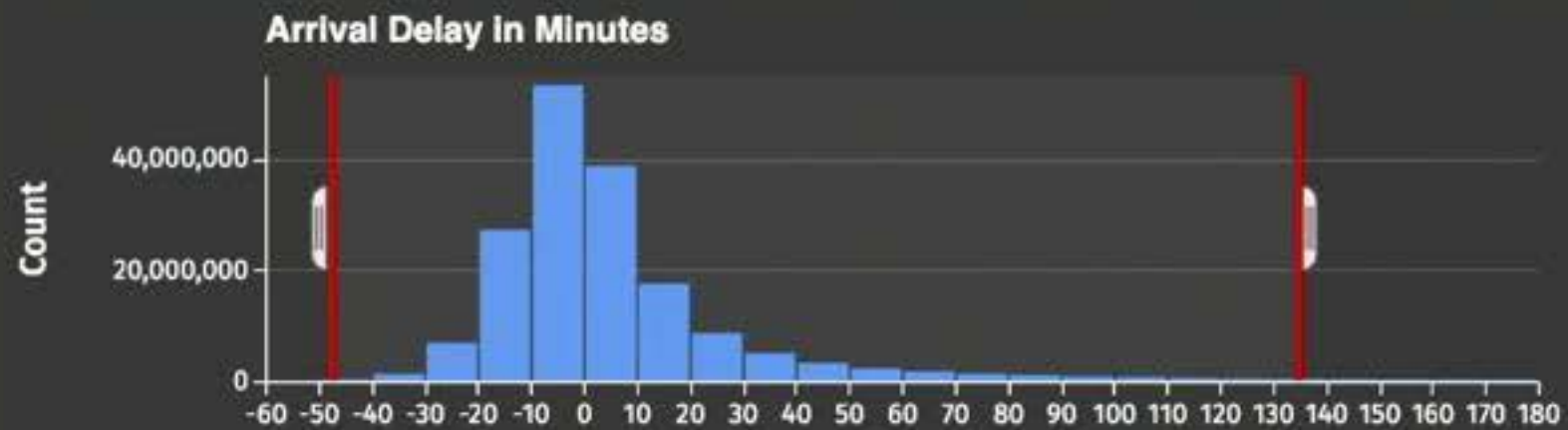


brushes in the precomputed view



serves requests from a data cube

Data Cube. Gray et al. 1997.



brushes in the precomputed view



serves requests from a data cube

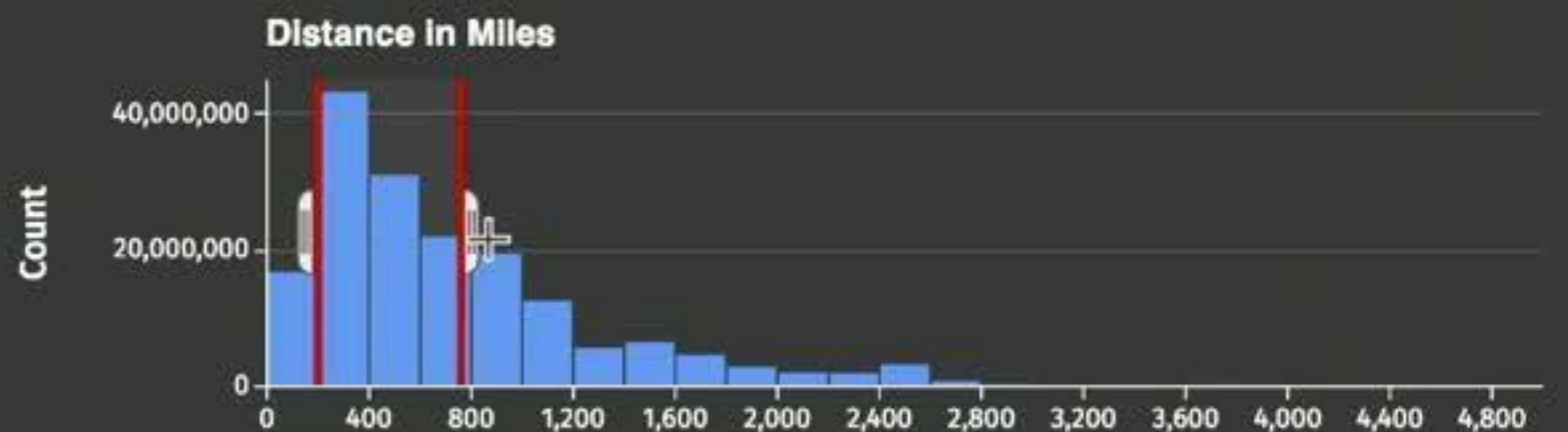
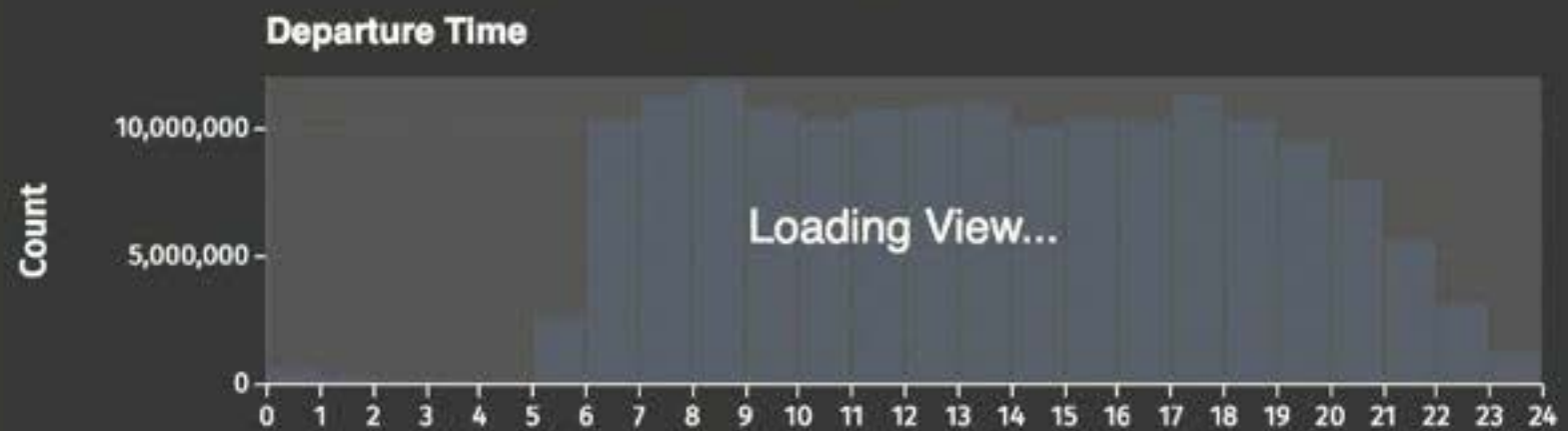
Data Cube. Gray et al. 1997.



interacts with a new view



computes new data cubes



brushes in the precomputed view



serves requests from a data cube

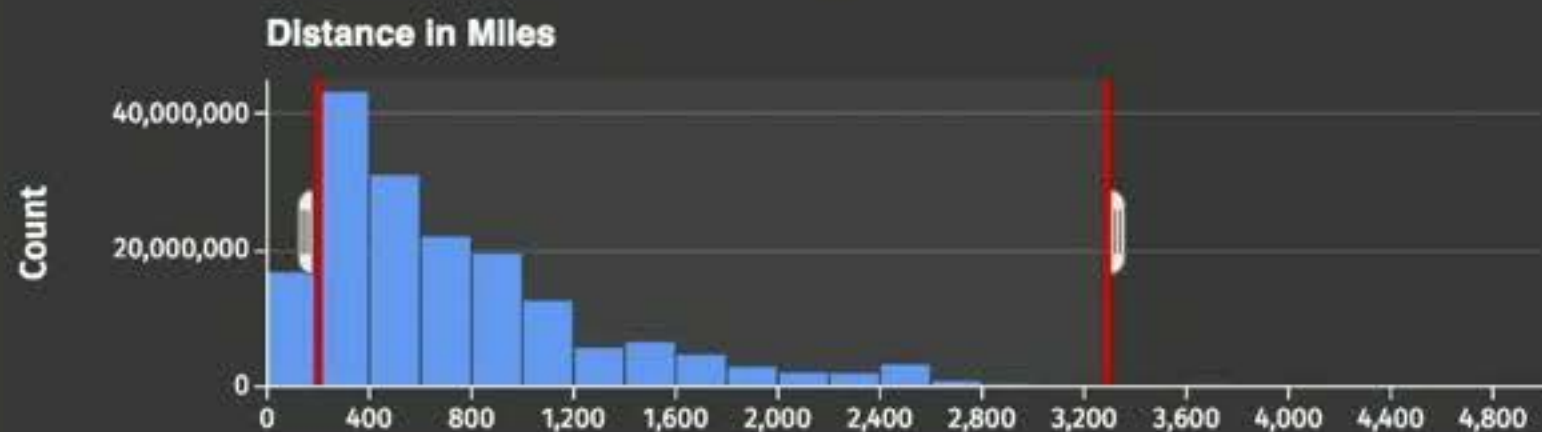
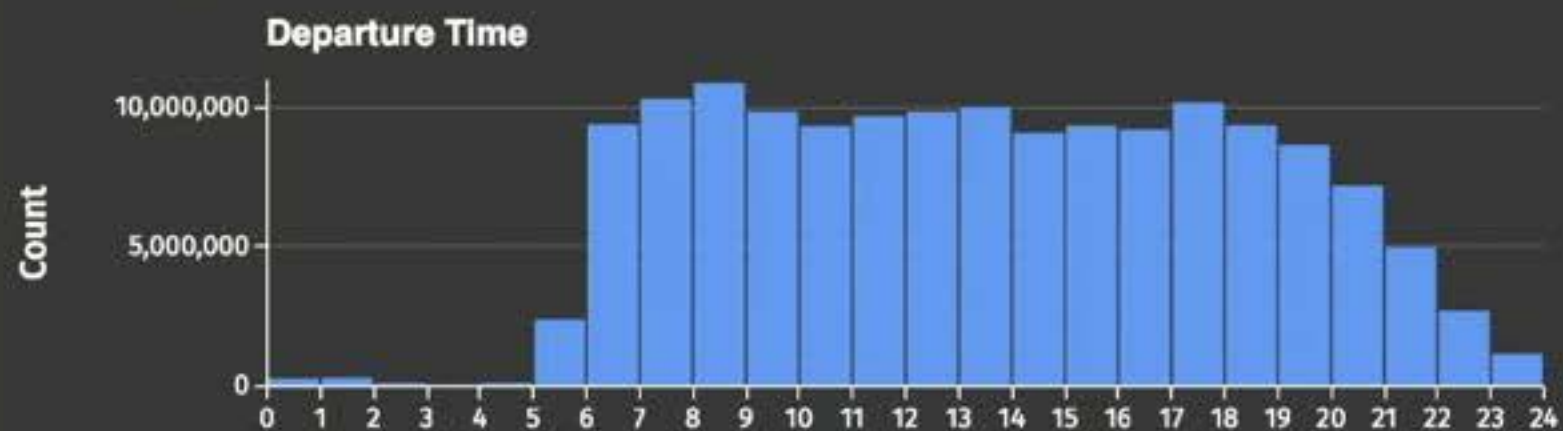
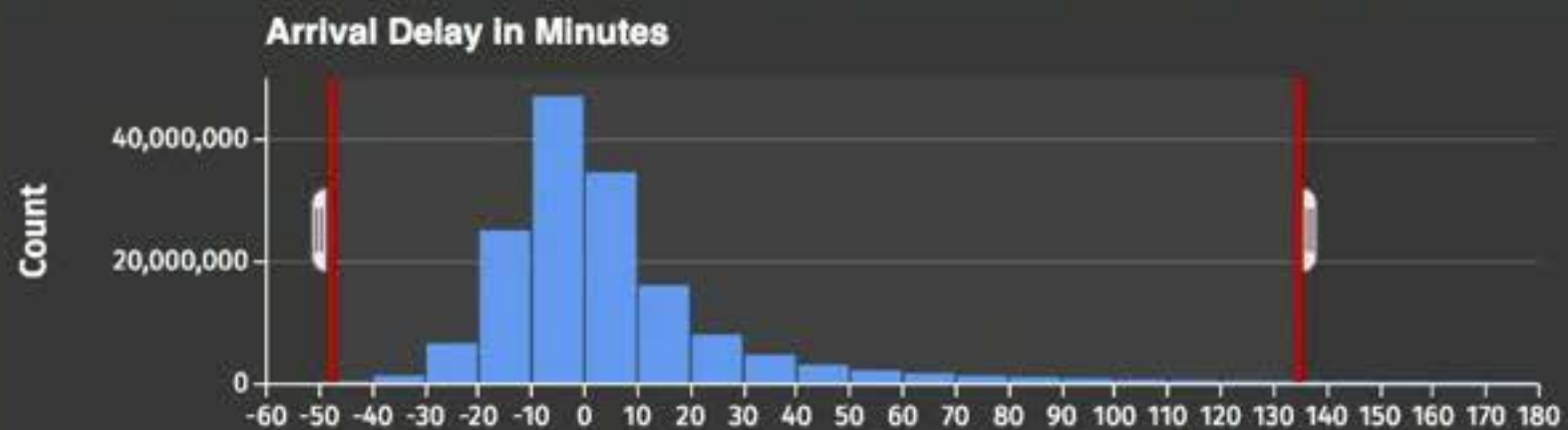
Data Cube. Gray et al. 1997.



interacts with a new view



computes new data cubes



brushes in the precomputed view



serves requests from a data cube

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*.

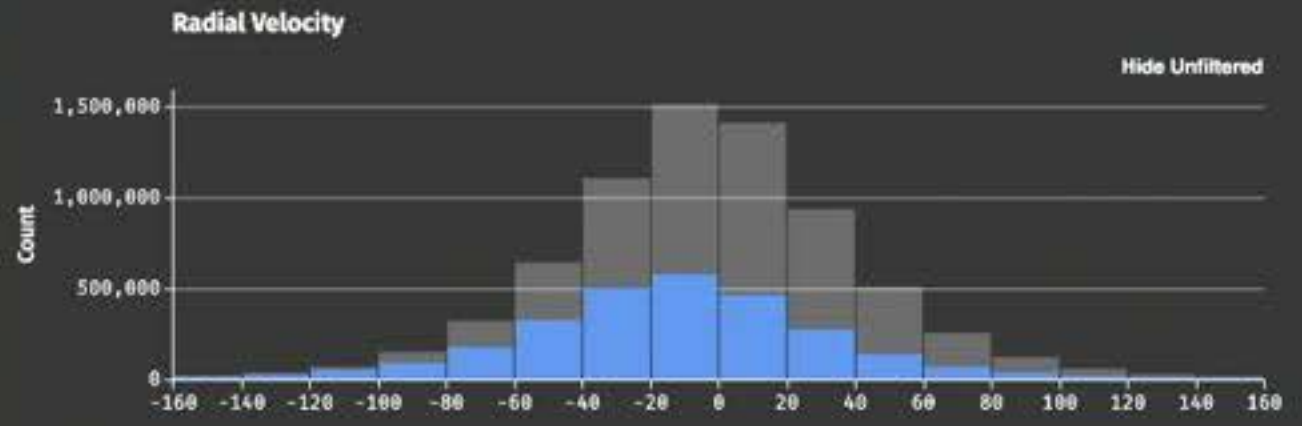
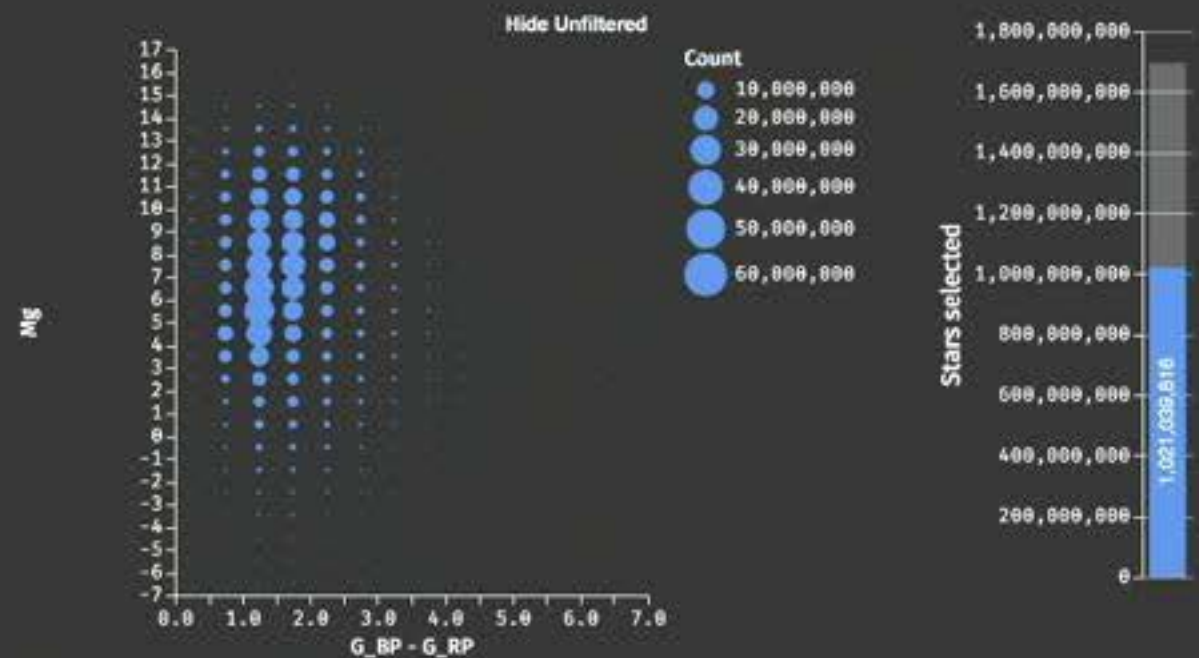
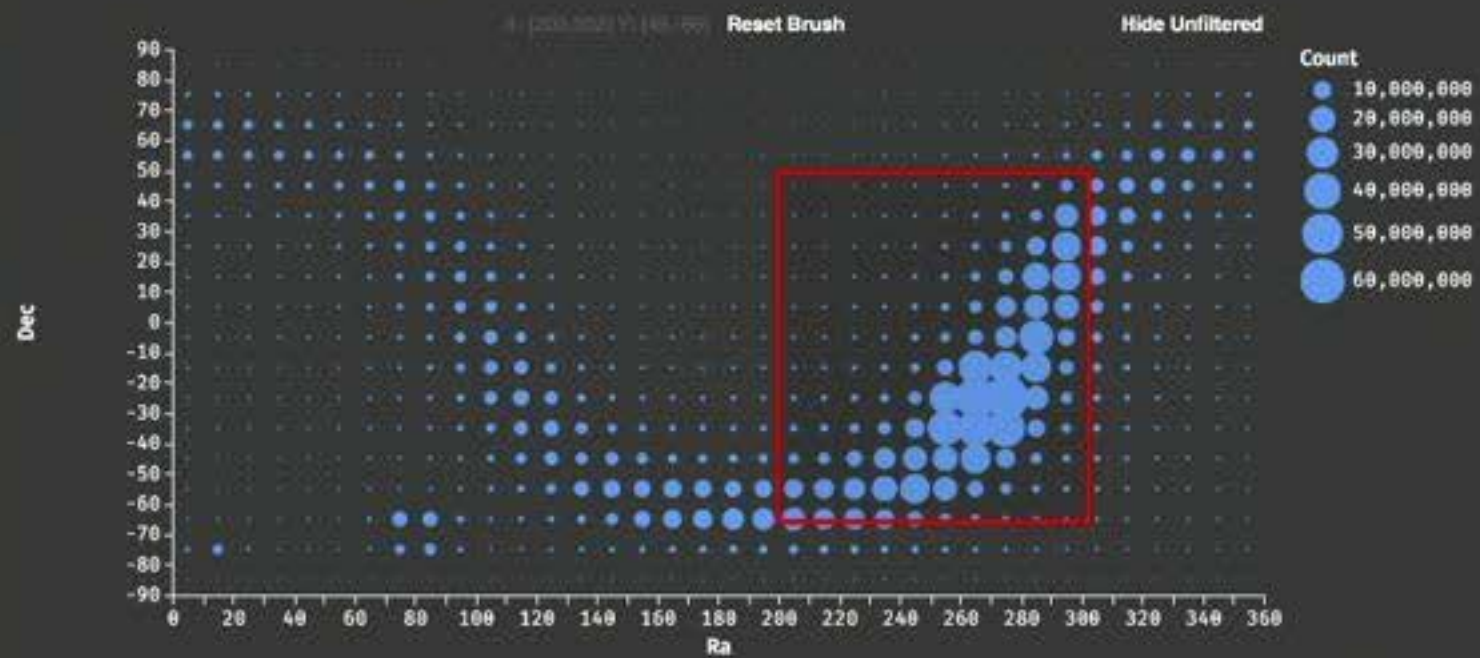
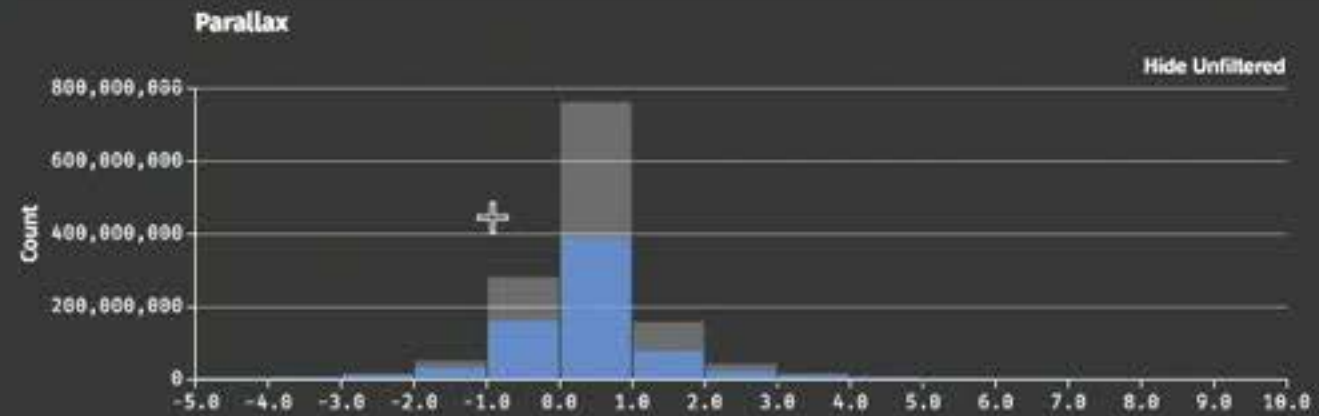
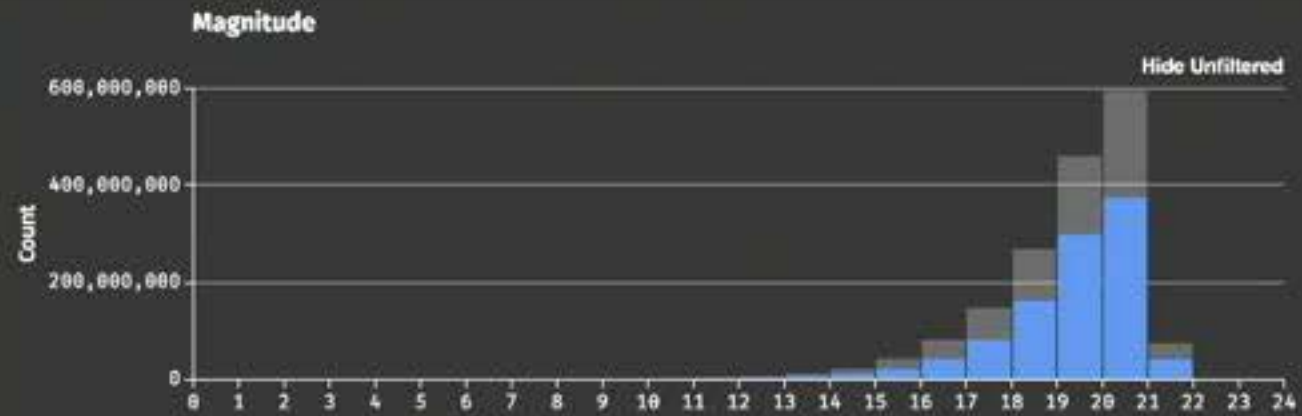
Dense cube. Decomposed into overlapping cubes.

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.

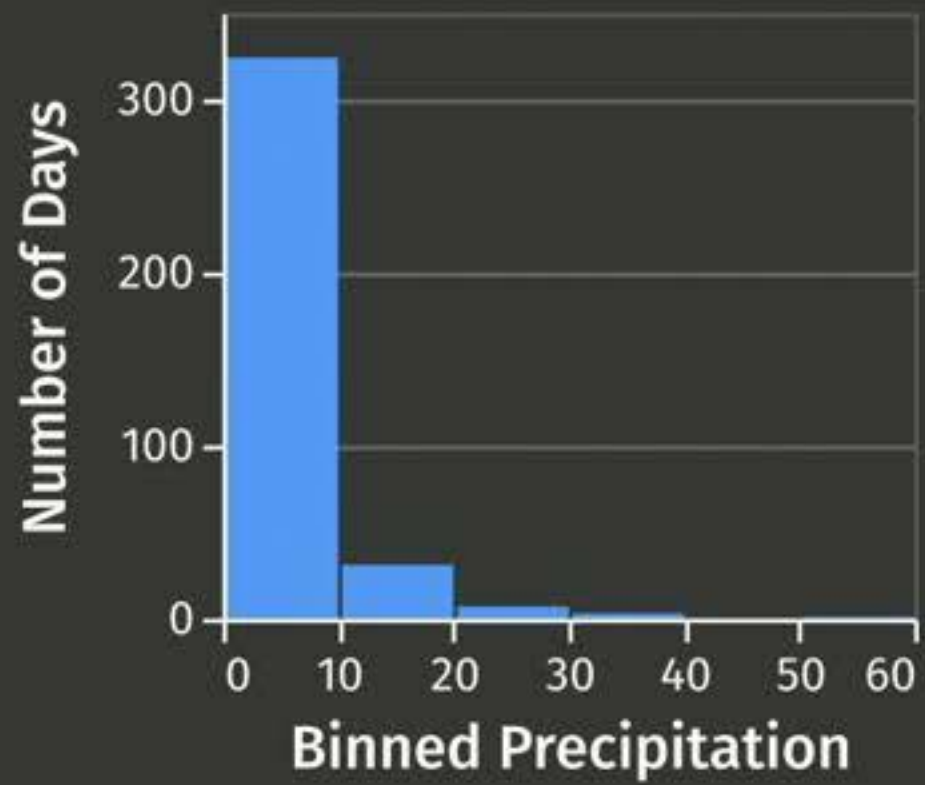


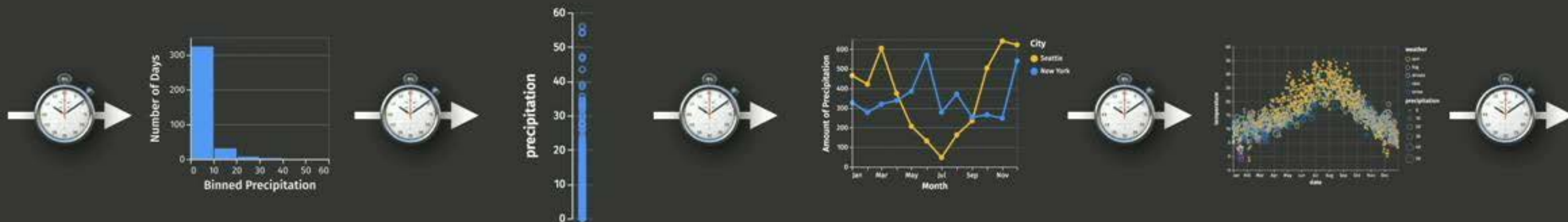
1.7 B stars.
1.2 TB of data.
Visualizations running in my browser.
Data stored in OmniSci database.

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

Data Platform Engineer at Stitch Fix

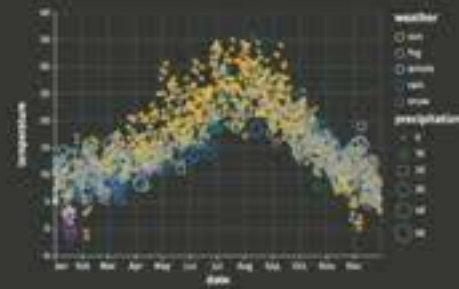
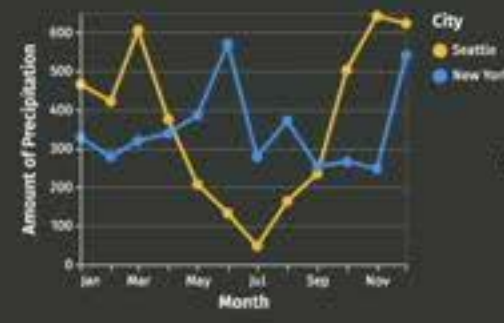
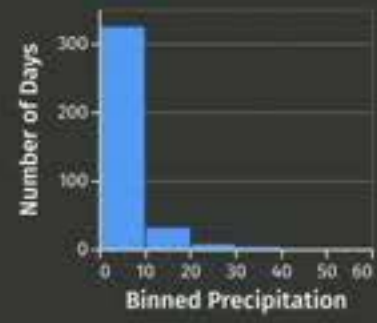
What if data too large to even query it in a reasonable time?



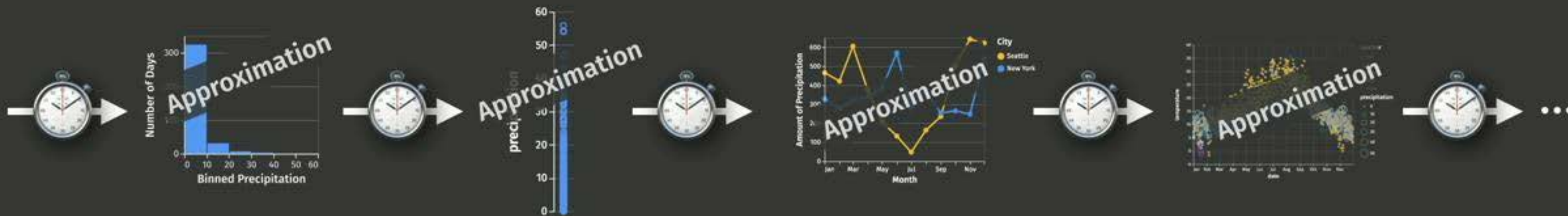


Latencies reduce engagement
and lead to fewer observations.

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



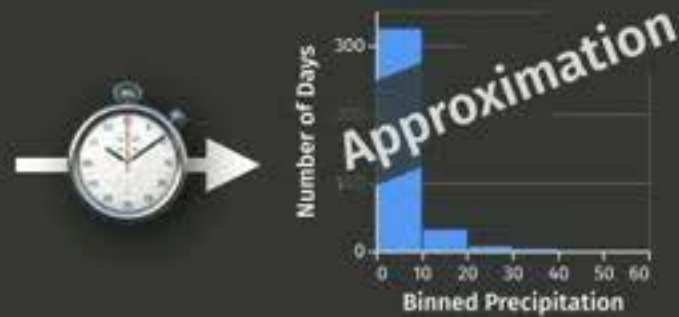
...



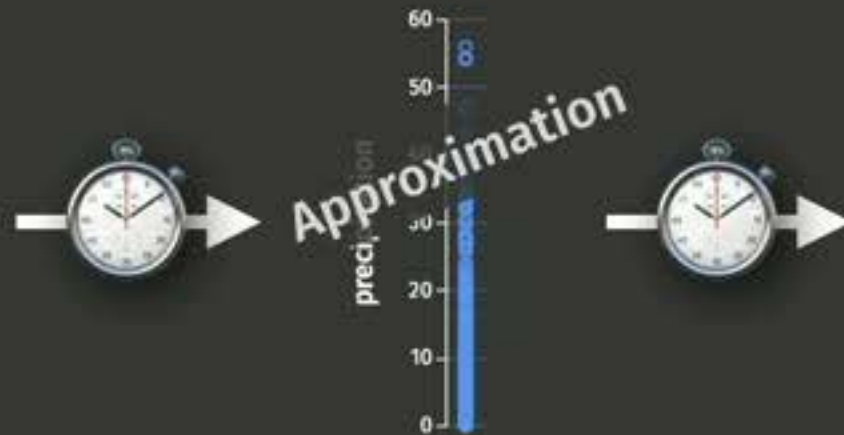
Approximation. Accuracy \rightarrow Speed

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

Small chance
of error



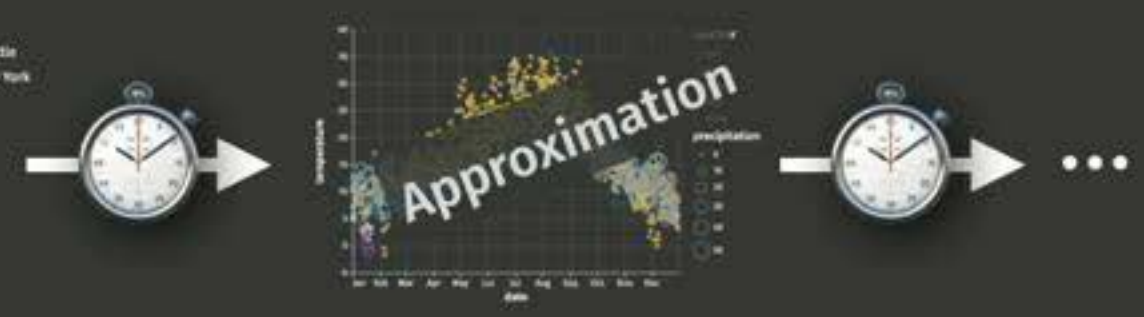
Small chance
of error



Small chance
of error



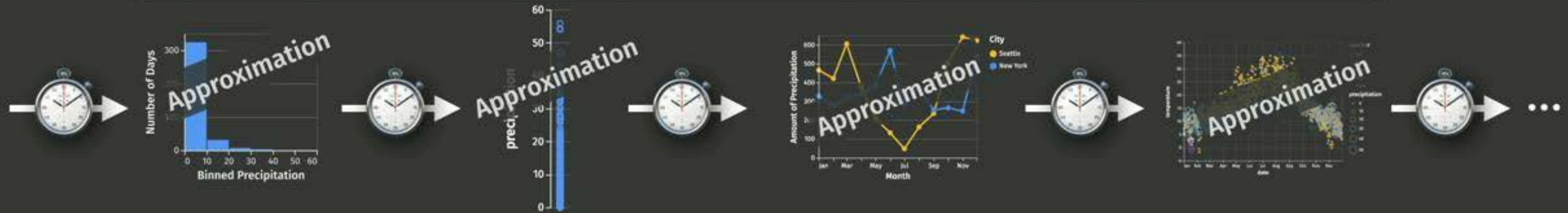
Small chance
of error



Approximation. Accuracy \rightarrow 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 \rightarrow 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.



This glass
is half full



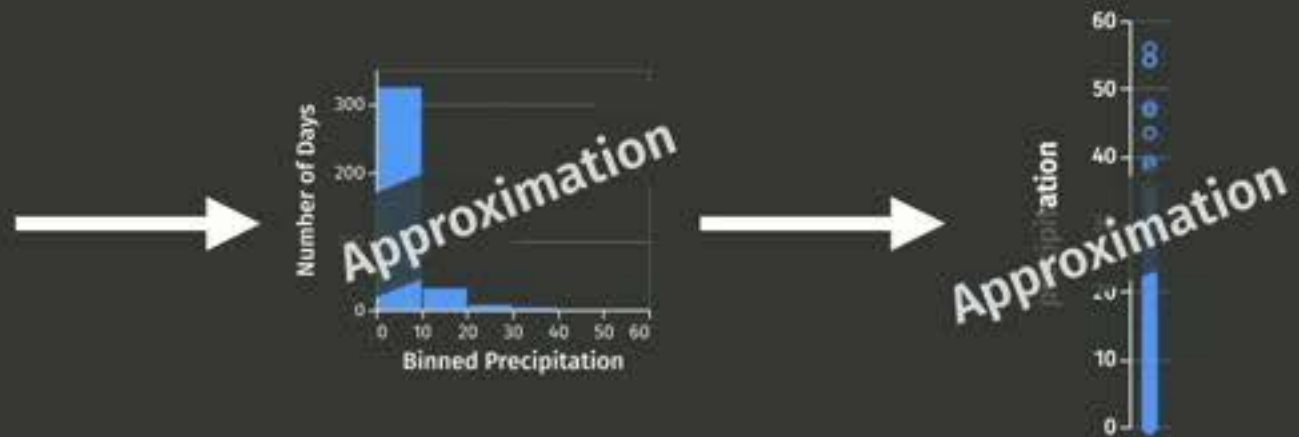
Optimistic Visualization

Trust but Verify

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

Optimistic Visualization

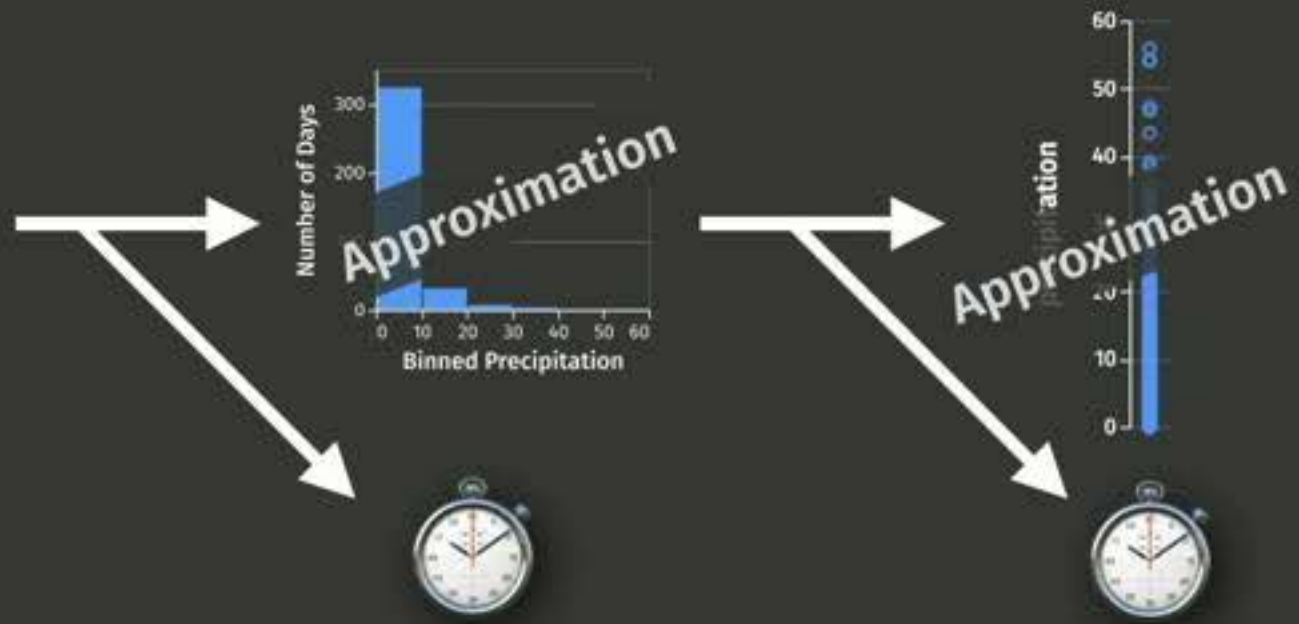
Trust but Verify. Moritz et al. *CHI 2017*.



1. Analysts uses initial estimates.

Optimistic Visualization

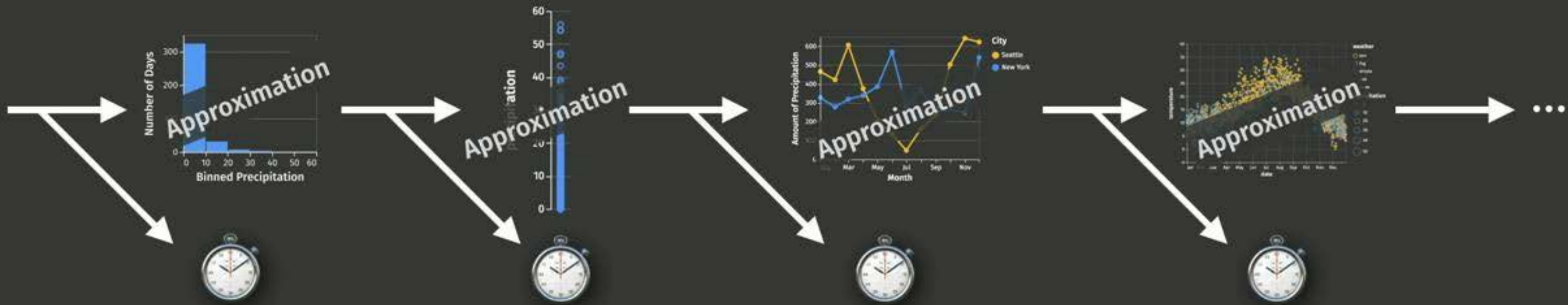
Trust but Verify. Moritz et al. *CHI 2017*.



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

Optimistic Visualization

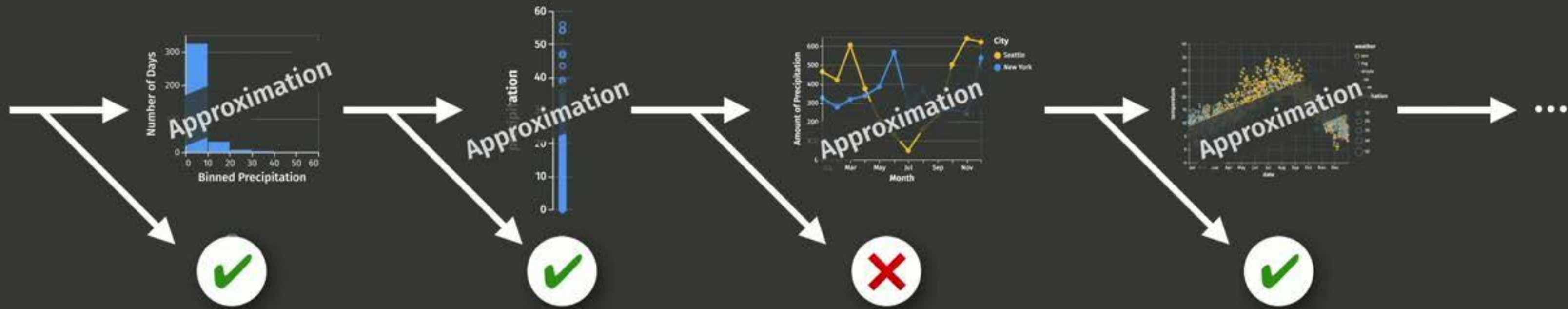
Trust but Verify. Moritz et al. *CHI 2017*.



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

Data: FAAData

Type to filter schema...

- # Year
- # Quarter
- # Month
- # DayOfMonth
- # DayOfWeek
- FlightDate
- A UniqueCarrier
- # AirlineID
- A Carrier
- A TailNum
- # FlightNum
- # OriginAirportID
- # OriginAirportSeqID
- # OriginCityMarketID
- A Origin
- A OriginCityName
- A OriginState
- A OriginStateFips
- A OriginStateName
- # OriginWac
- # DestAirportID
- # DestAirportSeqID
- # DestCityMarketID
- A Dest
- A DestCityName
- A DestState
- A DestStateFips
- A DestStateName
- # DestWac

Heatmap

X-Axis: Field: DepDelay, Binning: 64, Sort by key:

Y-Axis: Field: ArrDelay, Binning: 40, Sort by key:

Value: Function: Count

Persistent Filters: E.G. AND(Carrier \$IN\$[ha, d1])(DepDelay>=0)

Zoom: clear, Capture as Filter

(ArrDelay SRNGS [[-148.80619517543857,390.49205043859655]])

(DepDelay SRNGS [[-19.819658218570382,187.25649037534237]])

Load more data

Expect some errors: 2.3%

What have you learned? Remember

Approximate Values

Expected Error

Relative

Massive drop off after Sep 2001

Exact data loaded (18s)

3 decades of flights

Exact data loaded (50s)

Spike near 0 minutes

Loading exact data...

Clear History, Reset App

Pangloss Visualizes Uncertainty



Pangloss shows a History of Previous Charts

Data: FAADData

Type to filter schema...

- # Year
- # Quarter
- # Month
- # DayOfMonth
- # DayOfWeek
- FlightDate
- UniqueCarrier
- # AirlineID
- Carrier
- TailNum
- FlightNum
- OriginAirportID
- OriginAirportSeqID
- OriginCityMarketID
- Origin
- OriginCityName
- OriginState
- OriginStateFips
- OriginStateName
- # OriginWac
- # DestAirportID
- # DestAirportSeqID
- # DestCityMarketID
- Dest
- DestCityName
- DestState
- DestStateFips
- DestStateName
- # DestWac

Heatmap

X-Axis
Field: DepDelay
Binning: 64
Sort by key:

Y-Axis
Field: ArrDelay
Binning: 40
Sort by key:

Value
Function: Count

Persistent Filters
E.g. AND(Carrier IN\$[ha, d1])(DepDelay>=0)

Zoom
clear Capture as Filter
(ArrDelay SRNGS
[[-148.80619517543857,390.49205043859655]])
(DepDelay SRNGS
[[-19.819658218570382,187.25649037534237]])

Load more data **Expect some errors: 2.3%**

what have you learned? Remember

Approximate Values

ArrDelay

DepDelay

25M
20M
15M
10M
5M
0

[10,20], [30,40]
534k±72k

Expected Error
Relative

ArrDelay

DepDelay

400k
300k
200k
100k
0

Massive drop off after Sep 2001

Exact data loaded (18s)

3 decades of flights

Exact data loaded (50s)

Spike near 0 minutes

Loading exact data...

Clear History Reset App

In Pangloss, Analysts can Confirm results

Data: FAADData

Type to filter schemas

- # Year
- # Quarter
- # Month
- # DayofMonth
- # DayOfWeek
- FlightDate
- A UniqueCarrier
- # AirlineID
- A Carrier
- A TailNum
- # FlightNum
- # OriginAirportID
- # OriginAirportSeqID
- # OriginCityMarketID
- A Origin
- A OriginCityName
- A OriginState
- A OriginStateFips
- A OriginStateName
- # OriginWac
- # DestAirportID
- # DestAirportSeqID
- # DestCityMarketID
- A Dest
- A DestCityName
- A DestState
- A DestStateFips
- A DestStateName
- # DestWac
- A CRSDepTime

Heatmap

X-Axis
Field: DepDelay
Binning: 54
Sort by key:

Y-Axis
Field: ArrDelay
Binning: 54
Sort by key:

Value
Function: Count

Persistent Filters
e.g. AND(Carrier \$INS[ha, d1])(DepDelay>=0)

Zoom
(ArrDelay SRNGS
[[-148.80619517543857, 390.49205043859655]])
(DepDelay SRNGS
[[-19.819658218570382, 187.25649037534237]])

What have you learned?

The visualization is read only because you're looking at the history. [Return to the working vis](#) or make a [copy of the current chart](#).

Exact Data

Difference to Approximate Data
Relative

Exact data loaded (51s)

Exact data loaded (94s)

Exact data loaded (48s)

You are looking at the history and cannot make any changes.

Return to editing

Clear History

Reset App

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.*

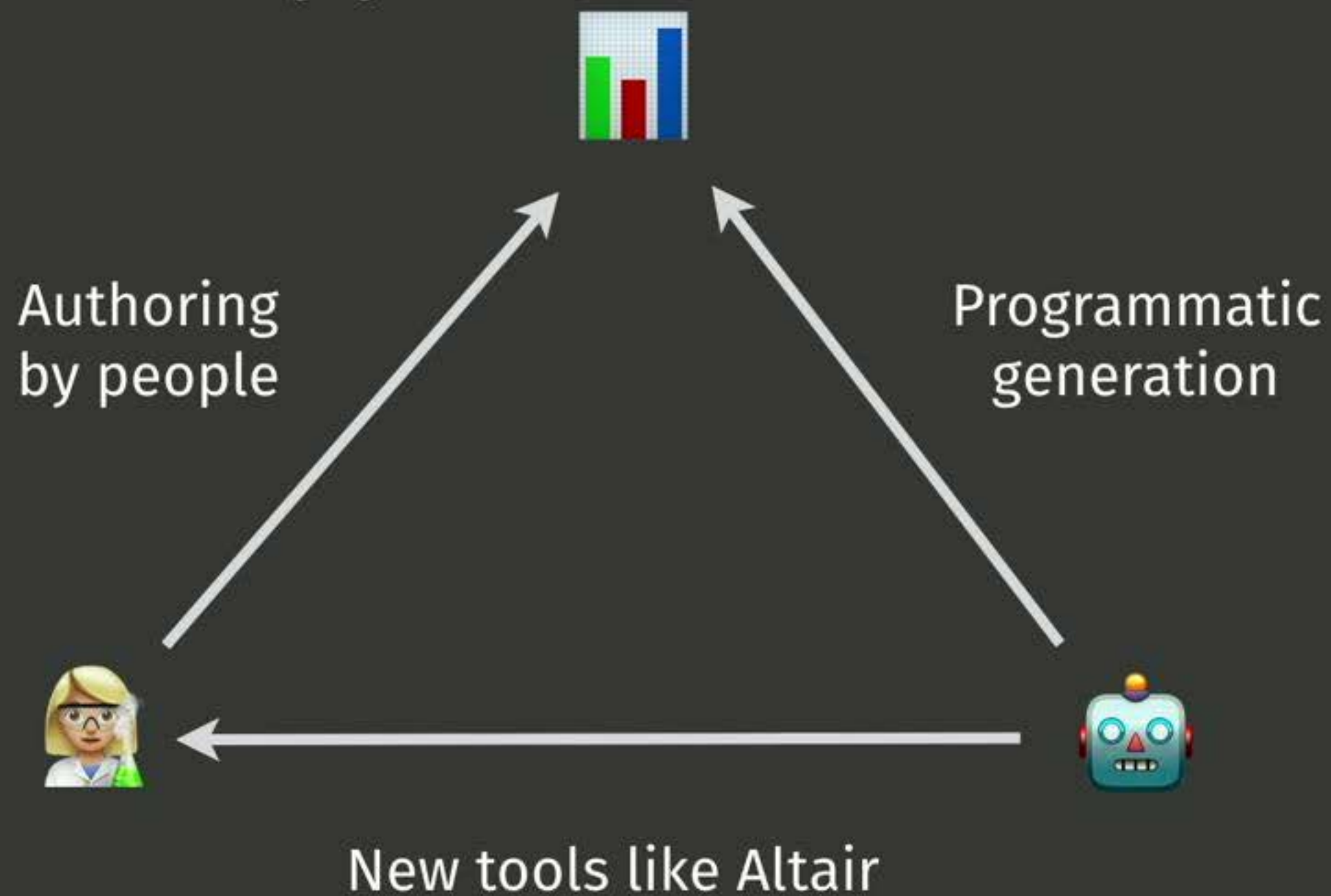
Fast and reliable approximations for
data exploration

My Mission:

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

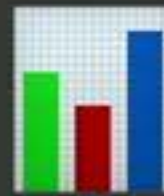


Vega-Lite
High-level visualization language





Draco
Model of visualization design



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.

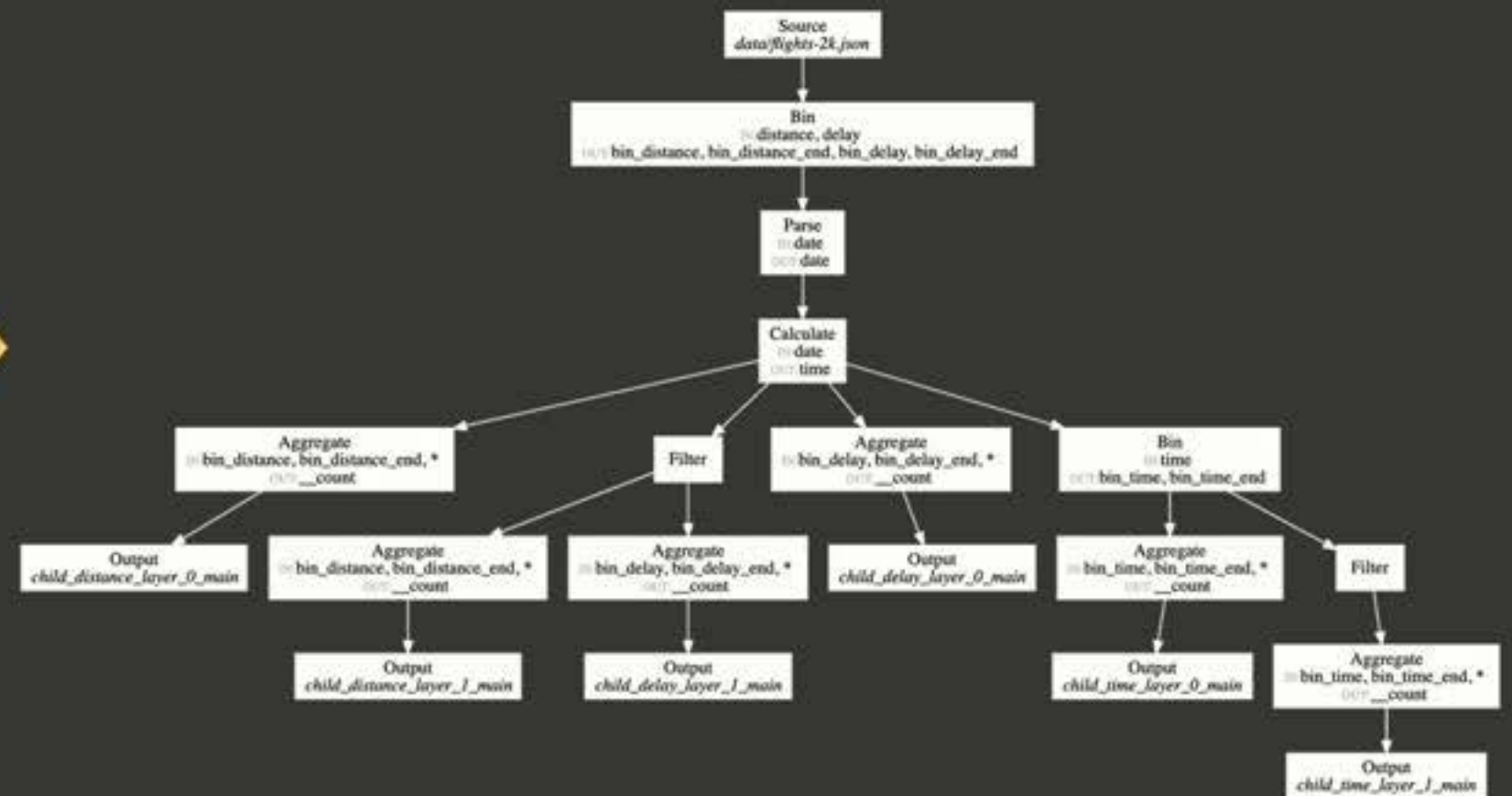
Challenge for the Future:

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

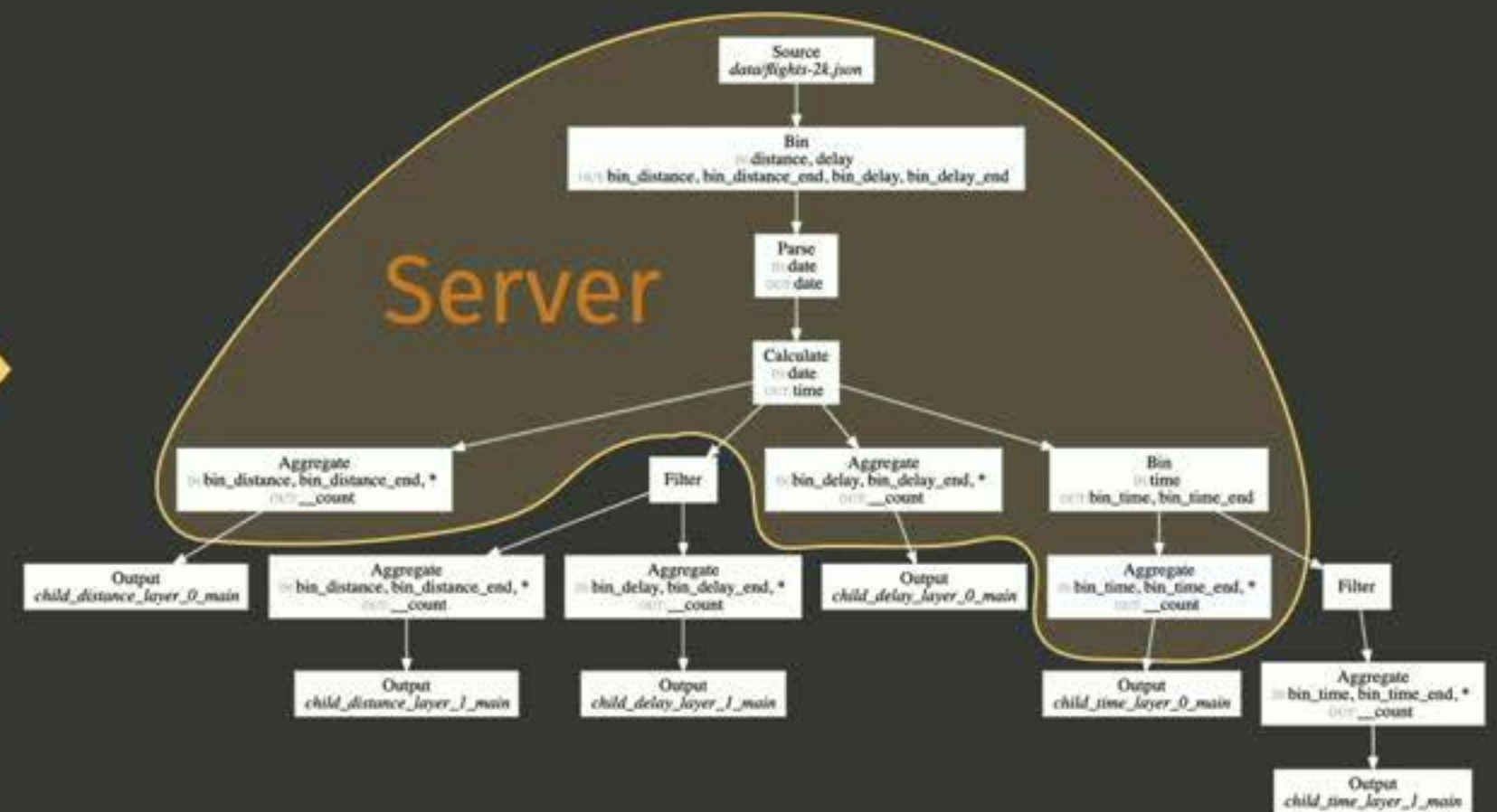
Reduce redundant computation.



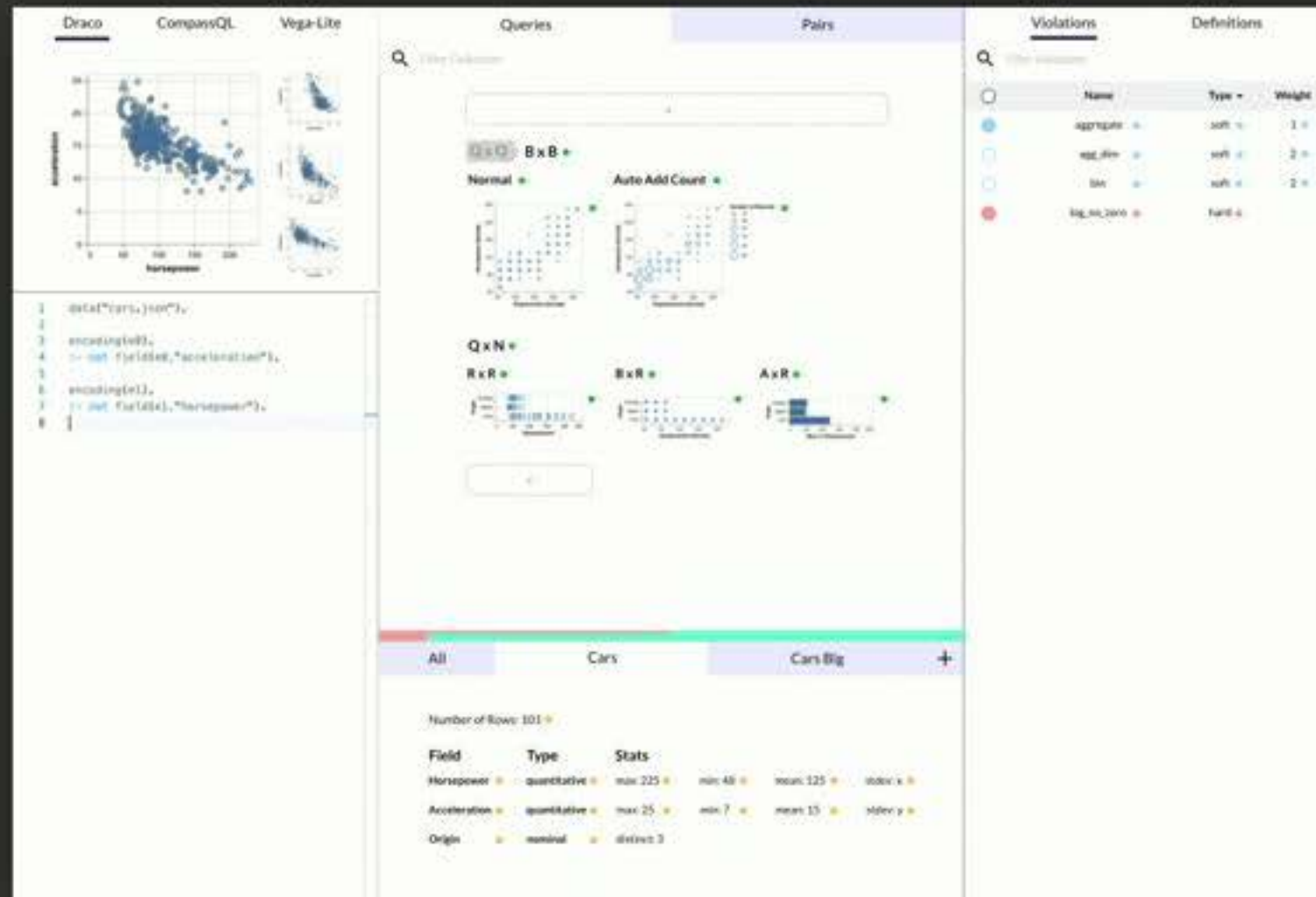
Vega-Lite's Research Frontier

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)

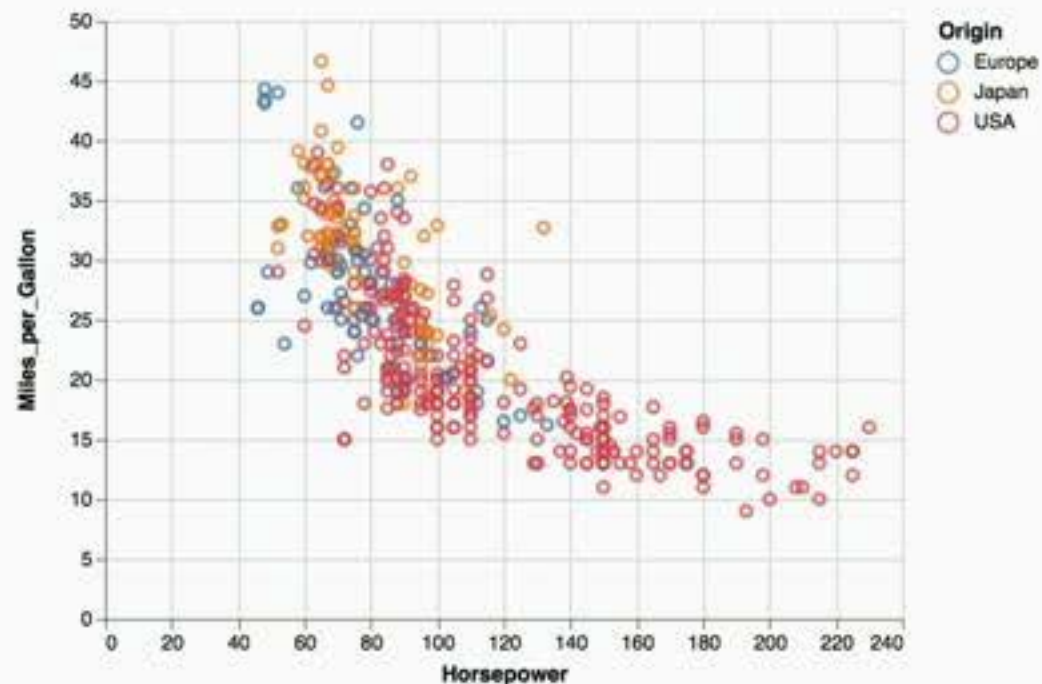
uwdata.github.io/draco-tuner

Draco's Research Frontier

```
import altair as alt

# load a simple dataset as a pandas DataFrame
from vega_datasets import data
cars = data.cars()

alt.Chart(cars).mark_point().encode(
    x='Horsepower',
    y='Miles_per_Gallon',
    color='Origin',
).interactive()
```



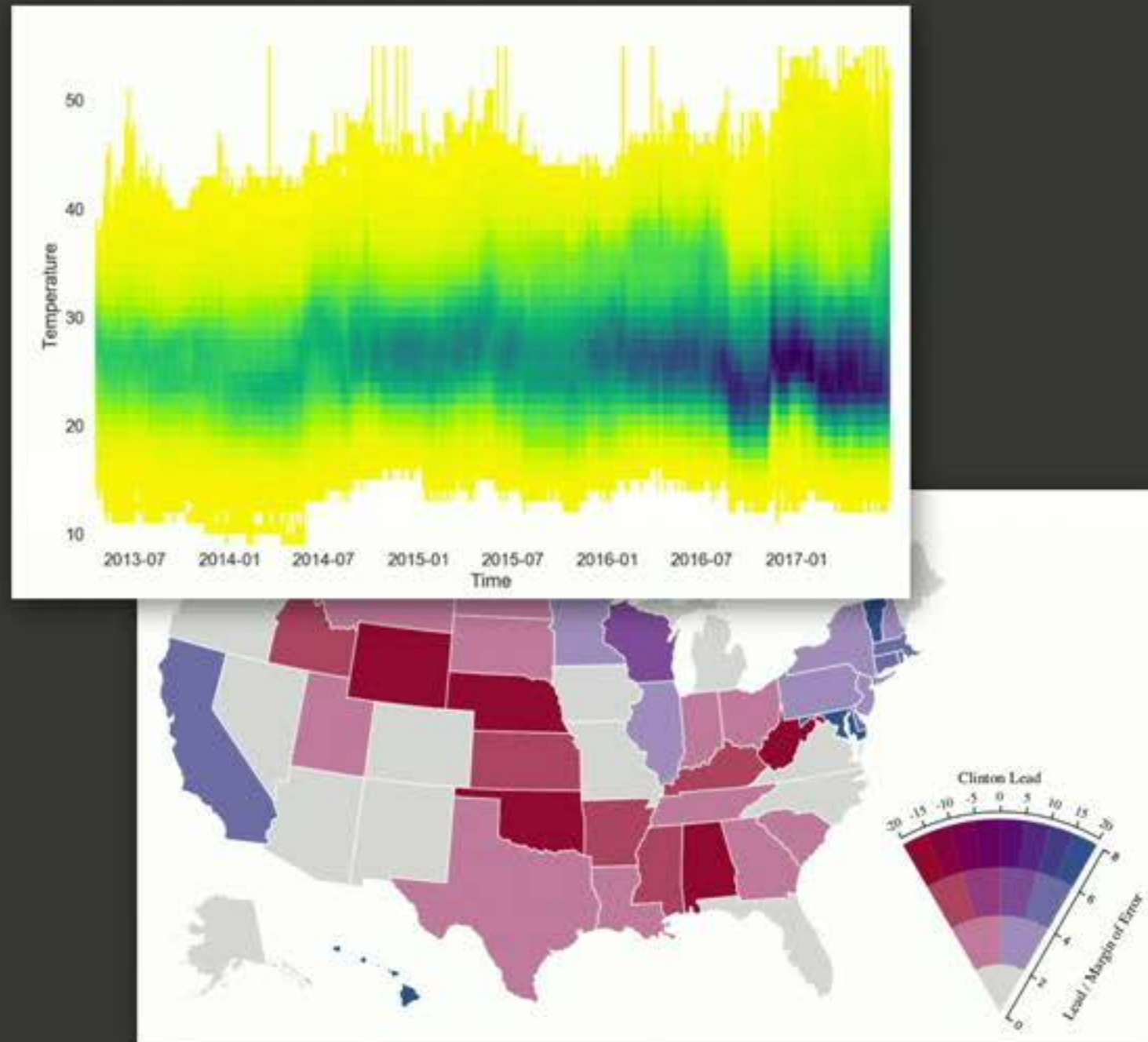
[Export as SVG](#) [Export as PNG](#) [View Source](#) [Open in Vega Editor](#)

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.

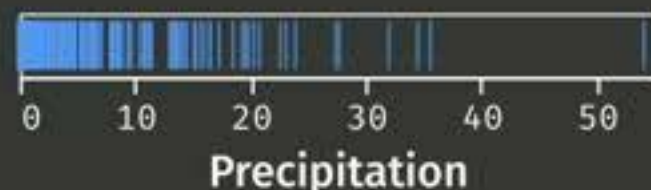
Integrate Draco into tools (e.g. Altair) to collect feedback.

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

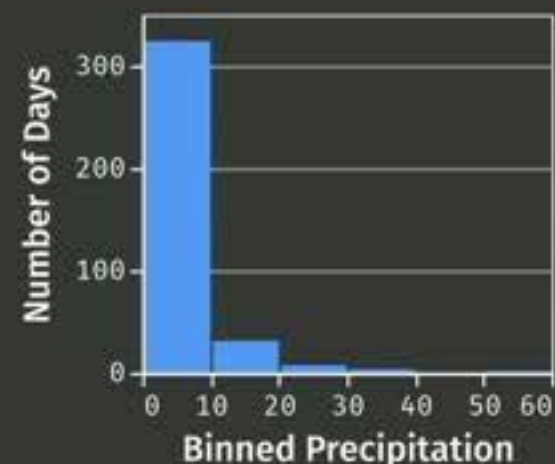
(UW, MIT, Northwestern)

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



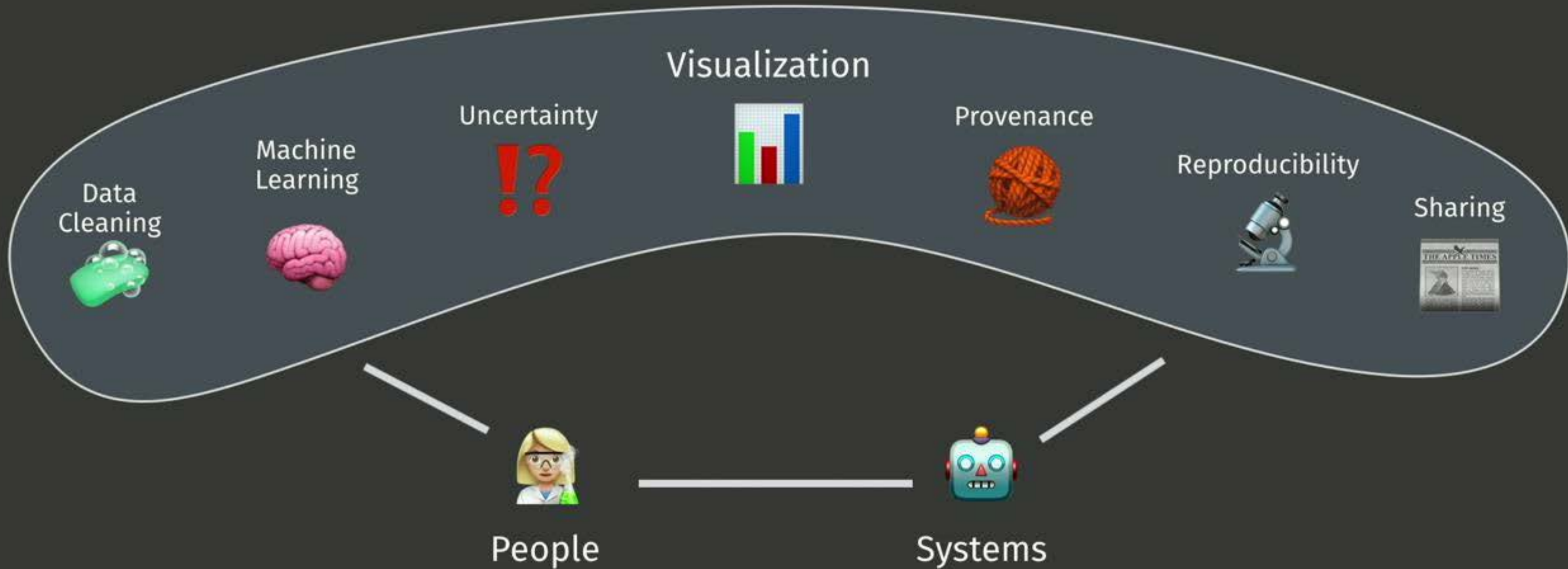
People



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