CS 8803-MDS

Human-in-the-loop Data Analytics

Lecture 23

11/14/22

Logistics

Office hour change

- 10-11AM this Friday

Evaluation plan due this Friday

Today's class

Investigating the Effect of the Multiple Comparisons Problem in Visual Analysis

- Archaeologist: Akshay

Vega-lite: A grammar of interactive graphics

- Authors: Yanhao, Yiheng
- Reviewer: Qiandong
- Archaeologist: Haotian
- Practioner: Aniruddha

Visualization design: the big picture

task

question & hypothesis

data

physical type

float, int, etc.

abstract type

nominal, ordinal etc.

domain

metadata, semantics

processing

algorithms

mapping

visual encoding

image

What: Data

Nominal (labels)

Fruits: Apples, oranges, …

Ordinal (rank-ordered, sorted)

Quality of meat: Grade A, AA, AAA

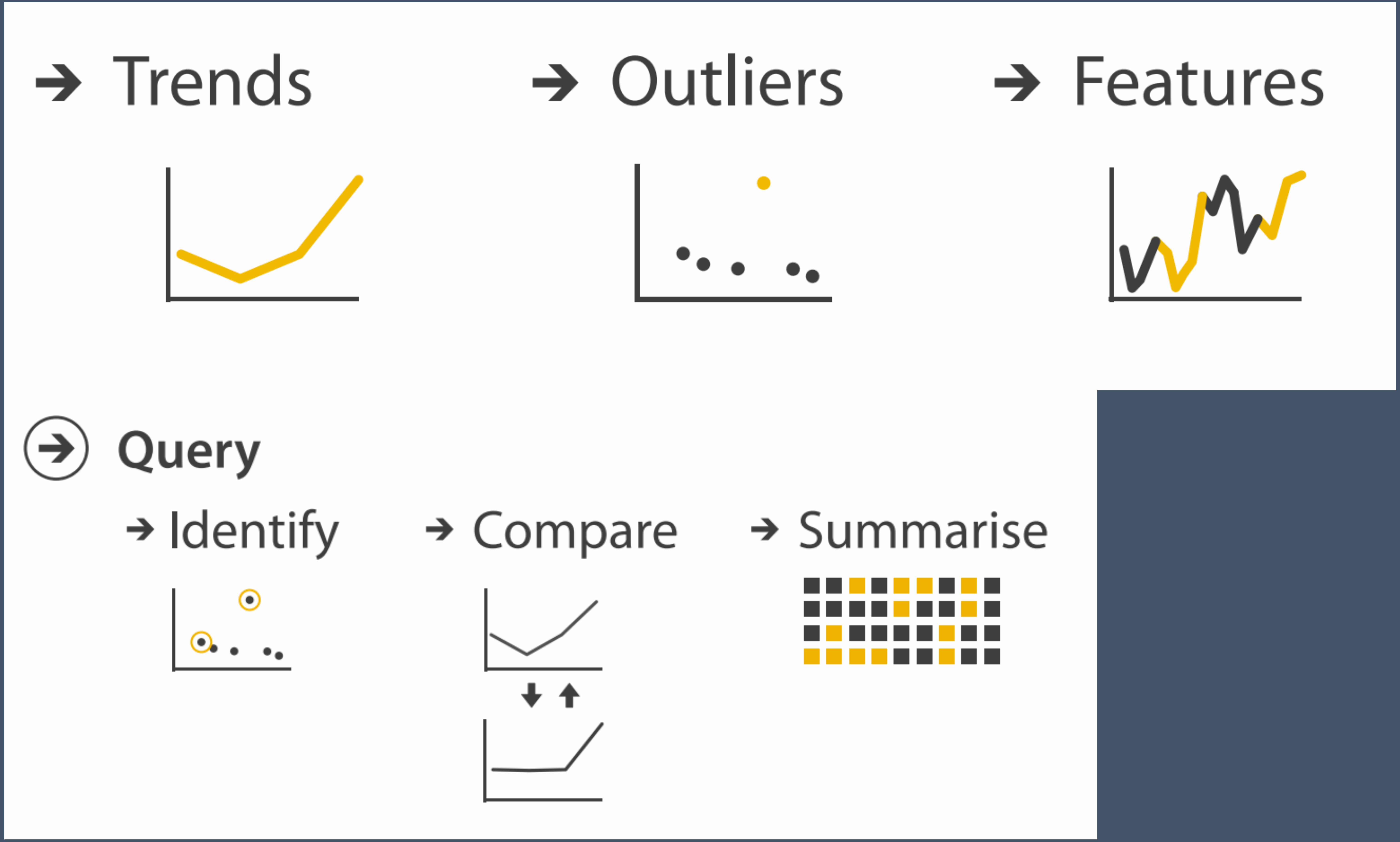
Interval (location of zero arbitrary)

Only differences (i.e. intervals) may be compared

Ratio (location of zero fixed)

Physical measurement: Length, Mass, Temp, … Counts and amounts

Why: Tasks


Trends
Outliers
Features
Query
Identify
Compare
Summarise

How: Visual Encodings

Position

Size

Value

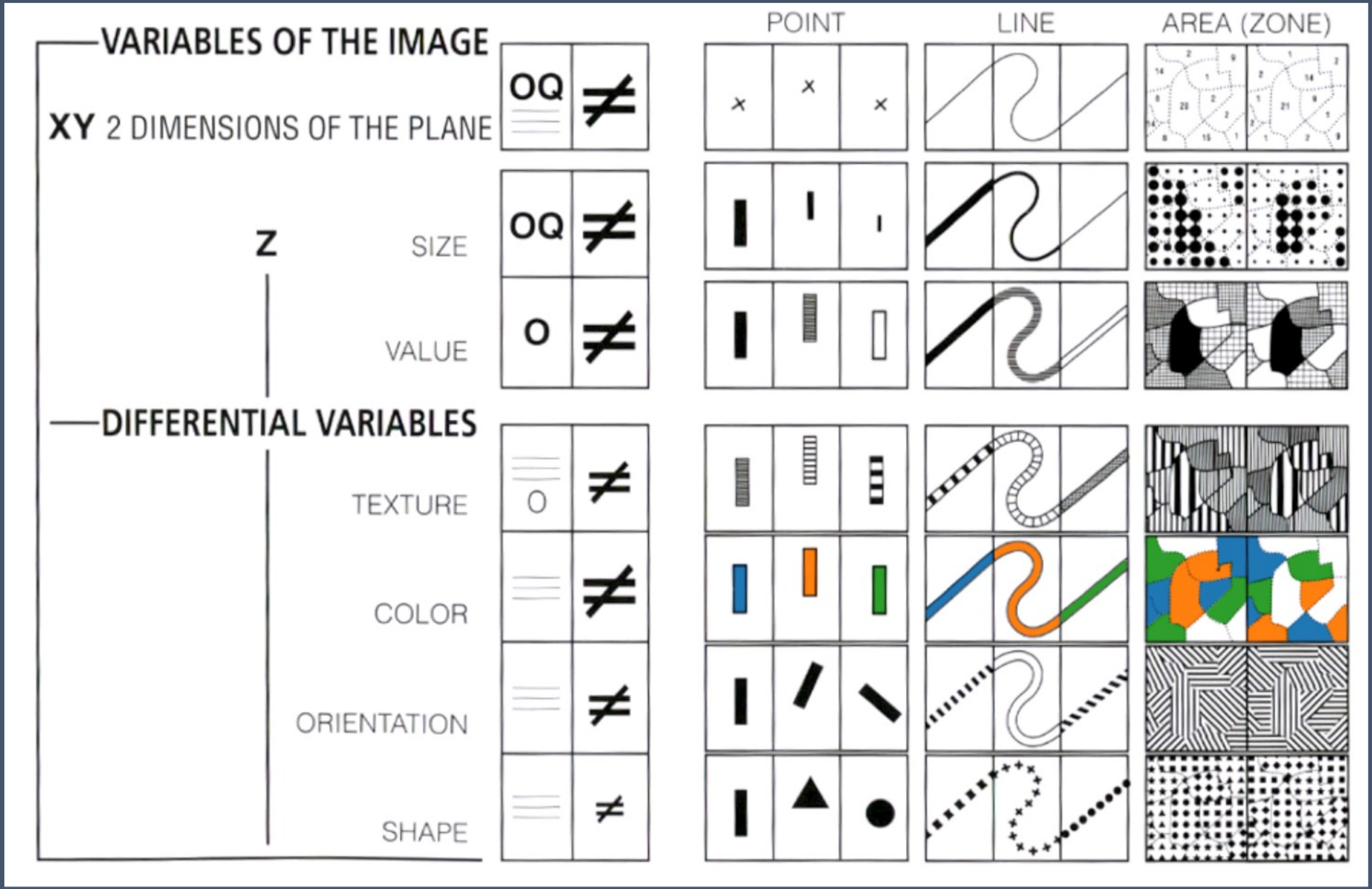
Texture

Color

Orientation

Shape

…



Choosing a visual encoding

Challenge

Assume 8 visual encodings and n data attributes. We would like to pick the "best" encoding among a combinatorial set of possibilities with size n^8

Principle of Consistency

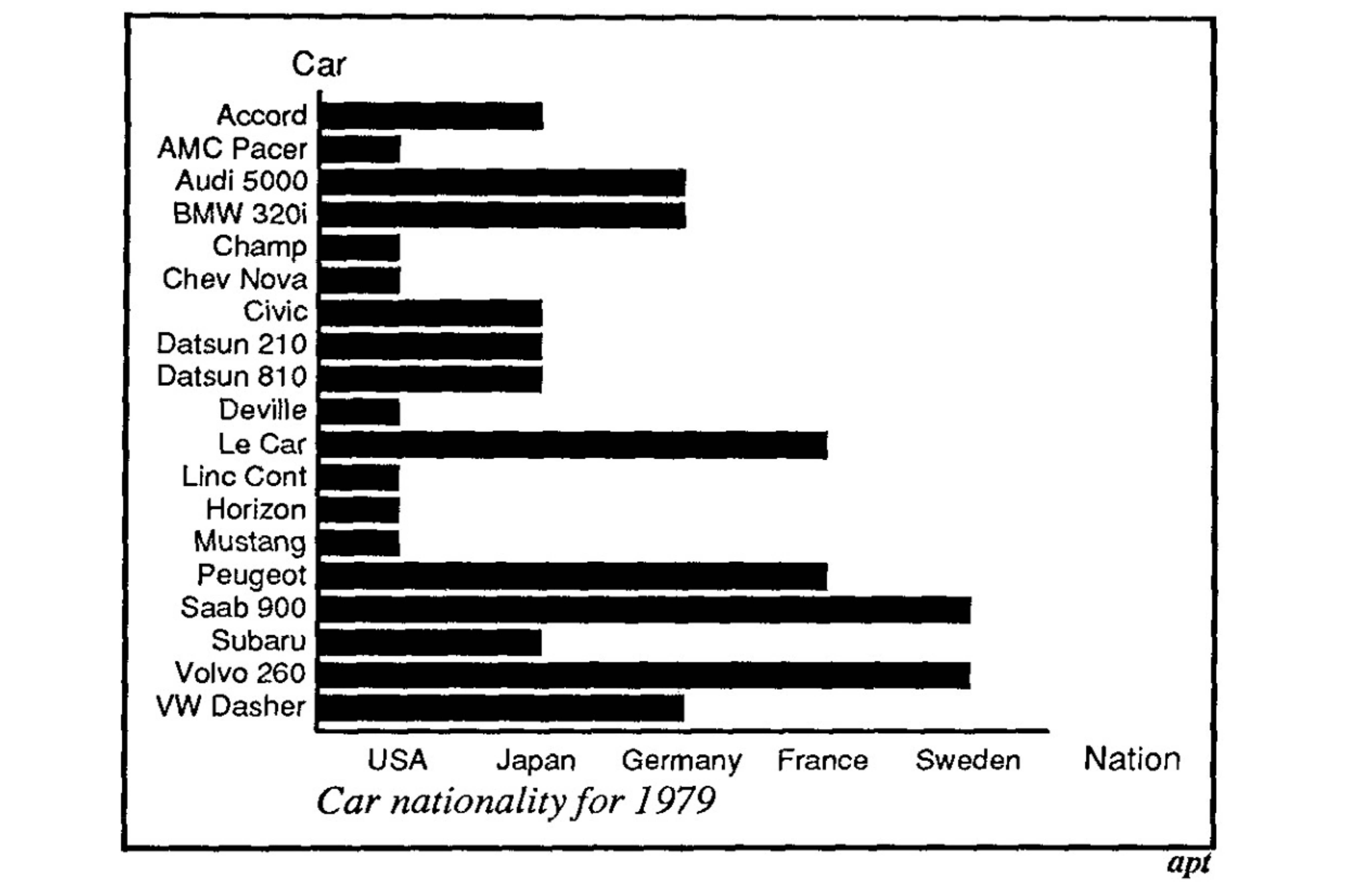
The properties of the image (visual variables) should match the properties of the data.

Principle of Importance Ordering

Encode the most important information in the most effective way.

Violation of consistency

Incorrect use of a bar chart. The lengths of bars are interpreted as a quantitative value.



Design Criteria (Machinlay, APT, 1986)

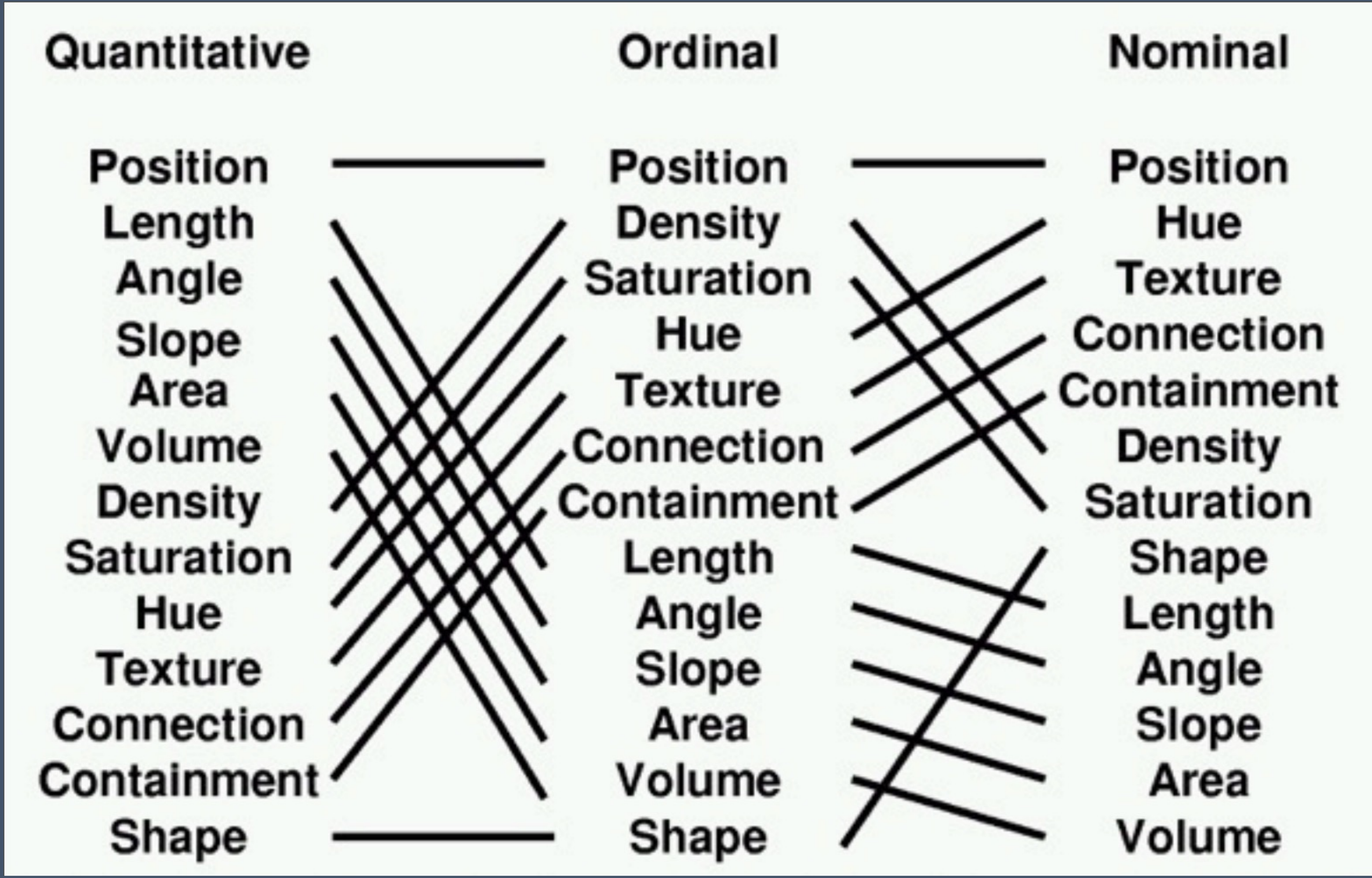
Effectiveness

A visualization is more effective than another visualization if the information conveyed by one visualization is more readily perceived than the information in the other visualization.

Expressiveness

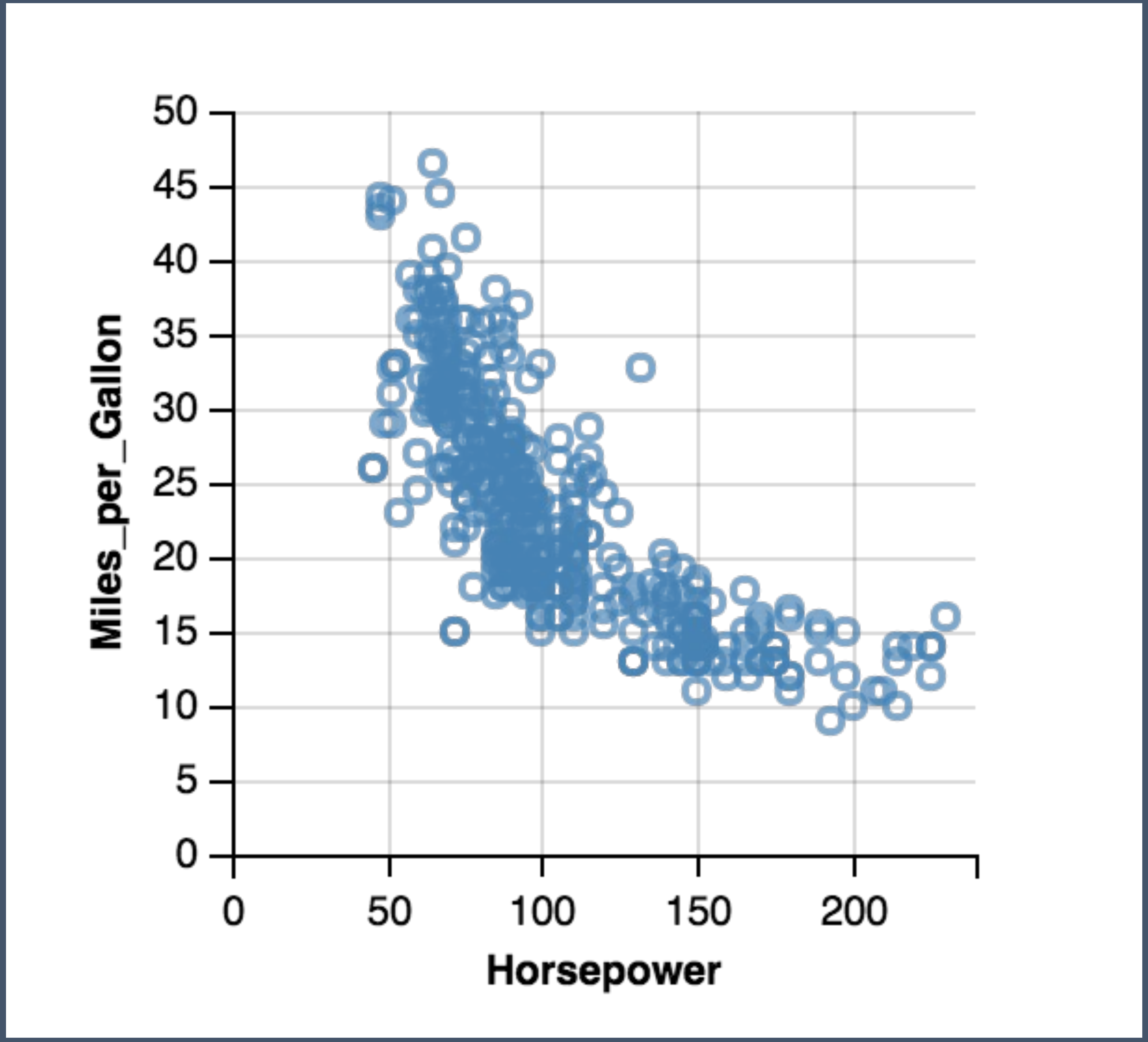
A set of facts is expressible in a visual language if the sentences (i.e. the visualizations) in the language express all the facts in the set of data, and only the facts in the data.

Mackinlay's ranking



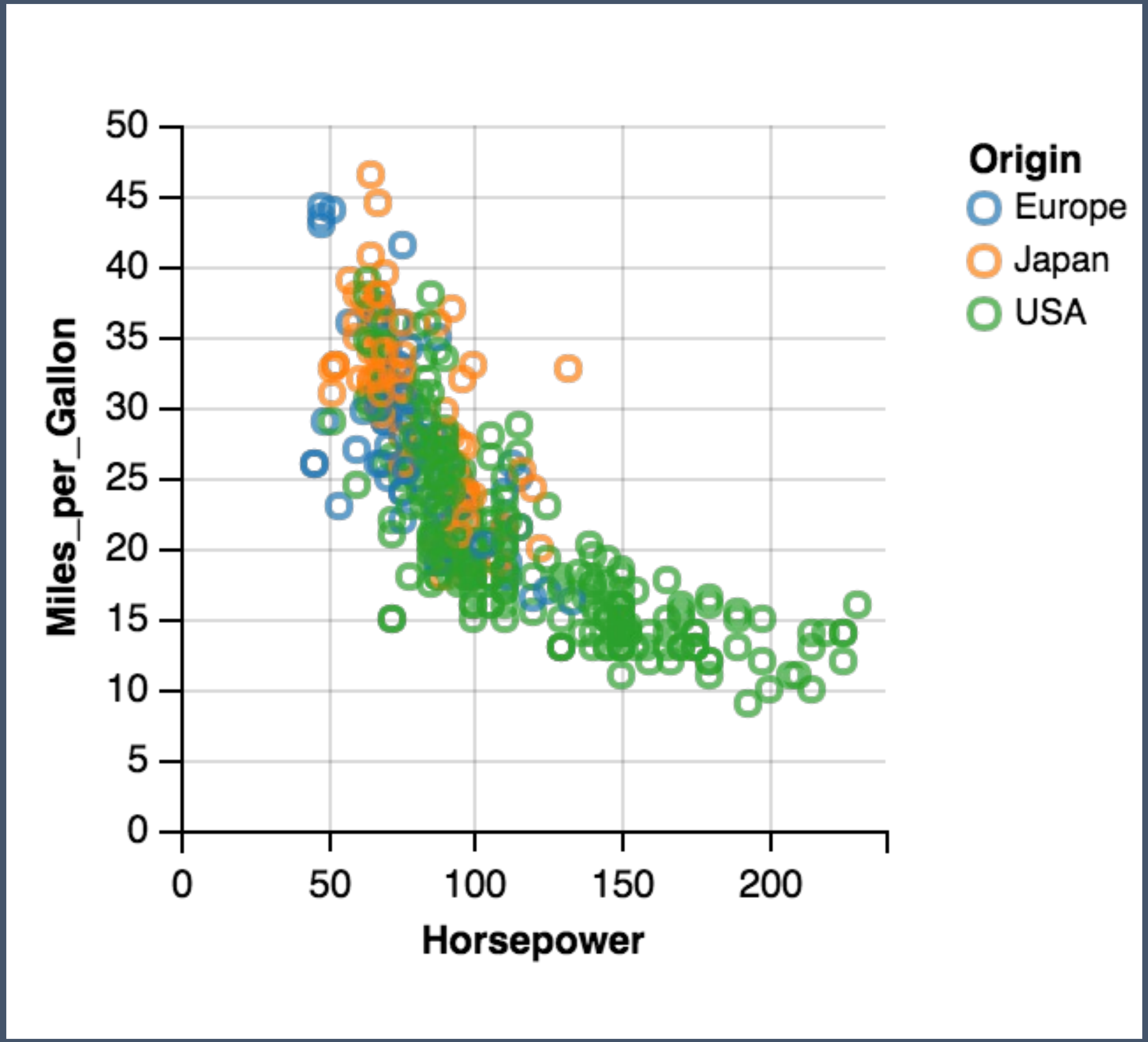
Conjectured *effectiveness* of the encoding

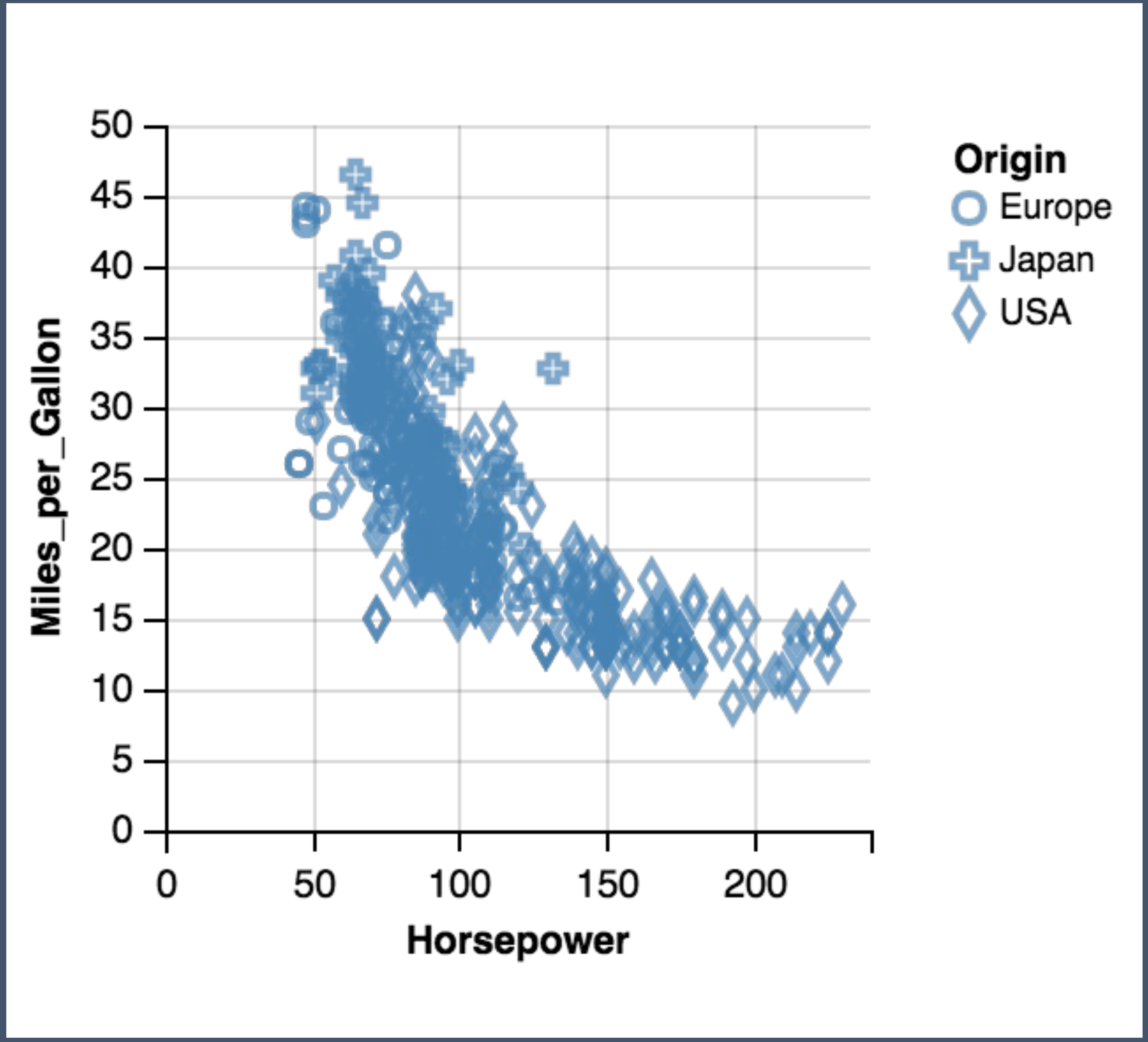
Which one is better?



Source: Vega-Lite Tutorial UC Davis

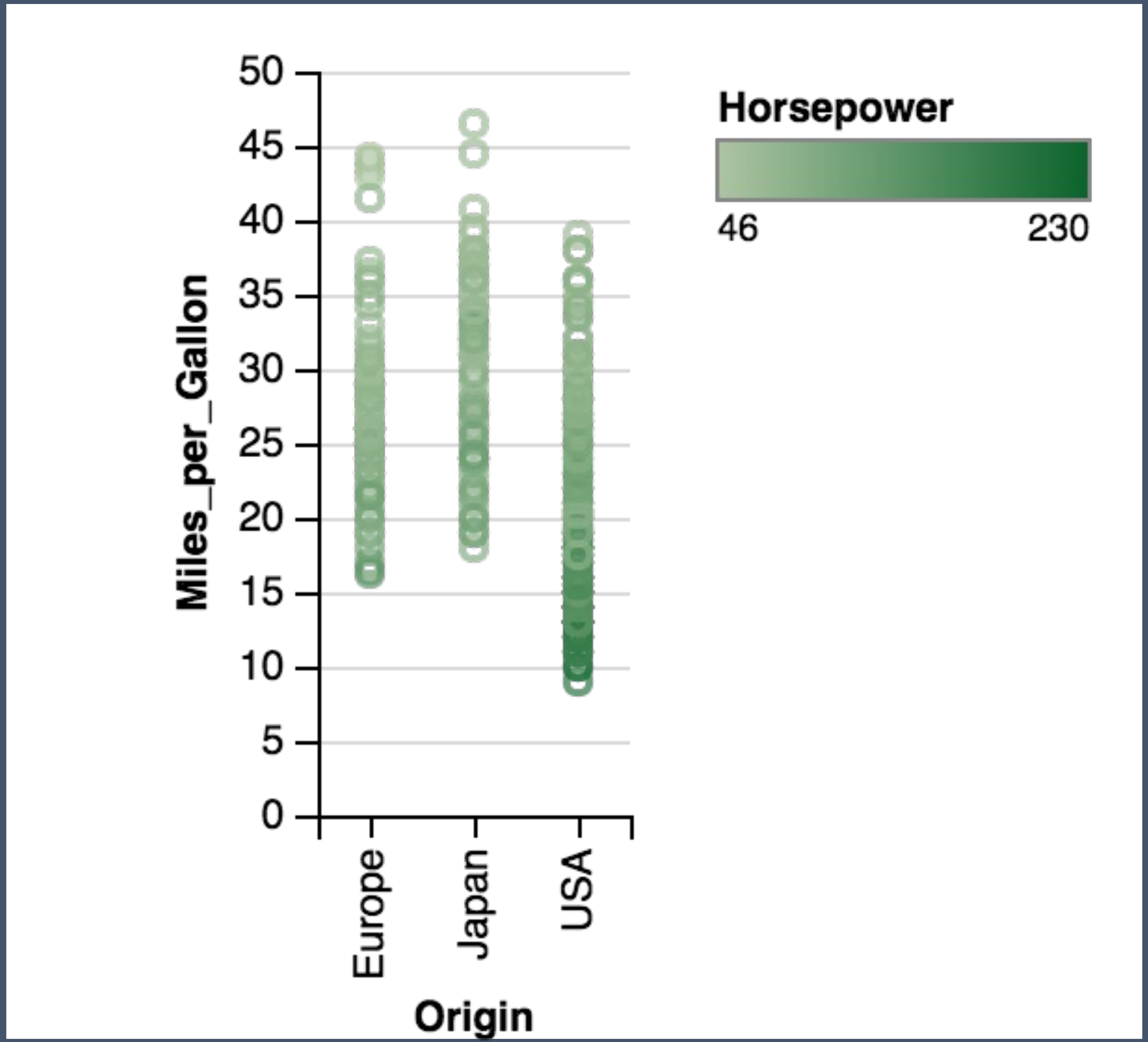
Which one is better?

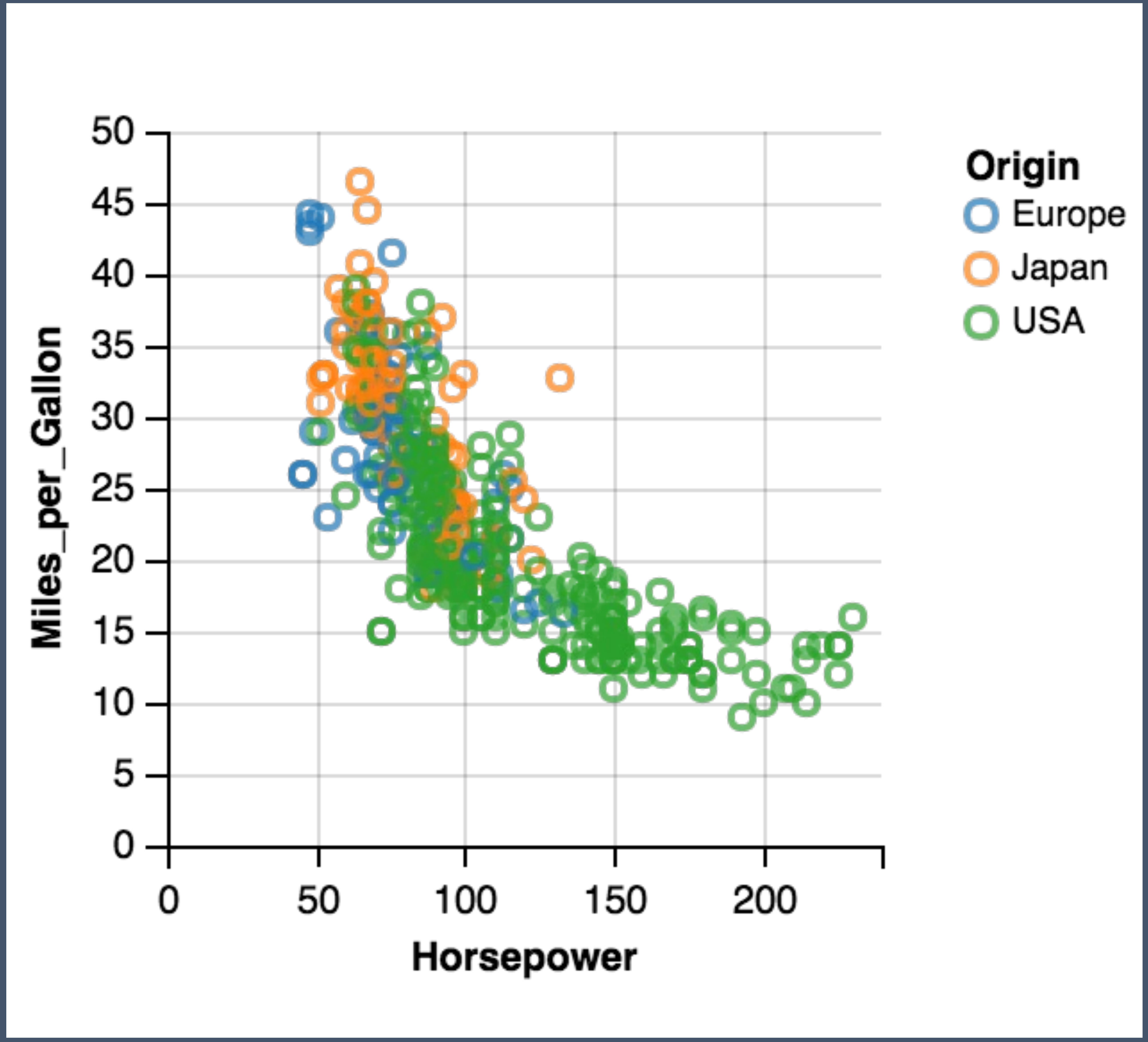




Source: Vega-Lite Tutorial UC Davis

Which one is better?





Source: Vega-Lite Tutorial UC Davis

APT: Automatic Chart Construction

- User formally specifies data model
- APT searches over design space
    - Tests expressiveness of each visual encoding
    - Generates image for encodings that pass test
    - Tests perceptual effectiveness of resulting image
- Outputs most effective visualization

Today's class

Investigating the Effect of the Multiple Comparisons Problem in Visual Analysis

- Archaeologist: Akshay

Vega-lite: A grammar of interactive graphics

- Authors: Yanhao, Yiheng
- Reviewer: Qiandong
- Archaeologist: Haotian
- Practioner: Aniruddha

Vega-Lite: A Grammar of Interactive Graphics

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

Presentor: Yanhao Wang, Yiheng Mao

Content

What's Vega-Lite?

Vega-Lite is a ***high-level grammar*** of interactive graphics. It provides a ***concise***, ***declarative JSON syntax*** to create ***an expressive range of visualizations*** for data analysis and presentation.

What’s Vega-Lite?

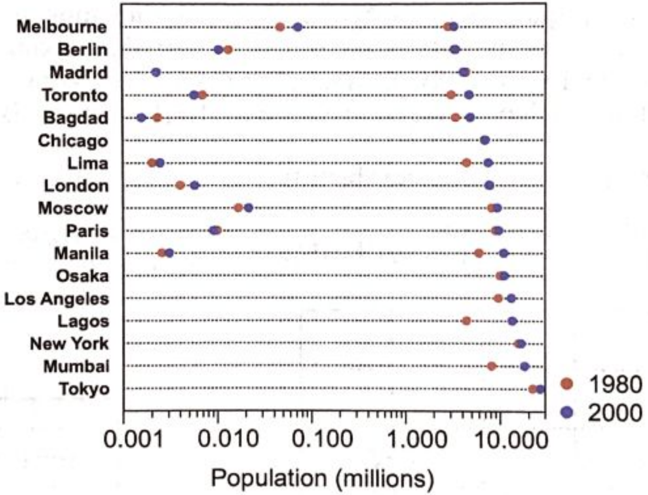
Vega-Lite is a *high-level* ***grammar of*** *interactive* ***graphics***. It provides a *concise, declarative JSON syntax* to create *an expressive range of visualizations* for data analysis and presentation.

Grammar of graphics

Statistical graphic specifications are expressed in six statements:

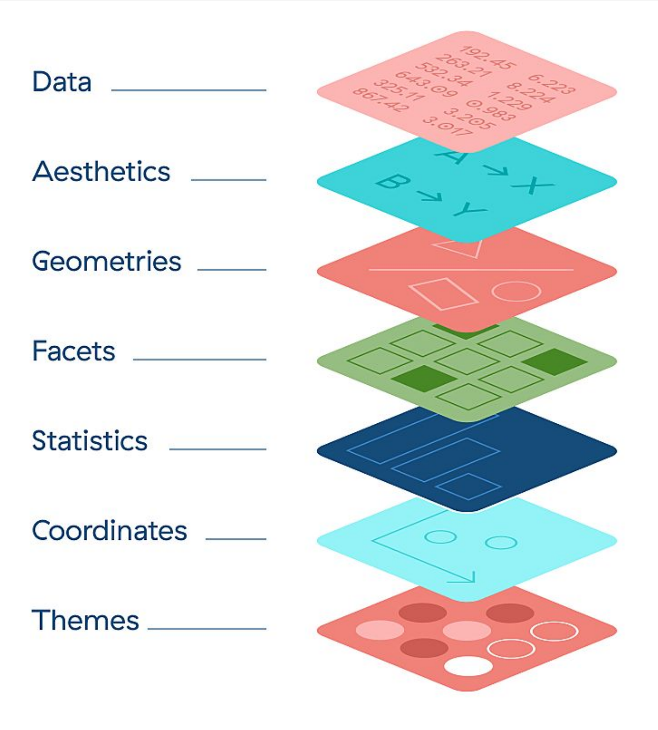
1) DATA: a set of data operations that create variables from datasets,
2) TRANS: variable transformations (*e.g., rank*),
3) SCALE: scale transformations (*e.g., log*),
4) COORD: a coordinate system (*e.g., polar*),
5) ELEMENT: graphs (*e.g., points*) and their aesthetic attributes (*e.g., color*),
6) GUIDE: one or more guides (*axes*, *legends*, etc.).

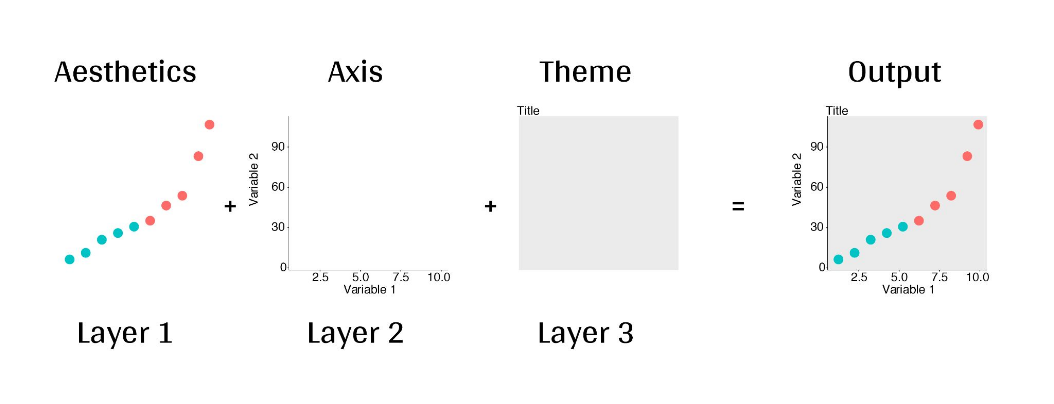
DATA: p1980 = "1980"
DATA: p2000 = "2000"
SCALE: *log*(*dim*(2), *base*(10))
COORD: *transpose*(*dim*(1, 2)))
ELEMENT: *point*(*position*(city*(pop1980+pop2000)), *color*(p1980 + p2000))



Leland Wilkinson. "The Grammar of Graphics"

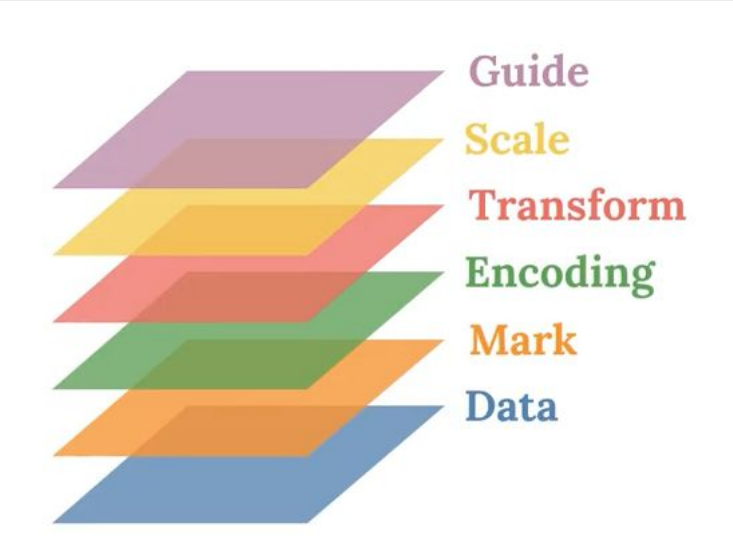
Grammar of graphics: ggplot





Wickham H. “ggplot2: Elegant Graphics for Data Analysis.”

Grammar of graphics: Vega-Lite



Axes & legends that visualize scales.

Functions that map data values to visual values.

Filter, aggregation, binning, etc.

Mapping between data and mark properties.

Data-representative graphics.

Input data source to visualize.

What's Vega-Lite?

Vega-Lite is a ***high-level*** *grammar of interactive graphics*. It provides a *concise, declarative JSON syntax* to create *an expressive range of visualizations* for data analysis and presentation.

Level of Abstraction

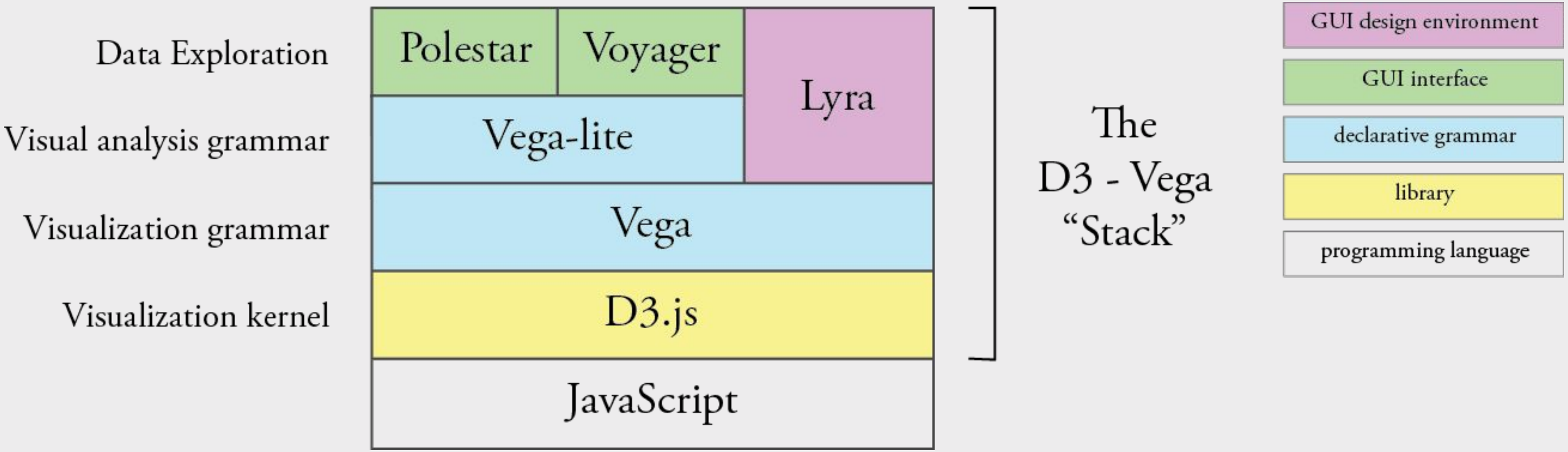
1. Graphics Libraries	2. Low-level Building Blocks	3. Visualization Grammars	4. High-level Building Blocks	5. Chart Templates

from scratch — Composable Building Blocks — ready-to-use

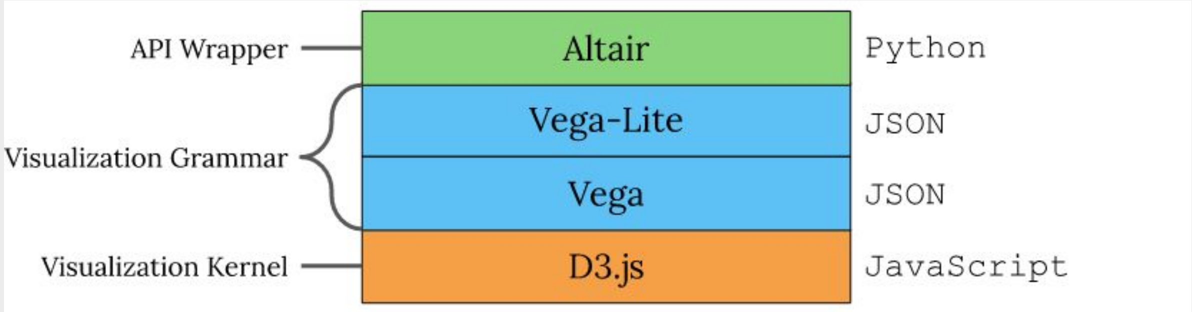
Expressive, most flexibility
Verbose specification
Fine-grained control
Explanatory data analysis

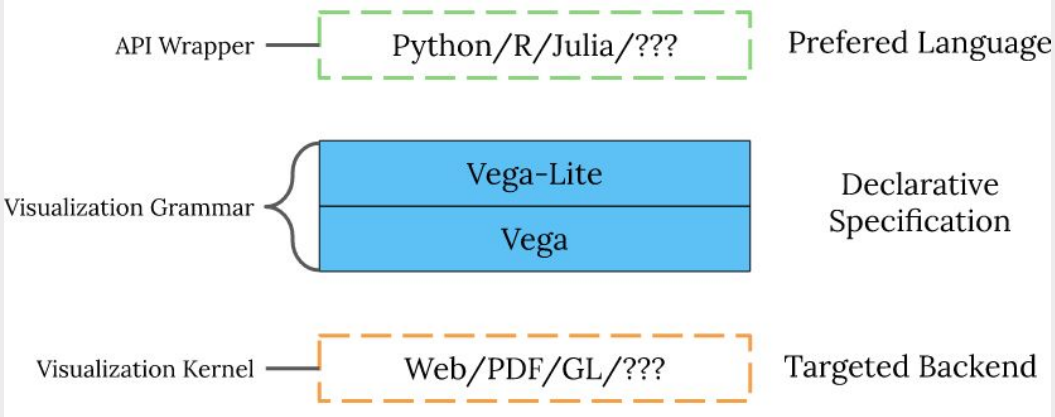
Concise, least effort
Limited expressiveness
Rapid iteration
Exploratory data analysis

Visualization Building Block Stack



Visualization Building Block Stack





What's Vega-Lite?

Vega-Lite is a *high-level grammar of interactive graphics*. It provides a *concise, declarative* ***JSON syntax*** to create *an expressive range of visualizations* for data analysis and presentation.

DESIGN SPACE OF DATA VISUALIZATION LIBRARIES
(ON THE WEB)

High-level
Less expressivity
~Less effort

Level of Abstraction

Low-level
More expressivity
~More effort

	API Design			
	Framework-specific	Plain JS	JSON + callbacks	JSON
Chart Templates	nivo vue-trend Recharts	Google Charts G2Plot dc.js	Chart.js Chartist.js	FusionCharts
High-level Building Blocks	Victory React-Vis Semiotic @deck.gl/react	dimple	ECharts HighCharts Plotly @deck.gl/core	@deck.gl/json
Visualization Grammars	Chart-Parts	G2 Muze		Vega-Lite Vega
Low-level Building Blocks	visx	D3 d3-annotation cola flubber labella		
Graphics Libraries	react-rough react-three-fiber	p5*js Rough.js three.js pixi.js		

What’s Vega-Lite?

Vega-Lite is a *high-level grammar of* ***interactive graphics***. It provides a *concise, declarative JSON syntax* to create *an expressive range of visualizations* for data analysis and presentation.

Support for *interactivity* is *limited* in existing high-level languages

Use a predefined set of common techniques

Linked selection, panning, zooming, etc.

Need to customize imperative event handling callbacks

Error-prone, require complex static analysis

Reactive Vega formulated declarative interaction primitives, but...

Remains to be a low-level abstraction

Verbose specification, impedes rapid authoring and hinders systematic exploration of alternative designs

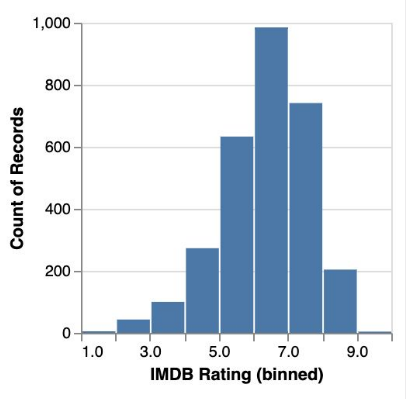
What’s Vega-Lite?

Vega-Lite is a *high-level grammar of interactive graphics*.

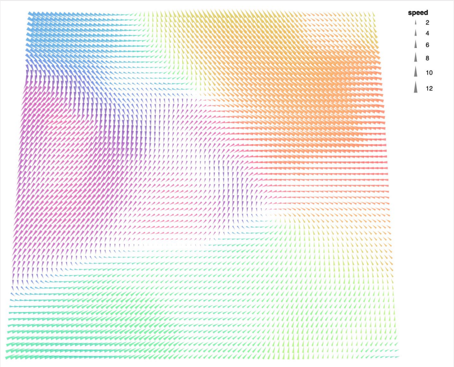
It provides a *concise, declarative JSON syntax* to create ***an expressive range of visualizations*** for data analysis and presentation.

Single-view plots

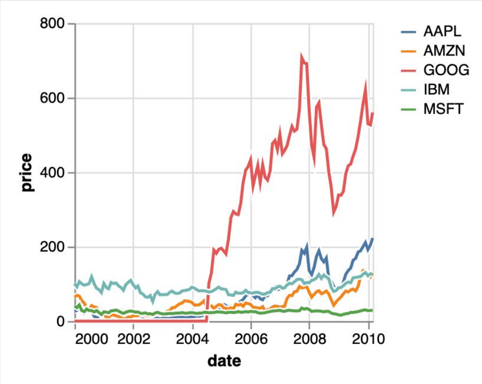
Histogram



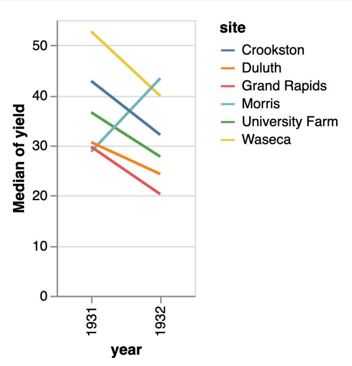
Wind Vector Map



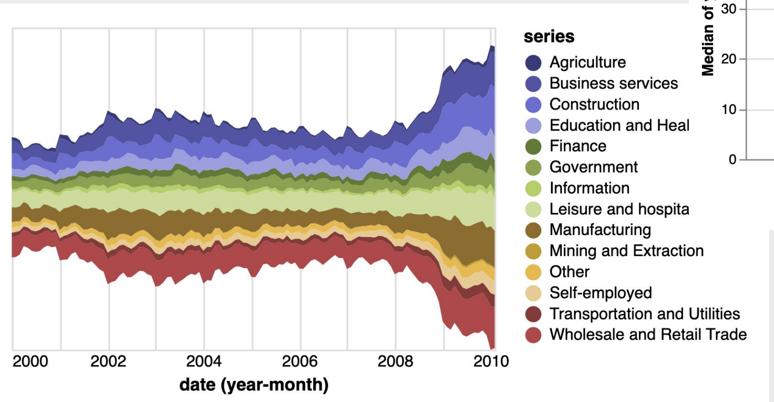
Multi-series Line Chart



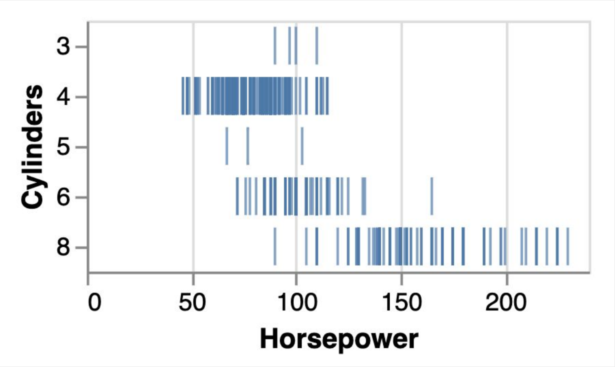
Slope Graph



Area Chart



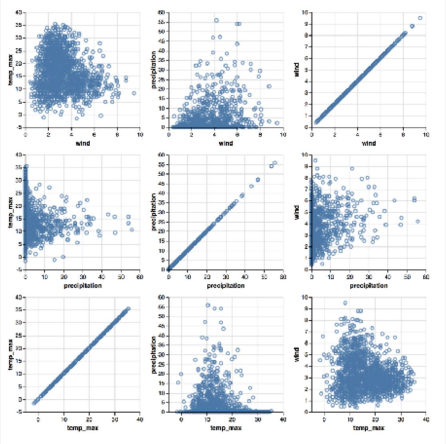
Stripplot



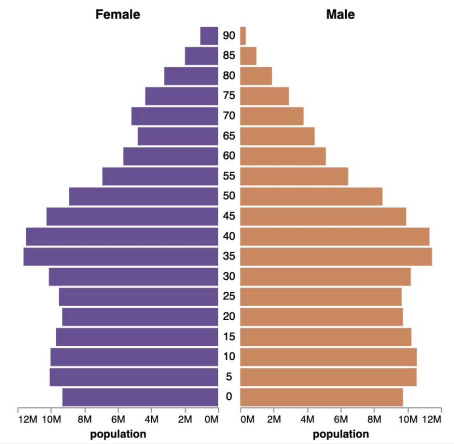
Multi-view plots & Layered plots

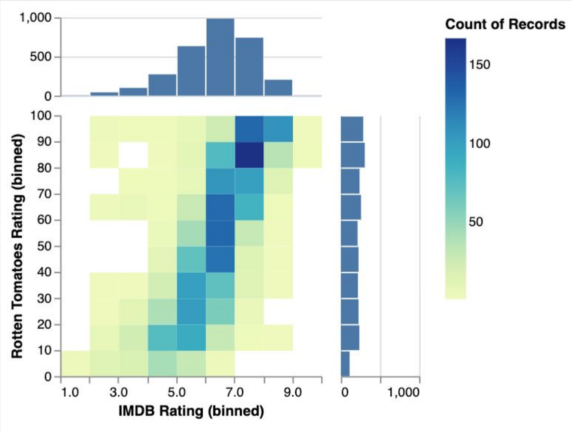
Layered View (Candlestick)

Scatterplot Matrix

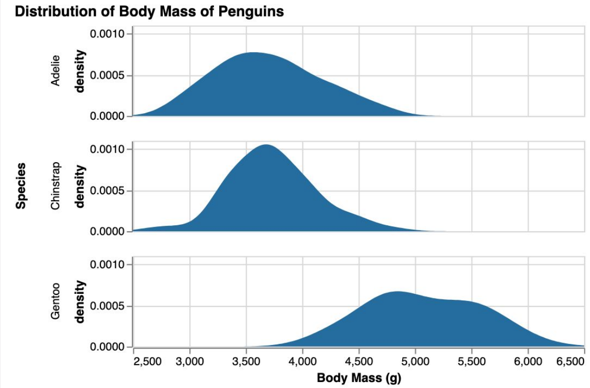


Concatenated View



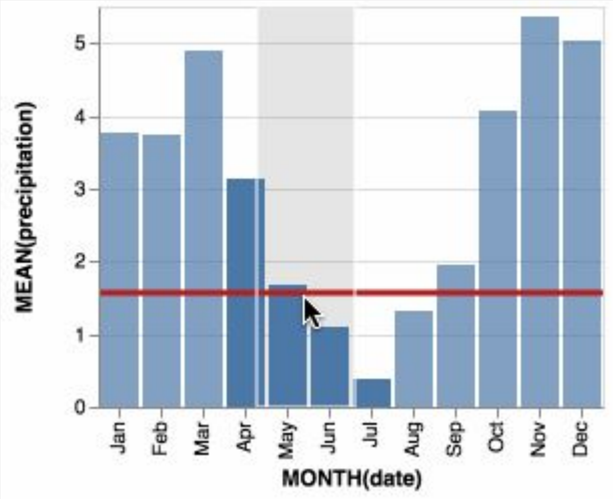


Faceted View

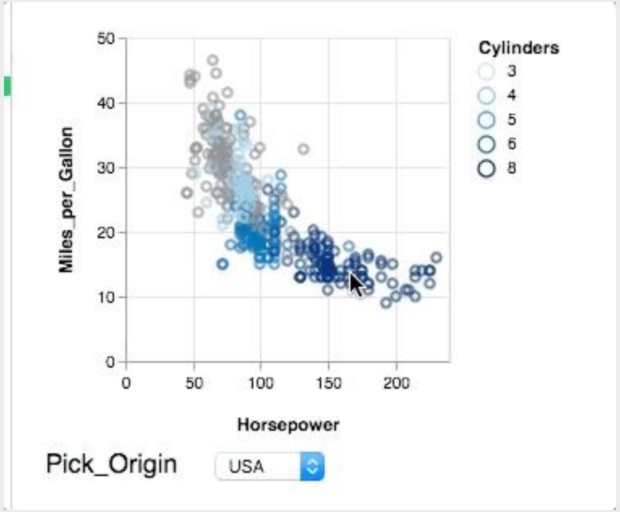


Interactive plots

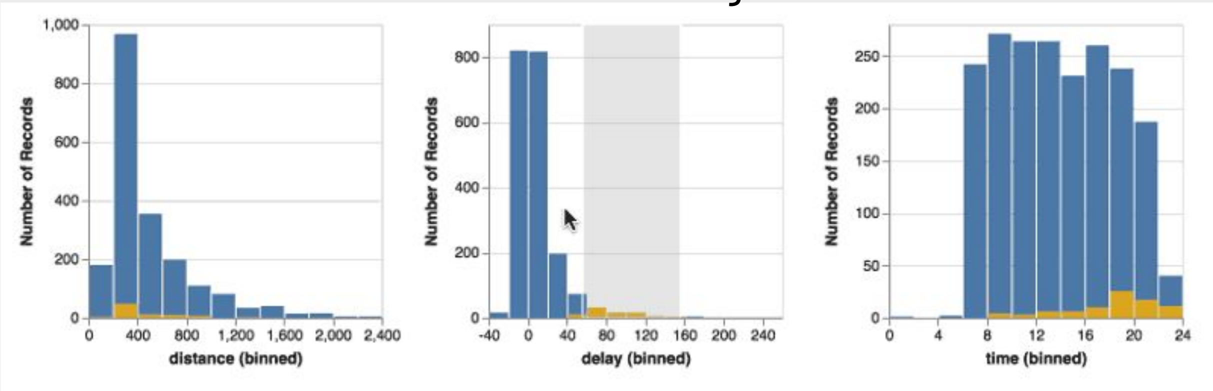
Brush



Focus & Context



Cross-Filtering



Why Vega-Lite?

High-level visualization grammar like Vega-lite can serve as an *intermediate representation* for...

Search & Inference

Enables systematic enumeration of data transforms

Recommendation

Enables filtering and ranking visualizations

Visualization generation

A defined search space for potential visualizations; textual, semantic representation

Content

01.

Intro and Background

- What's Vega-lite
- Why Vega-lite

02.

Vega-Lite Grammar Design

- Single View Specification
- Multi-view Composition
- Interactions

03.

Vega-Lite Compiler

- Architecture

04.

Example Visualizations

- Seven categories of techniques

05.

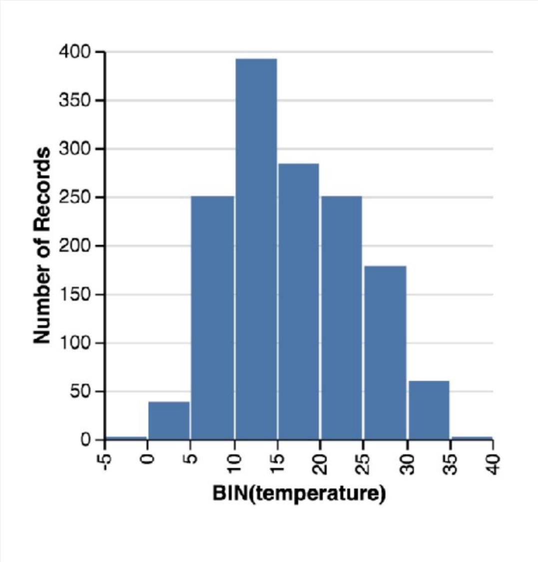
Discussions & Conclusion

- Limitations
- Future work

Single view specification

date	temp.	pp.	weather
1/1	10.6	10.9	"rain"
1/2	11.7	0.8	"drizzle"
1/3	12.2	10.2	"rain"
...	...	...	...

?

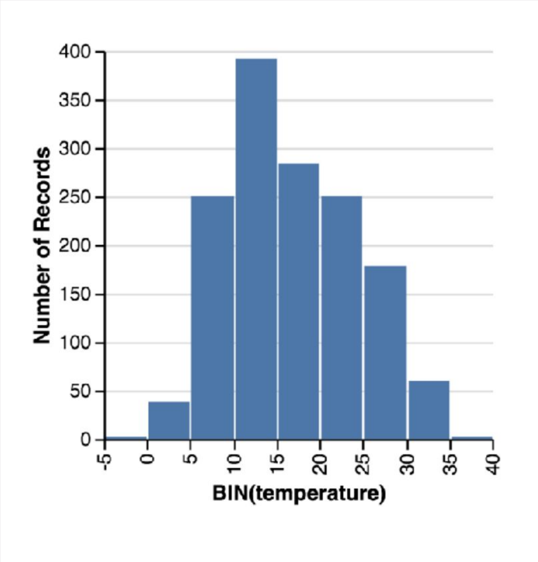


Bar chart, x=binned temp., y=count

Single view specification

date	temp.	pp.	weather
1/1	10.6	10.9	“rain”
1/2	11.7	0.8	“drizzle”
1/3	12.2	10.2	“rain”
...	...	...	...

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



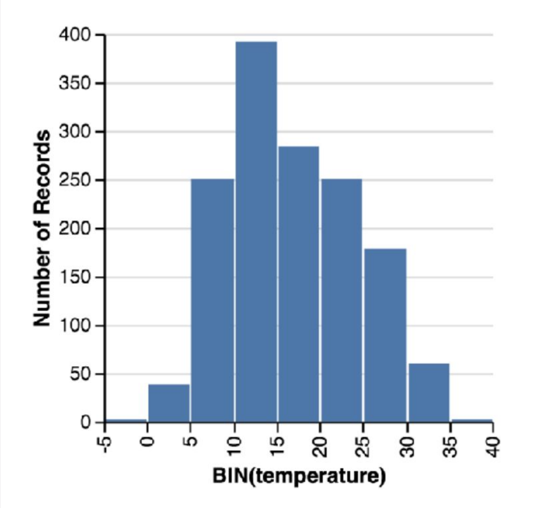
Bar chart, x=binned temp., y=count

Single view specification

unit := (data, transforms, mark-type, encodings)

date	temp.	pp.	weather
1/1	10.6	10.9	“rain”
1/2	11.7	0.8	“drizzle”
1/3	12.2	10.2	“rain”
...	...	...	...

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



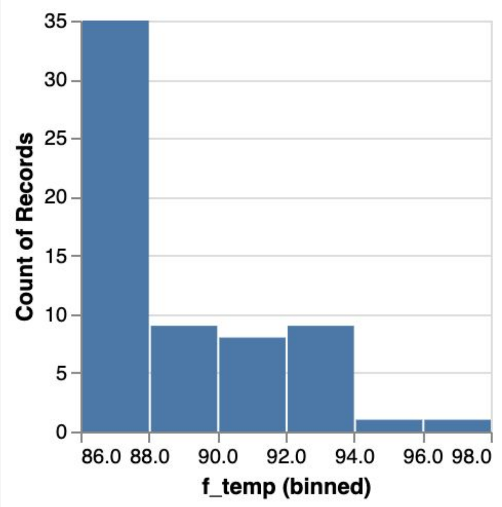
Single view specification

unit := (data, transforms, mark-type, encodings)

Transforms

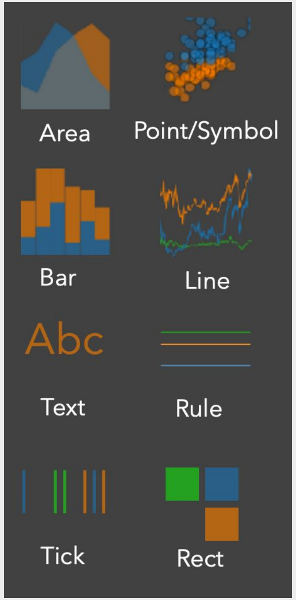
- Aggregate
- Bin
- Calculate
- Filter
- ...

```
{
  data: {url: "weather-seattle.json"},
  "transform": [
    {"calculate": "datum.temp*1.8+32", "as": "f_temp"},
    {"filter": "datum.f_temp >= 86"}
  ],
  mark: "bar",
  encoding: {
    x: {
       bin: true,
       field: "f_temp",
       type: "quantitative”
    },
    y: {
      aggregate: "count",
      type: "quantitative"
    }
  }
}
```

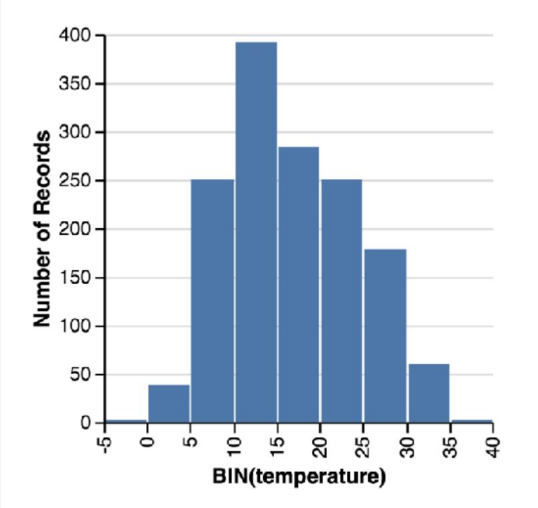


Single view specification

unit := (data, transforms, mark-type, encodings)

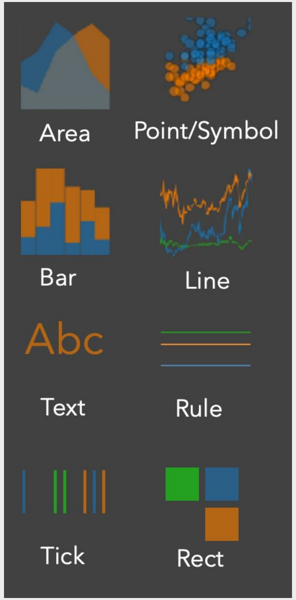



```
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {
      bin: true,
      field: "temperature",
      type: "quantitative”
    },
    y: {
     aggregate: "count",
     type: "quantitative"
    }
  }
}
```

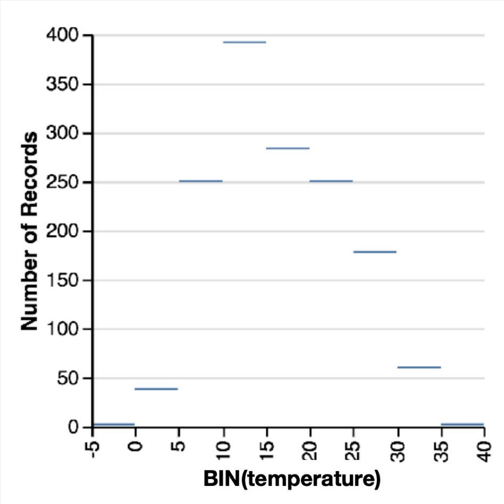


Single view specification

unit := (data, transforms, mark-type, encodings)




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

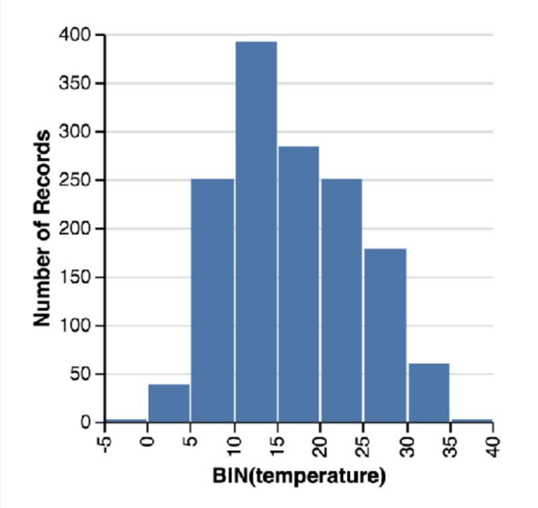


Single view specification

unit := (data, transforms, mark-type, encodings)

encoding := (channel, field, data-type, value, functions, scale, guide)

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



Single view specification

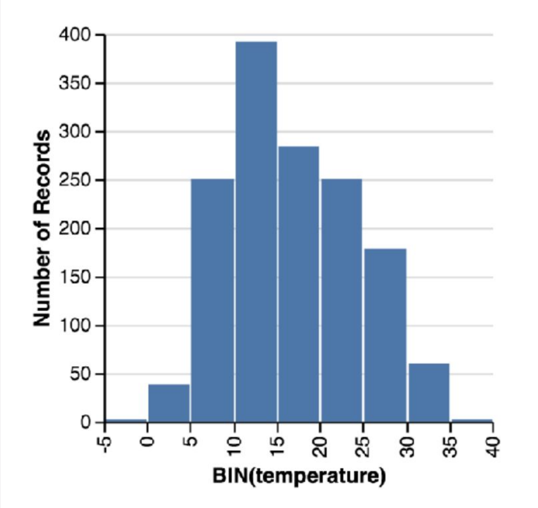
unit := (data, transforms, mark-type, encodings)

encoding := (channel, field, data-type, value, functions, scale, guide)

Channels

- X
- Y
- Color
- Shape
- Size
- Text
- Key
- Order
- Facet
- ...

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



Single view specification

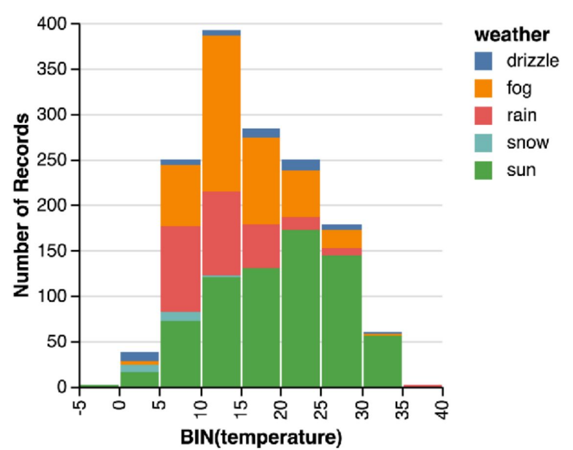
unit := (data, transforms, mark-type, encodings)

encoding := (channel, field, data-type, value, functions, scale, guide)

Channels

- X
- Y
- Color
- Shape
- Size
- Text
- Key
- Order
- Facet
- ...

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

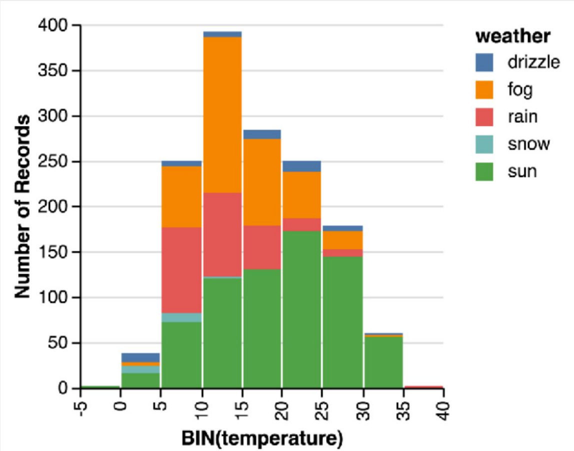


Single view specification

unit := (data, transforms, mark-type, encodings)

encoding := (channel, field, data-type, value, functions, scale, guide)

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



Single view specification

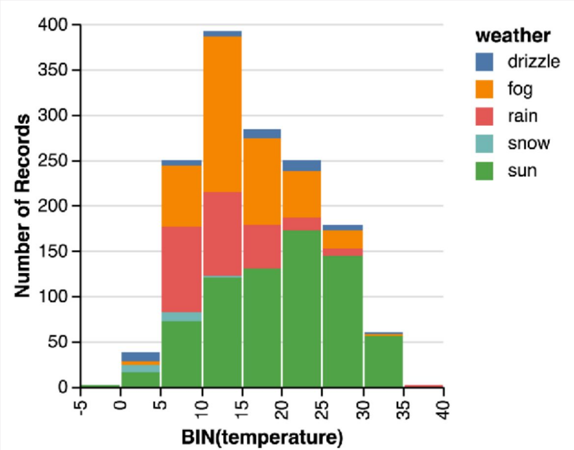
unit := (data, transforms, mark-type, encodings)

encoding := (channel, field, data-type, value, functions, scale, guide)

Data Types

- Quantitative
- Nominal
- Ordinal
- Temporal

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



Single view specification

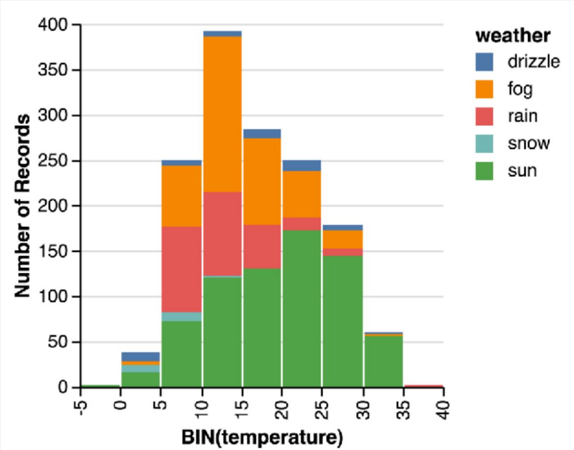
unit := (data, transforms, mark-type, encodings)

encoding := (channel, field, data-type, value, functions, scale, guide)

Functions

- Binning
- Aggregation
- Sorting
- ...

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



Single view specification

unit := (data, transforms, mark-type, encodings)

encoding := (channel, field, data-type, value, functions, scale, guide)

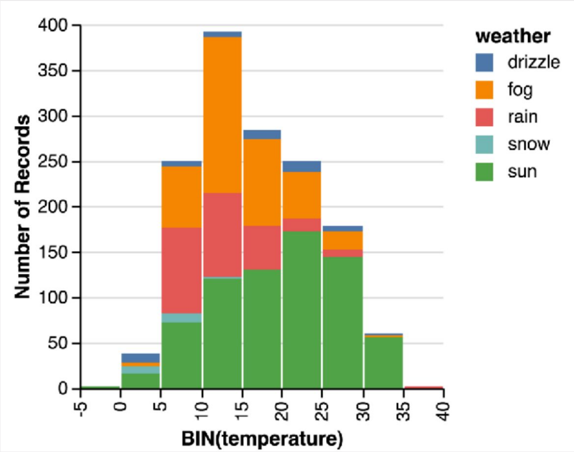
Scale:
f(data domain) -> Visual Range

Guide:
Visualize the scale (legend/axis)

Both with ***sensible default*** based on *channel* & *data-type*

- Palette (continuous/discrete)
- Axis (linear/ordinal)
- ...

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

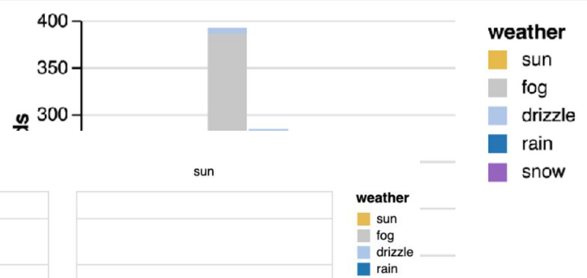


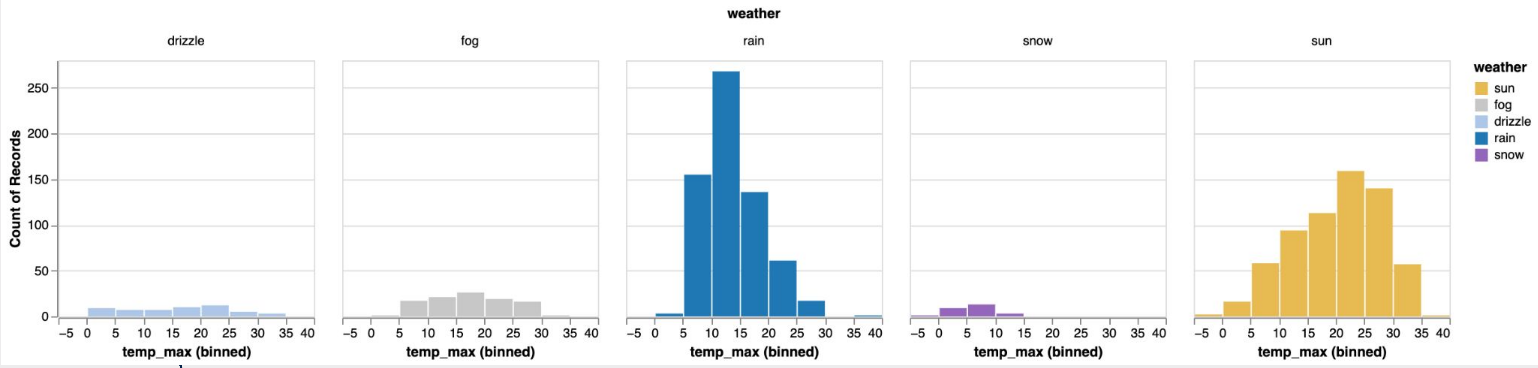
Single view specification

unit := (data, transforms, mark-type, encodings)

encoding := (channel, field, data-type, value, functions, scale, guide)

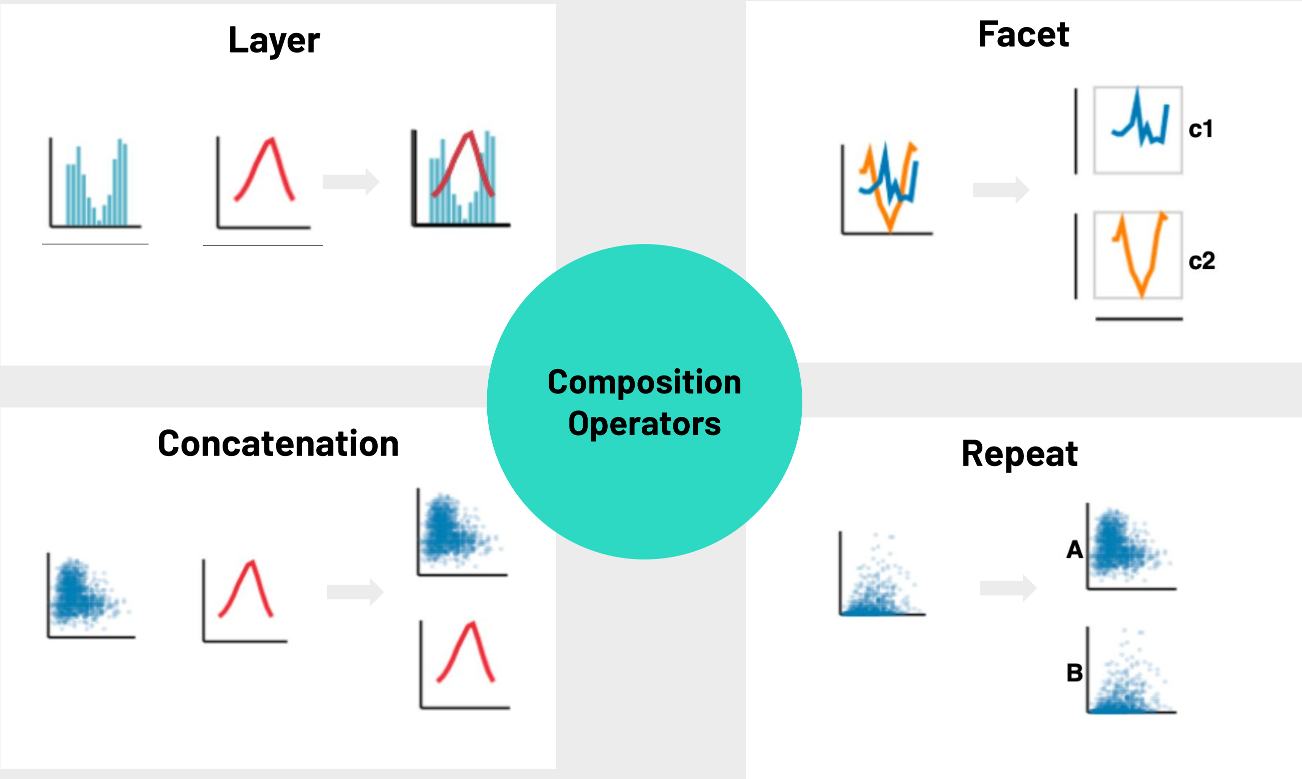
```
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {...},
```






```
  }
```

Layered & Multiview Specification



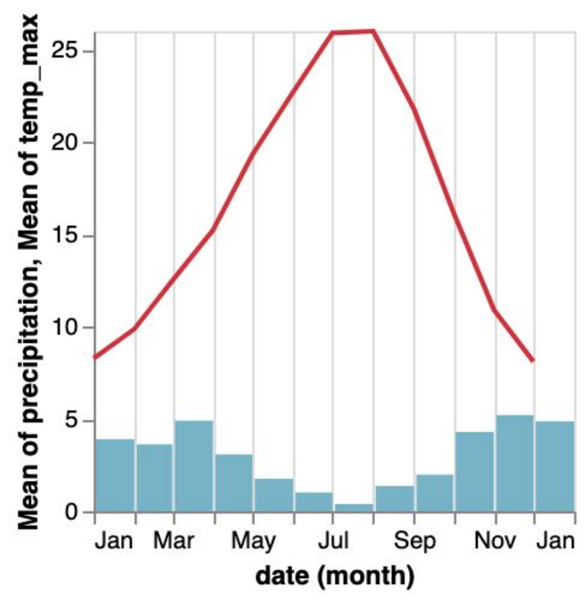
Layer

Composite views
cannot be layered

Default: **shared scales, merged guides**

layer ([unit1, unit2, ...], resolve)

```
{...
 "layer": [
   {
     "mark": "bar",
     "encoding": {
       "x": {"field": "date", "type": "temporal", "timeUnit": "month"},
       "y": {
         "field": "precipitation",
         "type": "quantitative",
         "aggregate": "mean",
         "axis": {"grid": false}
       },
       "color": {"value": "#77b2c7"}
     }
   },
   {
     "mark": "line",
     "encoding": {
       "x": {"field": "date", "type": "temporal", "timeUnit": "month"},
       "y": {
         "field": "temp_max",
         "type": "quantitative",
         "aggregate": "mean",
         "axis": {"grid": false}
       },
       "color": {"value": "#ce323c"}
     }
   }
 ]
}
```



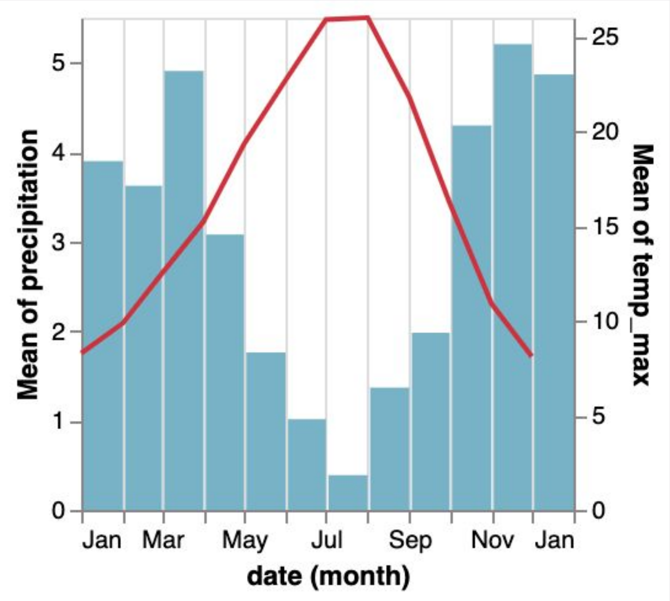
Layer

Composite views
cannot be layered

Default: **shared scales, merged guides**
Specify (channel, scale/guide, independent/union)
to override the default behavior

layer([unit1, unit2, ...], resolve)

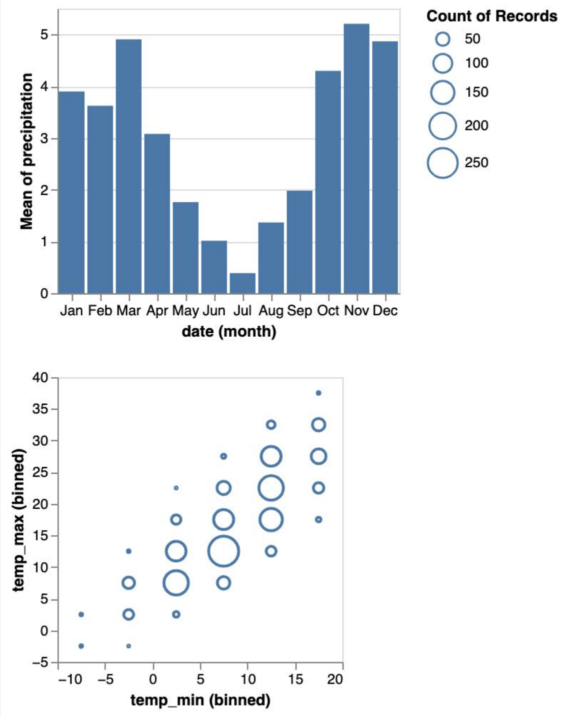
```
{...
 "layer": [
   {
     "mark": "bar",
     "encoding": {
       "x": {"field": "date", "type": "temporal", "timeUnit": "month"},
       "y": {
         "field": "precipitation",
         "type": "quantitative",
         "aggregate": "mean",
         "axis": {"grid": false}
       },
       "color": {"value": "#77b2c7"}
     }
   },
   {
     "mark": "line",
     "encoding": {
       "x": {"field": "date", "type": "temporal", "timeUnit": "month"},
       "y": {
         "field": "temp_max",
         "type": "quantitative",
         "aggregate": "mean",
         "axis": {"grid": false}
       },
       "color": {"value": "#ce323c"}
     }
   }
 ],
 "resolve": {"scale": {"y": "independent"}}
}
```



Concatenation

hconcat([view1, view2, ...], resolve)
vconcat([view1, view2, ...], resolve)

```
{
 ...
 "vconcat": [
   {
     "mark": "bar",
     "encoding": {
       "x": {
         "timeUnit": "month",
         ...
       },
       "y": {
         "aggregate": "mean",
         ...
       }
     }
   },
   {
     "mark": "point",
     "encoding": {
       "x": {
         "field": "temp_min",
         ...
       },
       "y": {
         "field": "temp_max",
         ...
       },
       "size": {
         "aggregate": "count",
         ...
       }
     }
   }
 ]
}
```



Default: shared scale and axis, if **aligned spatial channel have matching data types**

Facet

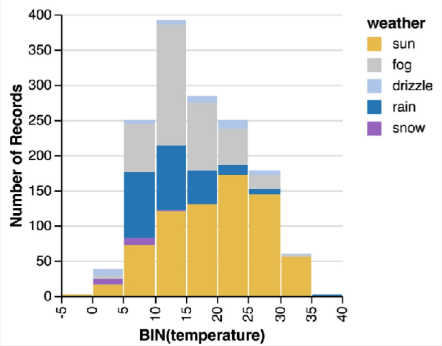
Partition using **distinct values on field**

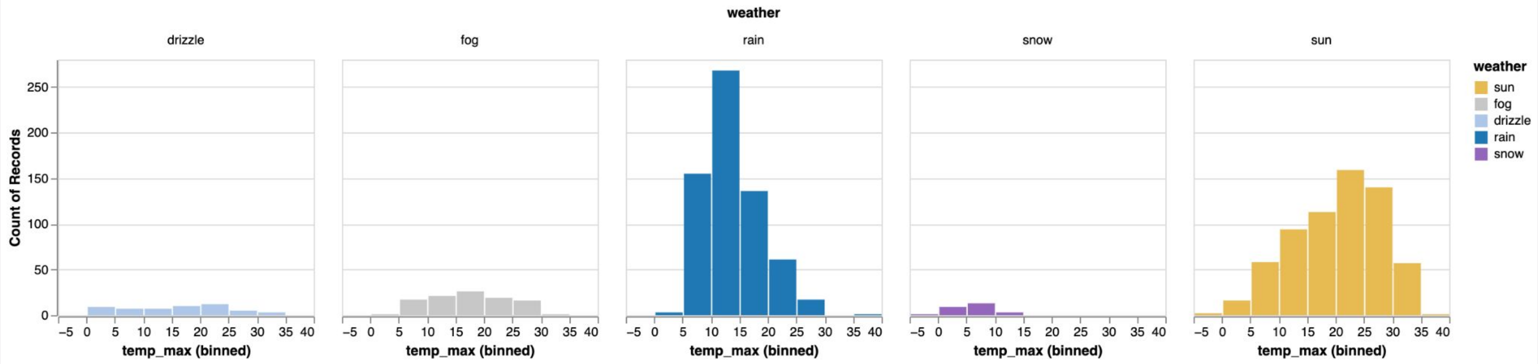
facet(channel, data, field, view, scale, axis, resolve)

Layout direction (row/column)

Shared scales and guides for quantitative fields; avoid empty categories for ordinal scales

```
{
 ...
 "encoding": {
   "x": {"bin": true, "field": "temp_max", "type": "quantitative"},
   "y": {"aggregate": "count", "type": "quantitative"},
   "color": {"field": "weather", "type": "nominal", "scale": {
     "domain": ["sun", "fog", "drizzle", "rain", "snow"],
     "range": ["#e7ba52", "#c7c7c7", "#aec7e8", "#1f77b4", "#9467bd"]
   }},
   "facet": {"field": "weather", "type":"nominal"}
 }
}
```



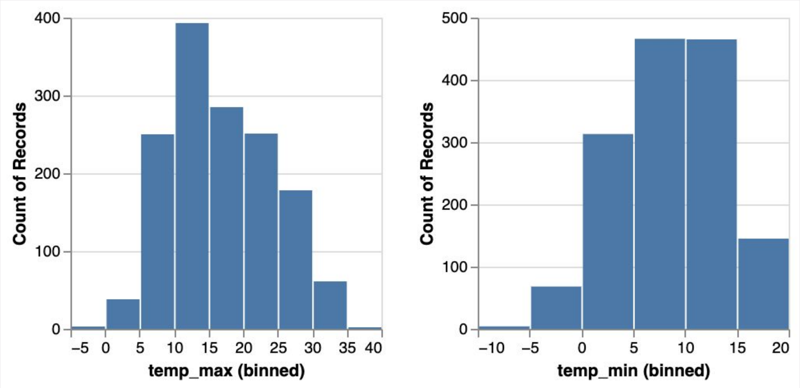


Repeat

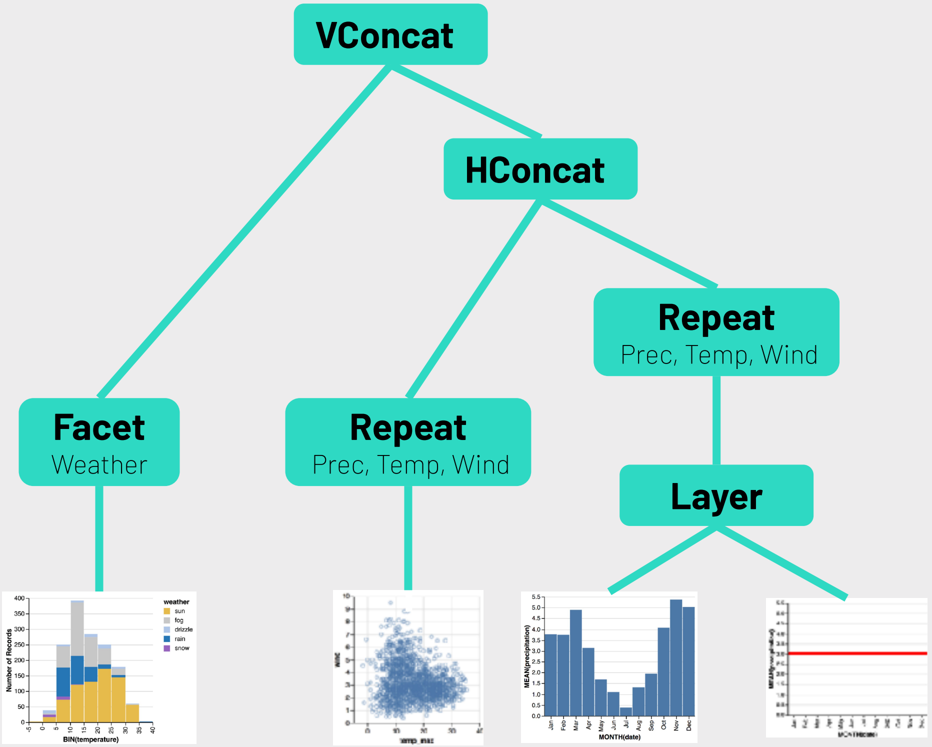
repeat(channel, values, scale, axis, view, resolve)

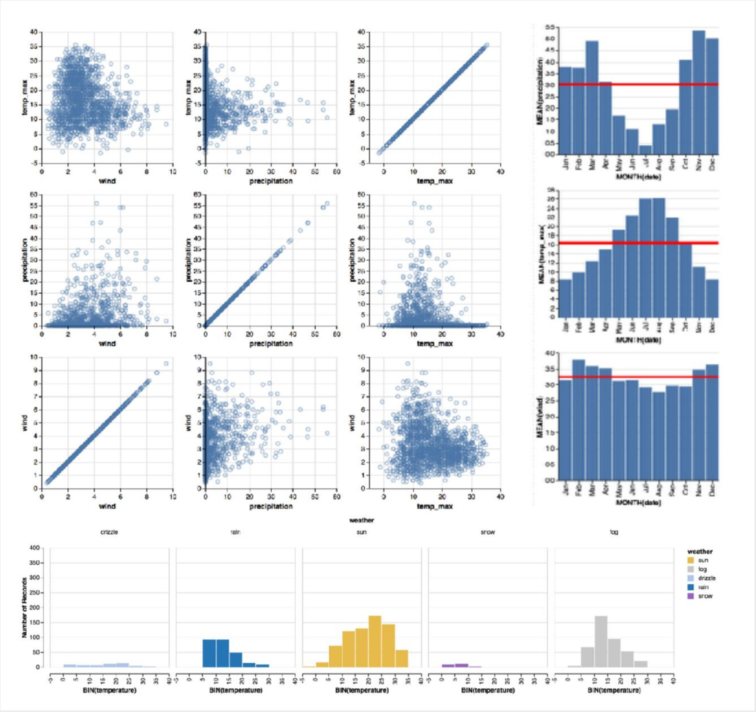
Default: independent scales and axes, shared legends when data fields coincides

```
{
 "repeat": { "column": ["temp_max","temp_min"] },
 "spec": {
   "data": {"url": "data/seattle-weather.csv"},
   "mark": "bar",
   "encoding": {
     "x": {"bin": true, "field": {"repeat": "column"},
"type": "quantitative"},
     "y": {"aggregate": "count", "type": "quantitative"}
   }
 }
}
```



Nested Views





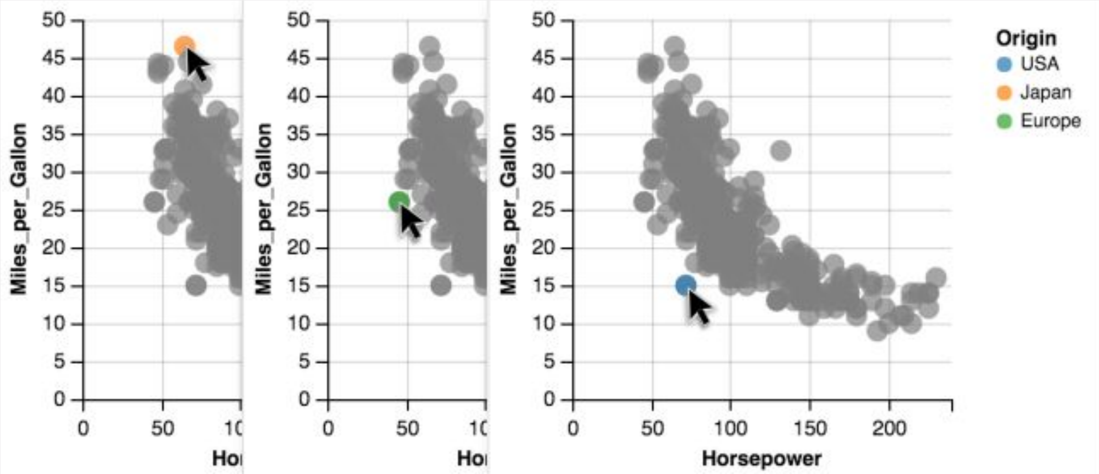
Interactions

To support specification of interaction techniques, Vega-Lite extends the definition of unit specifications to also include a set of selections. ***Selections*** identify the set of points a user is interested in manipulating.

Selection Components

Formal definition: selection := (name, type, predicate, domain|range, event, init, transforms, resolve)

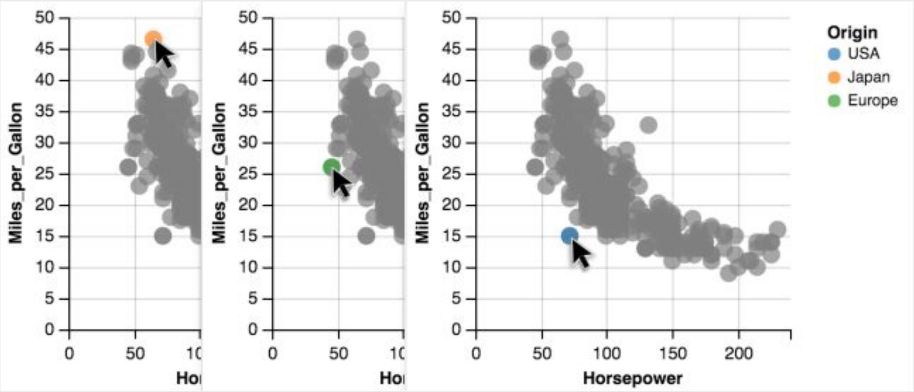
When an input event occurs, the selection is populated with backing points of interest. These points are **the minimal set needed** to identify all selected points.



Selection Components

Formal definition: selection := (name, type, predicate, domain|range, event, init, transforms, resolve)

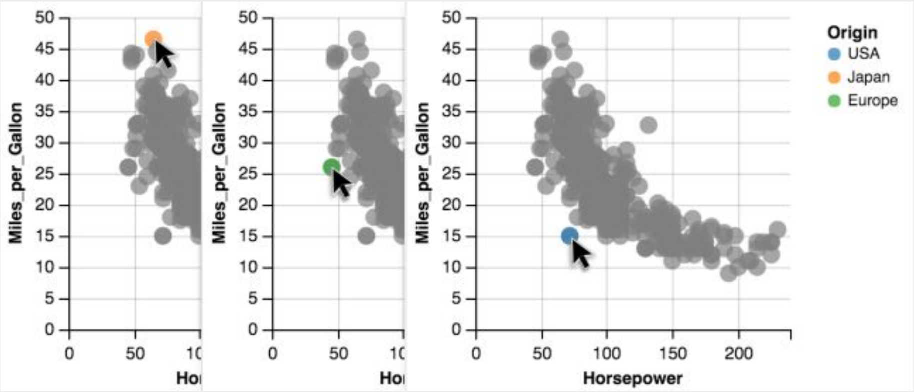
```
{
  "data": {"url": "data/cars.json"},
  "mark": "circle",
  "select": {
    "id": {"type": "point"}
  },
  "encoding": {
    "x": {"field": "Horsepower", "type": "Q"},
    "y": {"field": "MPG", "type": "Q"},
    "color": [
      {"if": "id", "field": "Origin", "type": "N"},
      {"value": "grey"}
    ],
    "size": {"value": 100}
}
```



Selection Components Example

How points are highlighted in a scatterplot using point and list selections

Adding a single **point selection** to parameterize the fill color of a scatterplot's circle mark.

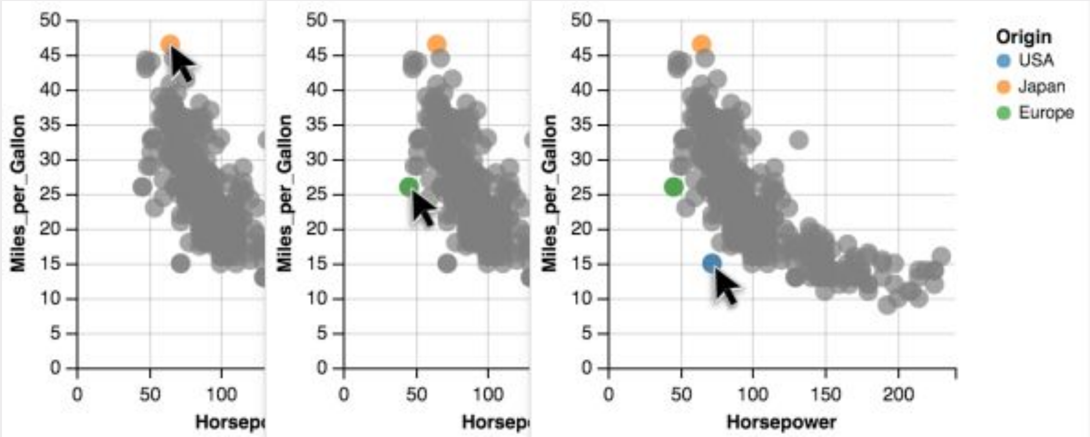


Selection Components Example

How points are highlighted in a scatterplot using point and list selections

```
"id": {"type": "list", "toggle": true}
```

Switching to a **list selection**, with the **toggle transform** automatically added (true enables default shift-click event handling).

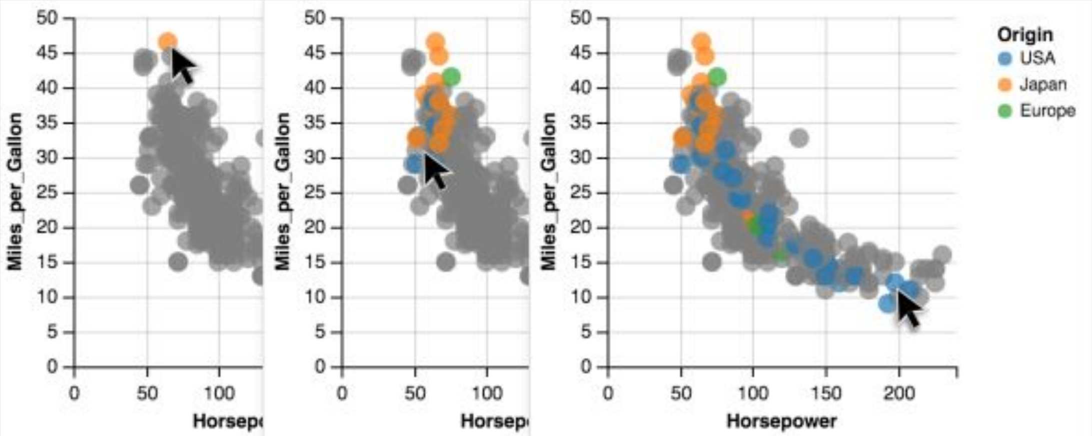


Selection Components Example

How points are highlighted in a scatterplot using point and list selections

```
"id": {"type": "list", "on": "mouseover", "toggle": true}
```

Specifying a **custom event trigger**: the first point is selected on mouseover and subsequent points when the shift key is pressed (customizable via the toggle transform).

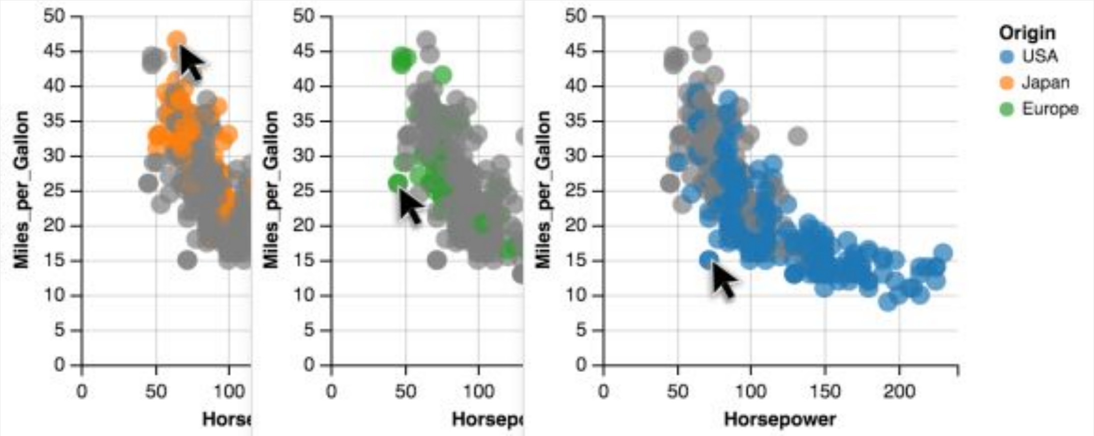


Selection Components Example

How points are highlighted in a scatterplot using point and list selections

```
"id": {"type": "point", "project": {"fields": ["Origin"]}}
```

Using the **project transform** with a single-point selection to highlight all points with a matching `Origin`

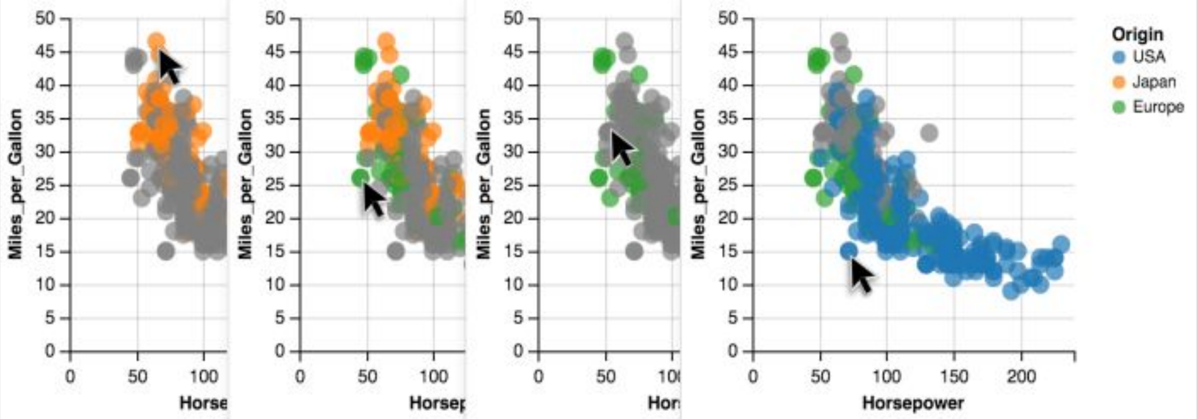


Selection Components Example

How points are highlighted in a scatterplot using point and list selections

```
"select": {
  "id": {"type": "list", "toggle": true, "project": {"fields": ["Origin"]}}
}, ...
```

Combining it with a **list selection** to select multiple `Origins`



Selection Transforms

Selection Transforms are **composable operators** that modify a selection's components.

We have identified **five types** of transforms as a minimal set to support both common and custom interaction techniques.

Selection Transforms

- *project(fields, channels):* Alters a selection's predicate function to determine inclusion by matching only the given fields.

- *toggle(event):* When the event occurs, the corresponding point is added or removed from a list selection's backing dataset.

- *translate(events, by):* Offsets the spatial properties (or corresponding data fields) of backing points by an amount determined by the coordinates of the sequenced events.

- *zoom(event, factor):* Applies a scale factor, determined by the event, to the spatial properties (or corresponding data fields) of backing points.

- *nearest():* Computes a Voronoi decomposition, and augments the selection's event processing, such that the data value or visual element nearest the selection's triggering event is selected.

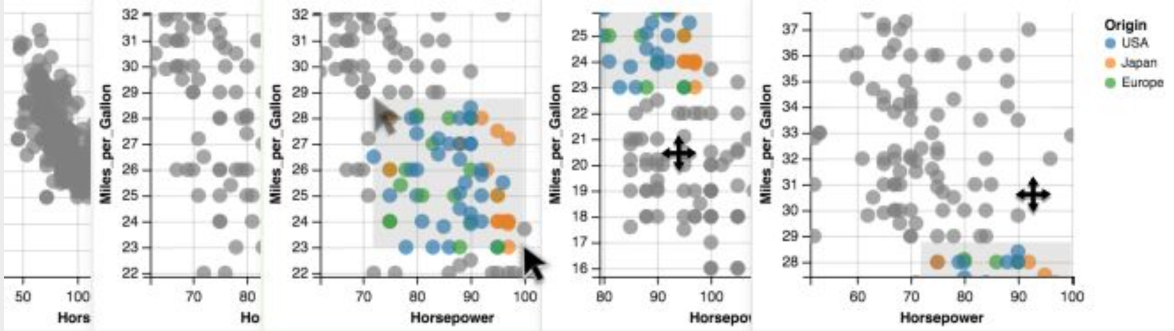
Selection-Driven Visual Encodings

Selections **parameterize visual encodings** to make them interactive — visual encodings are automatically reevaluated as selections change. Selections have three main uses:

- Selections can be used to drive an `if-then-else` **chain of logic** within an encoding channel definition.
- Selected points can be explicitly materialized and used as **input data** for other encodings within the specification.
- A materialized selection can also **define scale extents**, which is very useful when performing zooming or panning.

Visual Encoding Example

A materialized selection can also **define scale extents**, which is very useful when performing zooming or panning.



First initialize a list selection with the x and y scale domain, and then apply translate and zoom.

```
"select": {
  "region": {
    "type": "interval",
    "on": "[mousedown[event.shiftKey], mouseup] > mousemove"
  },
  "grid": {
    "type": "interval", "init": {"scales": true}, "zoom": true
    "translate": "[mousedown[!event.shiftKey], mouseup] > mousemove"
  }
}, ...
```

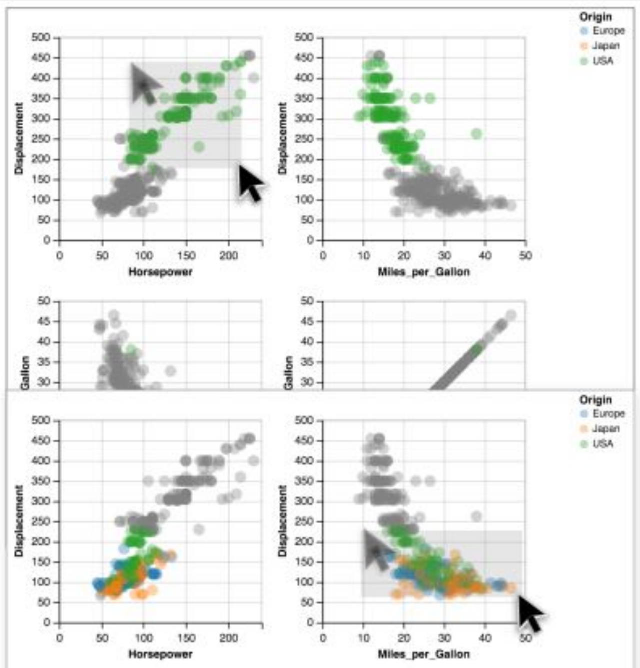
Disambiguating Composite Selections

A selection's events are registered on the unit's mark instances, and materializing a selection applies its predicate against the unit's input data by default. When units are composite, however, selection definitions and applications become **ambiguous**.

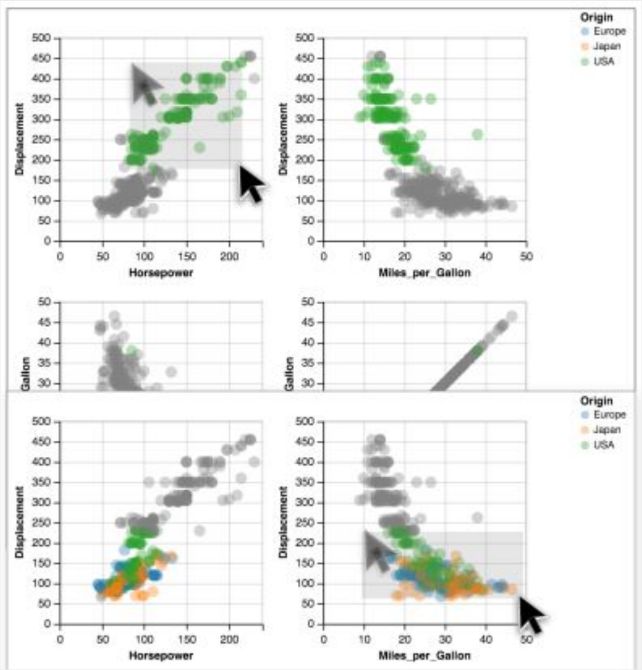
Disambiguating Composite Selections - Brush Example

```
"select": {
  "region": {
    "type": "interval", "translate": true, "zoom": true,
    "on": "[mousedown[event.shiftKey], mouseup] > mousemove",
    "resolve": "single" },
  "grid": {
    "type": "interval", "init": {"scales": true}, "zoom": true
    "translate": "[mousedown[!event.shiftKey], mouseup] > mousemove",
    "resolve": "single"
  }
}
```

Is there one region for the overall visualization, **or** one per cell? If the latter, **which** cell's region should be used?



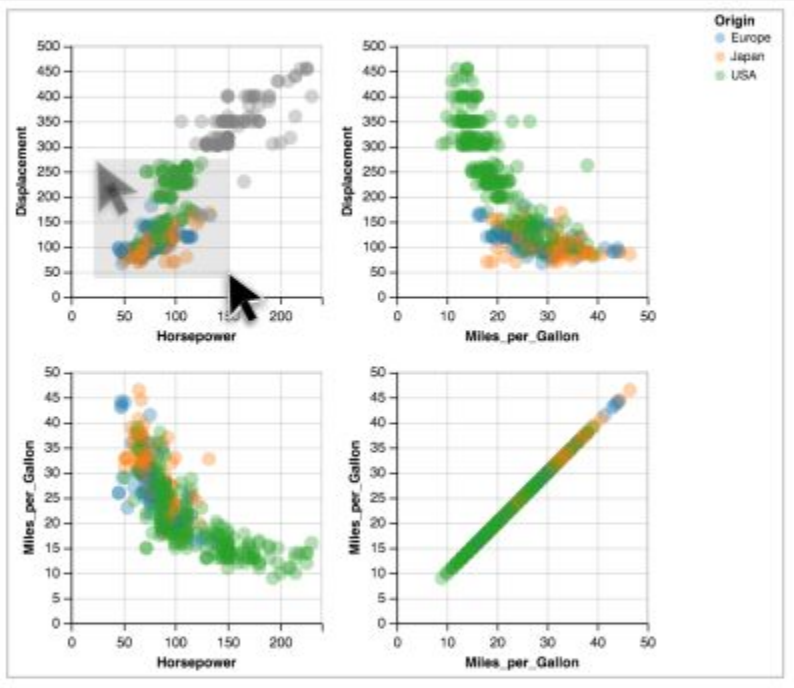
Disambiguating Composite Selections – Brush Example



Single, Independent, Union, Intersect

Composite selections are resolved to a ***single*** global selection: brushing in a cell **replaces** previous brushes. This is the default resolve.

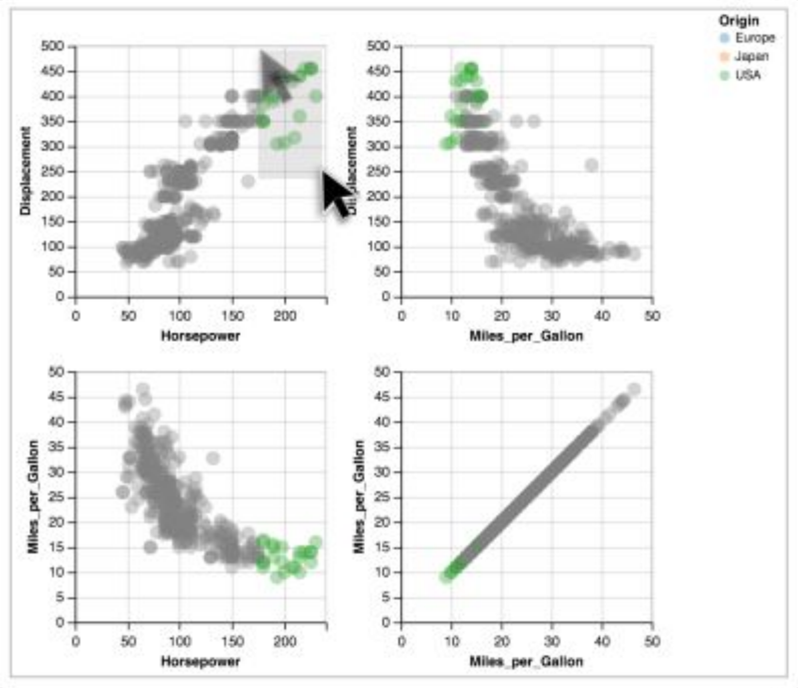
Disambiguating Composite Selections - Brush Example



Single, Independent, Union, Intersect

Independent resolve: each cell uses its **own** brush

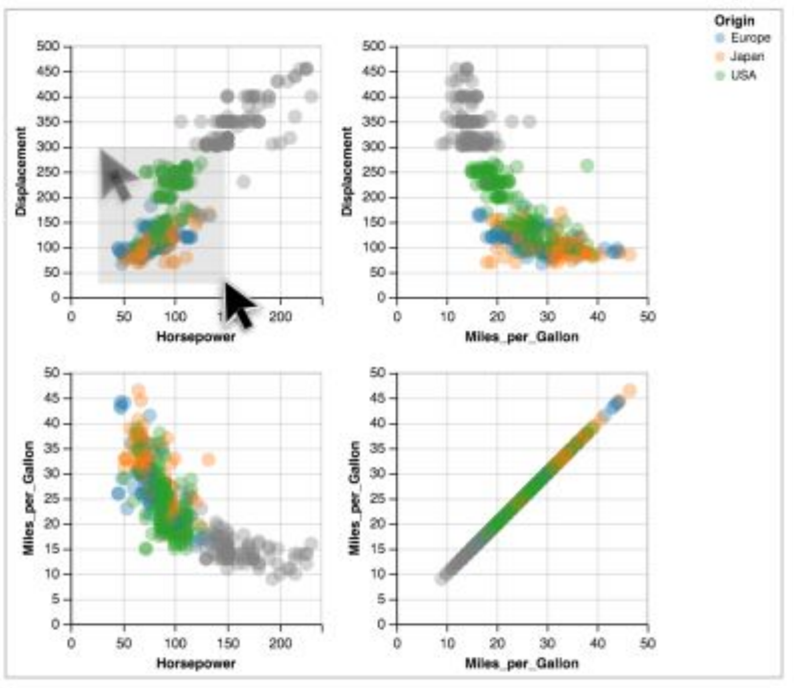
Disambiguating Composite Selections - Brush Example



Single, Independent, Union, Intersect

Union resolve: points are highlighted if they fall in **any** brush

Disambiguating Composite Selections - Brush Example



Single, Independent, Union, Intersect

Intersect resolve: points are highlighted only if they are within **all** brushes

Content

01.

Intro and Background

- What's Vega-lite
- Why Vega-lite

02.

Vega-Lite Grammar Design

- Single View Specification
- Multi-view Composition
- Interactions

03.

Vega-Lite Compiler

- Architecture

04.

Example Visualizations

- Seven categories of techniques

05.

Discussions & Conclusion

- Limitations
- Future work

Compiler Architecture

The compiler compiles the **high-level Vega-Lite specification** to a **low-level Reactive Vega specification** for execution. There are two challenges:

- There is **no one-to-one correspondence** between components of the Vega-Lite and Vega specifications.
- To facilitate rapid authoring of visualizations, Vega-Lite specifications **omit lower-level details** including scale types and the properties of the visual elements such as the font size.

The compiler must resolve the resulting ambiguities.

Compiler Architecture - Parse

Firstly, the compiler **parses** a Vega-Lite specification to disambiguate it. It does so primarily by applying rules crafted to produce **perceptually effective** visualizations. For example, if the color channel is mapped to an nominal field, and the user has not specified a scale domain, a categorical color palette is inferred. If the color is mapped to a quantitative field, a sequential color palette is chosen instead.

Compiler Architecture – Build

Secondly, the compiler **builds** an internal representation of this unambiguous specification, consisting of **a tree of models**. Each model represents a unit or composite view produced by the algebraic operators described before, and stores a series of components, effectively **bridging the gulf** between the two levels of abstraction.

Compiler Architecture - Merge

Once the necessary components have been built, the compiler performs a **bottom-up traversal** of the model tree to **merge** redundant components. This step is critical for ensuring that the resultant Vega specification **does not** perform unnecessary computation that might hinder interactive performance.

Compiler Architecture - Assemble

Finally, the compiler **assembles** the requisite Vega specification. **Selection components**, in particular, produce **signals** to capture events and the necessary backing points, and list and intervals construct data sources as well to hold multiple points. Each run-time selection transform (i.e., **trigger transforms** mentioned earlier) generates signals **as well**, and may augment the selection's data source with data transformations.

Content

Example Visualizations-Seven categories of techniques

To **evaluate expressivity**, we choose examples that cover Yi et al.'s taxonomy of interaction methods, consisting of seven categories of techniques:

- **Select**: to mark items of interest
- **Explore**: to examine subsets of the data
- **Encode**: to change the visual representations used
- **Connect**: to highlight related items within and across views
- **Abstract/elaborate**: to vary the level of detail
- **Reconfigure**: to show different arrangements of the data
- **Filter**: to show elements conditionally

Results & Comparisons

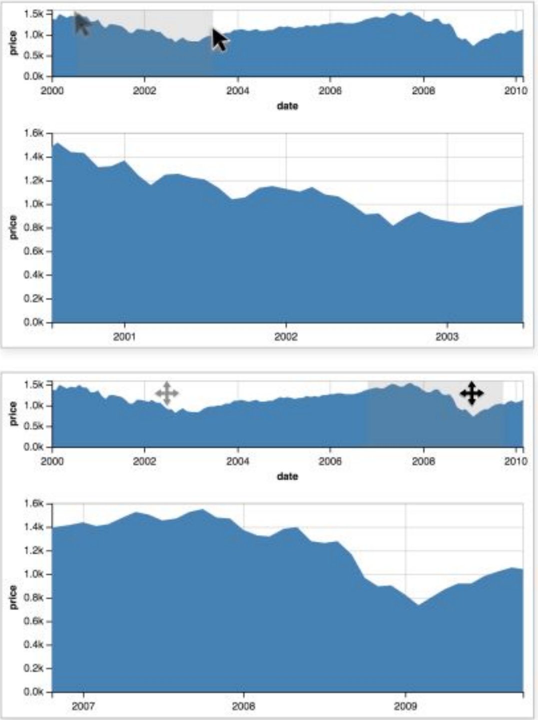
- **Select:**
  Vega-Lite specifications are ***an order of magnitude more concise*** than their Vega counterparts. With Vega-Lite, users need ***only*** specify the semantics of their interaction and the compiler fills in appropriate default Values. With Vega, users need to ***manually*** author all the components of an interaction technique.

- **Explore & Encode:**
  Vega-Lite's higher-level approach not only offers ***more rapid specification***, but it can also ***enable*** interactions that a user may not realize are expressible with lower-level representations

- **Connect:**
  To move from a single interactive scatterplot to an interactive SPLOM, Vega requires an ***extra*** level of indirection to identify the specific cell a user is interacting in, and to ensure that the correct data values are used to determine inclusion within the brush. In Vega-Lite, this complexity is succinctly encapsulated by the ***resolve*** keyword which can be systematically varied to explore alternatives

Results & Comparisons - Abstract/elaborate

```
{
  "vconcat": [
    {
      "data": {"url": "data/sp500.csv","formatType": "csv"},
      "mark": "area",
      "select": {
        "region": {
          "type": "interval",
          "project": {"channels": ["x"]}
        }
      },
      "encoding": {
        "x": {"field": "date", "type": "temporal", ...},
        "y": {"field": "price", "type": "quantitative", ...}
      }
    },
    {
      "data": {"url": "data/sp500.csv","formatType": "csv"},
      "mark": "area",
      "encoding": {
        "x": {
          "field": "date", "type": "temporal", ...,
          "scale": {"domain": {"selection": "region"}}
        },
        "y": {"field": "price","type": "quantitative"}
      }
    }
  ]
}
```

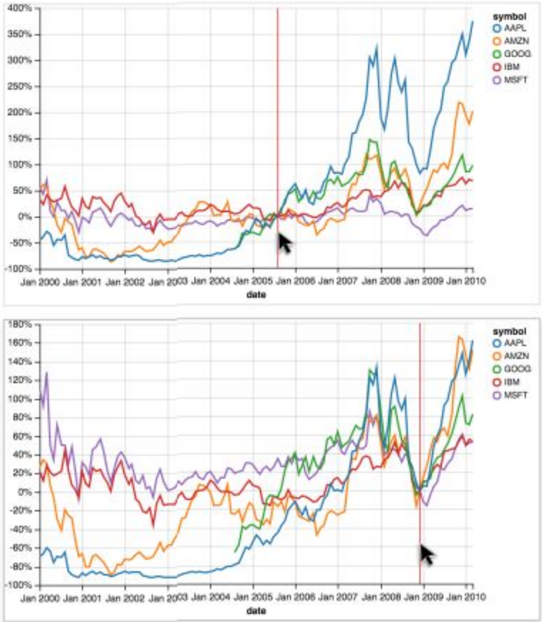


A selection defined in one unit specification can be explicitly given as the **scale domain** of another in a concatenated display.

Doing so creates an **overview + detail interaction**: brushing in the top (overview) chart displays only the brushed items at a higher resolution in the larger (detail) chart at the bottom.

Results & Comparisons - Reconfigure

```
{
  "data": {"url": "data/stocks.csv"},
  "layers": [{
    "transform": {
      "lookup": {
        "index": {"selection": "indexPt", "keys": ["symbol"]}
      },
      "calculate": [{
        "field": "indexed_price",
        "expr": "(datum.price - datum.index.price)/datum.index.price"
      }]
    },
    "select": {
      "indexPt": {
        "type": "point", "on": "mousemove",
        "project": {"fields": ["date"]},
        "nearest": true
      }
    },
    "mark": "line",
    "encoding": {
      "x": {"field": "date", "type": "temporal", ...},
      "y": {"field": "indexed_price", "type": "quantitative", ...},
      "color": {"field": "symbol", "type": "nominal"}
    }
  }, {
    "mark": "rule",
    "encoding": {
      "x": {"selection": "indexPt.date", "type": "temporal"},
      "color": {"value": "red"}
    }
  }]
}
```



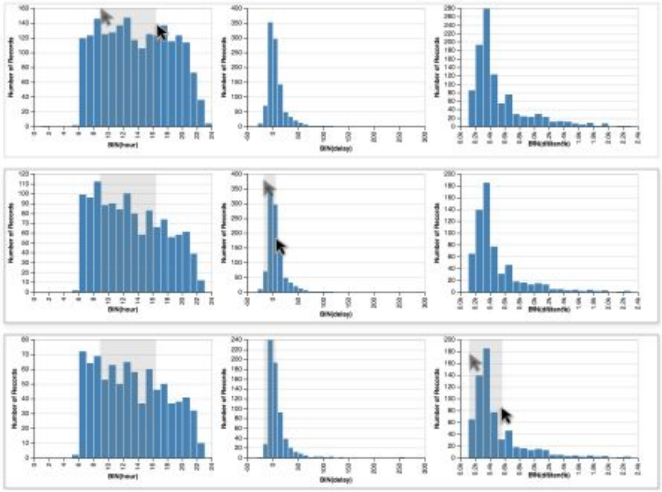
By projecting the date field, the point selection represents **both** a single data value as well as a set of values that share the selected date.

We can **reference** the point selection **directly**, to position the red vertical rule, and also **materialize** it as part of the lookup data transform.

Results & Comparisons - Filter

As selections provide a predicate function, it is **trivial** to use them to filter a dataset.

```
{
  "repeat": {"column": ["hour", "delay", "distance"]},
  "spec": {
    "select": {
      "region": {
        "type": "interval",
        "project": {"channels": ["x"]},
        "resolve": "intersect_others"
      }
    },
    "data": {"url": "data/flights-2k.json"},
    "transform": {
      "filterWith": "region",
      ...
    },
    "mark": "bar",
    "encoding": {
      "x": {"field": {"repeat": "column"}, "bin": true, ...},
      "y": {"aggregate": "count", "field": "*", ...},
      "color": {"value": "steelblue"}
    }
  }
}
```



As the user brushes in one histogram, the datasets that drive each of the other two are **filtered**, the data values are re-aggregated, and the bars rise and fall.

The Vega-Lite compiler **automatically instantiates** the translate transform, allowing users to drag brushes around rather than having to **reselect** them from scratch.

Content

01.

Intro and Background

- What's Vega-lite
- Why Vega-lite

02.

Vega-Lite Grammar Design

- Single View Specification
- Multi-view Composition
- Interactions

03.

Vega-Lite Compiler

- Architecture

04.

Example Visualizations

- Seven categories of techniques

05.

Discussions & Conclusion

- Limitations
- Future work

Limitations & Future Work

- **Model architecture limitation**: components that are determined during compiling ***cannot*** be manipulated interactively, For example, a selection ***cannot*** specify alternate fields to bin or aggregate over and more complex selection types (e.g., lasso selections) ***cannot*** be expressed as the Vega-Lite system does not support arbitrary path marks. Some ***alternative systems*** such as an interpreter that instantiates its grammar could potentially circumvent this issue.

- **Limited support for highly specialized methods**: specialized methods such as querying time-series with relaxed selections ***cannot*** be expressed by default grammar and may need to implement ***custom transforms*** to extend the base semantics. Hopefully by making the system open source, there could be some ***community-built additions*** that address highly specialized methods.

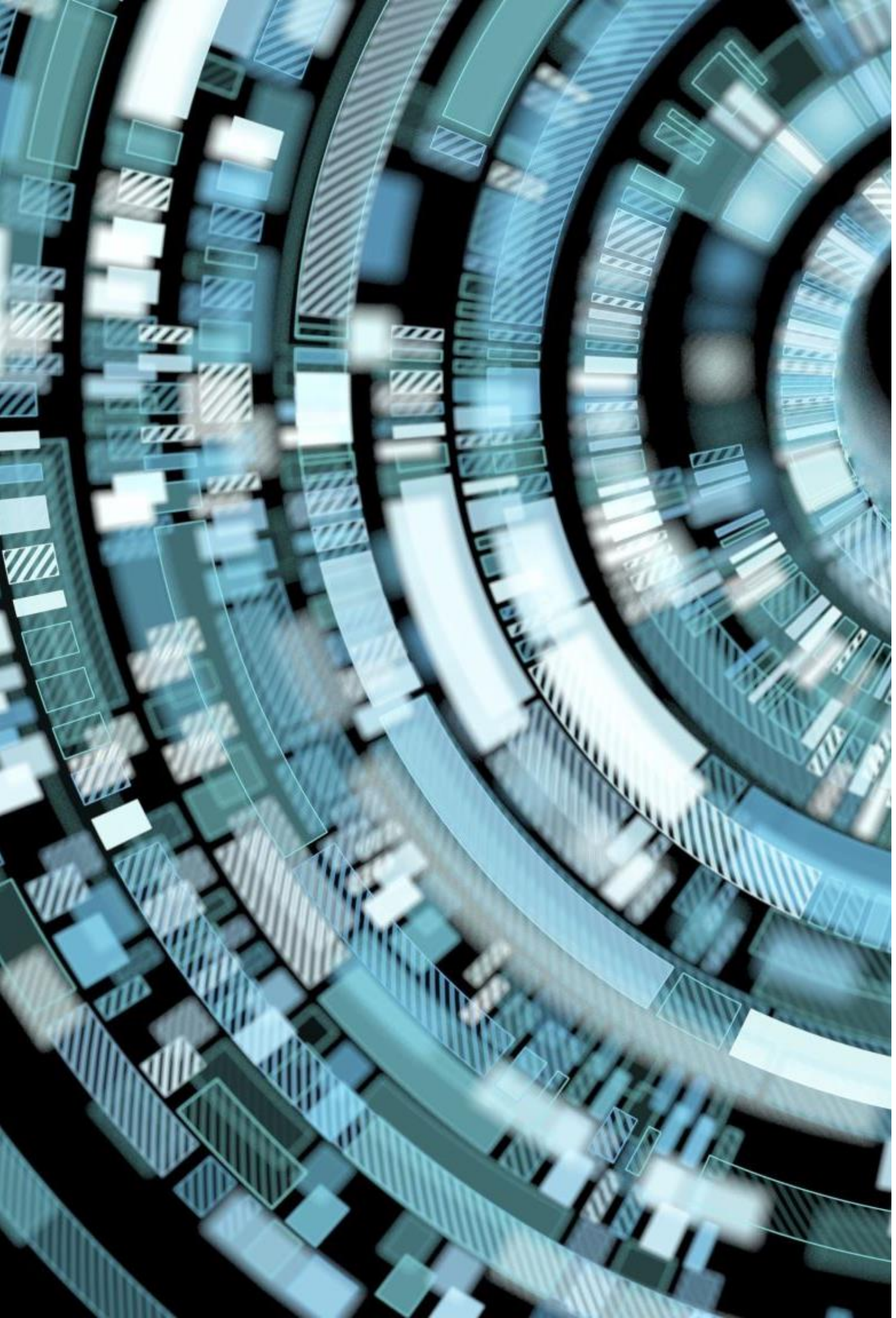
Thank you!

Any Questions?

Vega-Lite: A Grammar of interactive graphics

Practitioner Presentation

Aniruddha Mysore



Scenarios - Need for Visualization

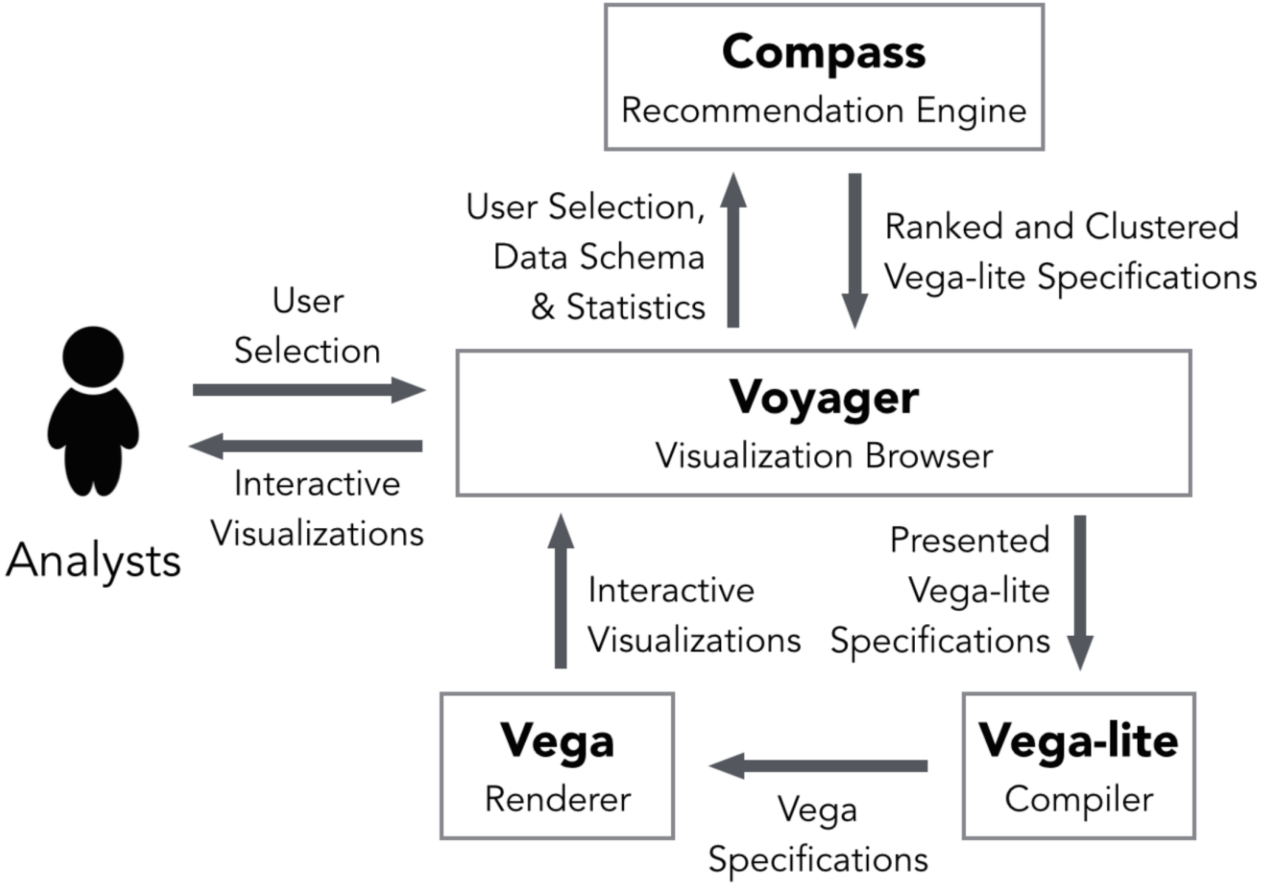
- We want to set guidelines for making graphics across the company and provide commonly used visualizations as a library
- Should support interactive graphics
- Ideally: High-level declarative language
- Bonus: Should be supported on many platforms
    - Python for data analysts, notebooks
    - JavaScript for embedding on dashboards

Vega-Lite Usage Scenarios

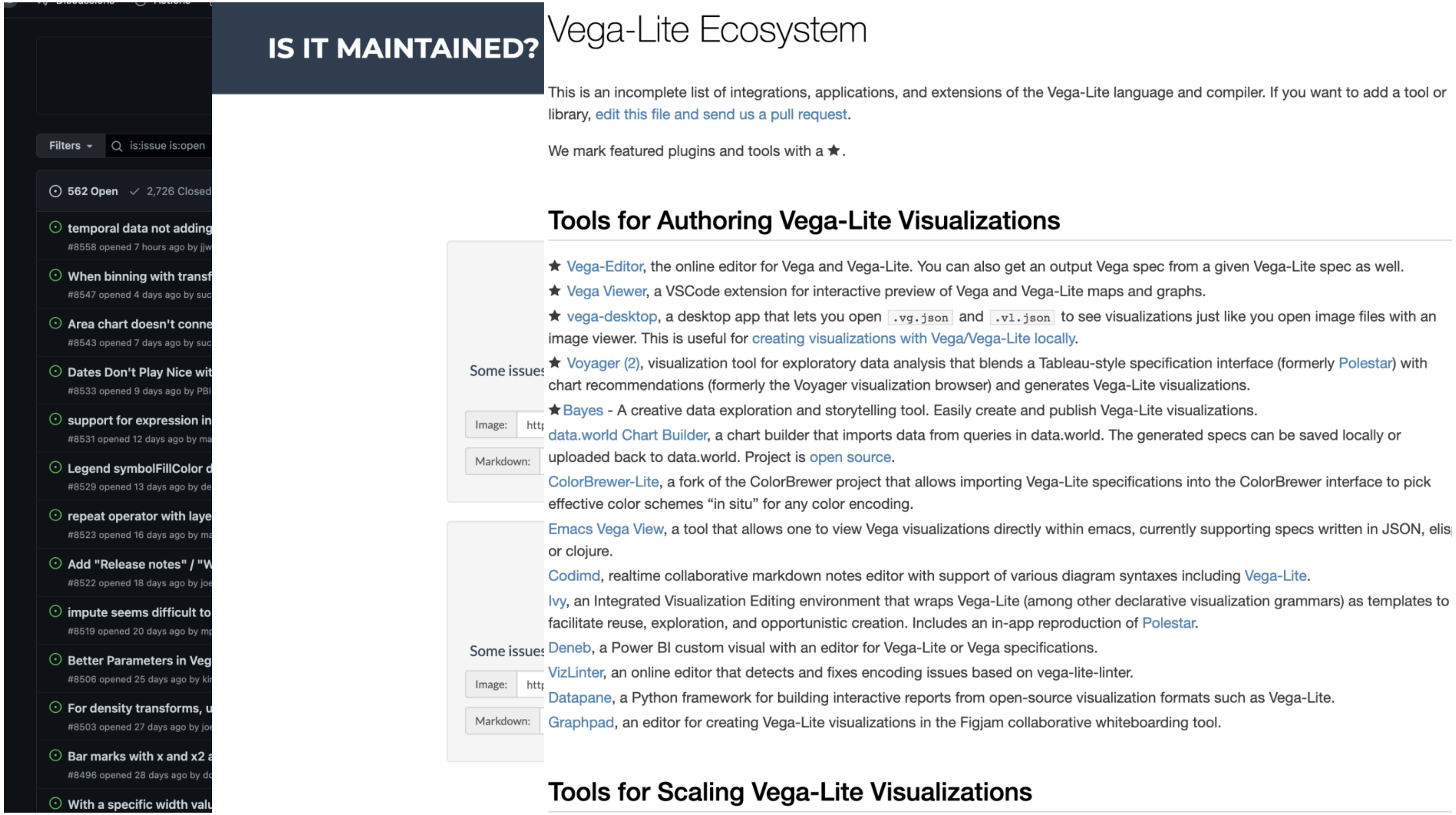
- Visualization for a single language/framework
  - Python – Altair
  - Julia – Vega Lite for Julia
  - Rust – Vega Lite for Rust
  - R – Vega Lite for R

- Complete stack for automated visualization

The Vega/Voyager stack



Is vega-lite a *good* open-source package?



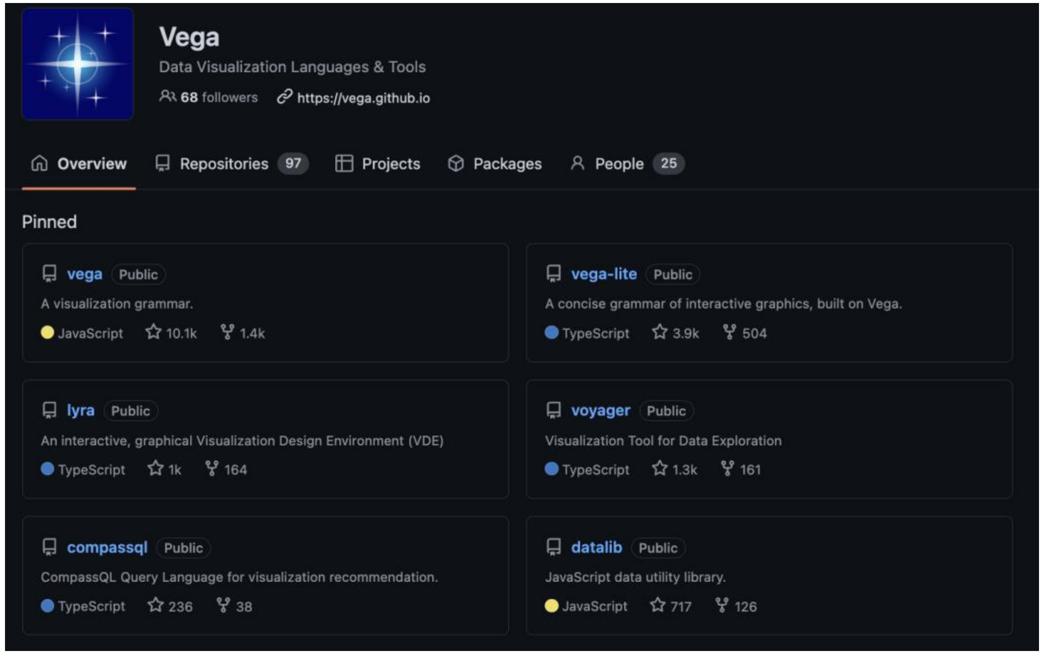
Design Space of Data Visualization Libraries
(on the web)

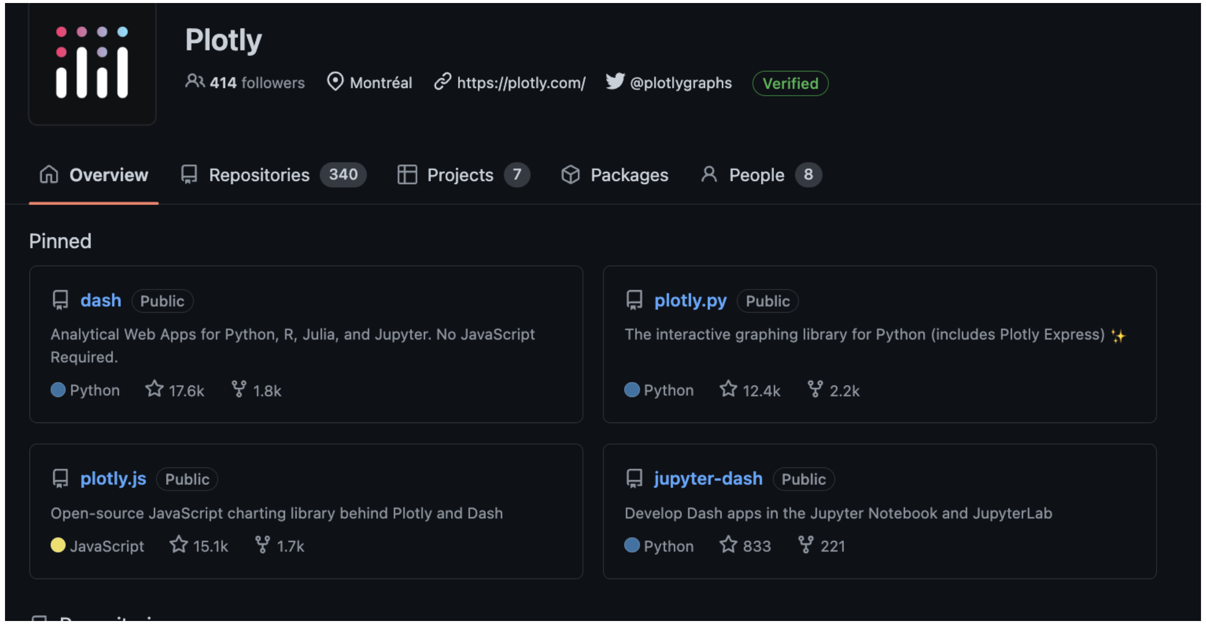
API Design

Level of Abstraction		Framework-specific	Plain JS	JSON + callbacks	JSON
High-level Less expressivity ~Less effort	Chart Templates	nivo vue-trend Recharts	Google Charts G2Plot dc.js	Chart.js Chartist.js	FusionCharts
	High-level Building Blocks	Victory React-Vis Semiotic @deck.gl/react	dimple	ECharts HighCharts Plotly @deck.gl/core	@deck.gl/json
	Visualization Grammars	Chart-Parts	G2 Muze		Vega-Lite Vega
	Low-level Building Blocks	visx	D3 d3-annotation cola flubber labella		
Low-level More expressivity ~More effort	Graphics Libraries	react-rough react-three-fiber	p5*js Rough.js three.js pixi.js		

Navigating the Wide World of Data Visualization Libraries, [Krist Wongsuphasawat]

Possible Candidates





Would we use this?

“

> When deciding which library to use, look for the appropriate abstraction level for the time you have, your own coding comfort, the tasks you are trying to accomplish, and the target developers and users. Then look at API design and other factors that might be included into the consideration, such as:
>
> - **Rendering technology:** SVG, Canvas, WebGL
> - **Performance:** Bundle size, Speed, Server-side Rendering
> - **Others:** Type-safety, License, Theming, Animation, etc.

”

Krist Wongsuphasawat

- Will go with what developers prefer

Bonus: Entrepreneur Role

Crime pays, but (good) research pays more

TRIFACTA

- Potter's wheel (2001), Data Wrangler - visual interaction & intelligent inference for data transform (2011)
- Company founded 2012, product for data transformation and visualization
- Joe Hellerstein (UC Berkeley) Jeffrey Heer (UWash) and Sean Kandel (Stanford)
- Raised $76 million

tableau

- Visualization for data cubes and relational databases (1999)
- Company founded 2003, products for business intelligence and viz dashboards
- Christian Chabot, Pat Hanrahan and Chris Stolte from Stanford University
- Sold to Salesforce for $16.3 billion

Vega-Lite: A Grammar of Interactive Graphics

Reviewer: Qiandong Tang

In Summary

- Vega-Lite is a grammar that enables concise and high-level specifications of interactive data visualizations
- Introduce an algebra for constructing composite views using layer, concatenate, facet, and repeat operators
- Extend the Vega grammar to support interaction by adding selection components and selection transformation operators

Strong Points

- Concise and portable - Domain-specific languages (DSL) (e.g. JSON) are easy to modify and reusable
- User-friendly - Vega-Lite is easy to install and setup, providing comprehensive documentations and tutorials
- Open-source - Vega-Lite is actively maintained and supported by a mature ecosystem

Weak Points

- Limited expressivity - Some facet and layer combinations could create data ambiguities that prevent Vega-Lite from rendering
- Limited extensibility - Using JSON as the underlying specification could lead it hard to extend
- No grammar checking - Mistakes are invertible when learning new grammars; Linting is critical to reduce mistakes from providing invalid specifications

Weak Points

- Limited expressivity - Some facet and layer combinations could create data ambiguities that prevent Vega-Lite from rendering
- Limited extensibility - Using JSON as the underlying specification could lead it hard to extend
- No grammar checking - Mistakes are invertible when learning new grammars; Linting is critical to reduce mistakes from providing invalid specifications

VizQL (SQL-like syntax, SIGMOD '06)

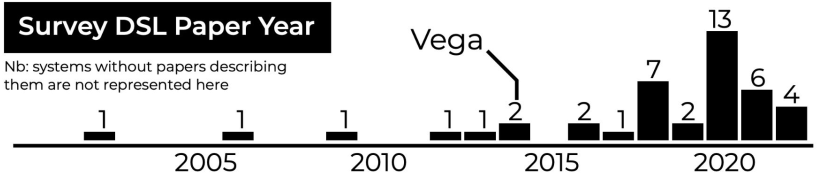


Fig. 2. Since **Vega**'s publication JSON-style DSLs have become popular.

A. M. McNutt, "No Grammar to Rule Them All: A Survey of JSON-style DSLs for Visualization," in IEEE Transactions on Visualization and Computer Graphics, 2022, doi: 10.1109/TVCG.2022.3209460.

Accept

Vega-Lite: A Grammar of Interactive Graphics

Archeologist: Haotian Sun

TAKEAWAYS

- Propose a high-level grammar that allows swift specification of data visualization more interactively.
- Propose a composition algebra and use several operators to transfer the single-view specifications into multi-view ones.
- Use dedicated compiler to bridge the low- and high-level specifications for Vega and Vega-lite, respectively.
- Propose a high-level interaction grammar with compositions of selections and predicates.

PREVIOUS PAPERS - 1999

The Grammar of Graphics

- “The Origin of Things”
- Propose formal grammars for statistical graphics to concisely specify visualizations
- Many follow-ups and commercialization (Tableau, R packages,...)
- Inspire many expressive lower-level grammars, such as Protovis, D3, and Vega, for creating explanatory and highly-customized graphics.
- Similar to this paper, Vega-Lite also represents basic plots with a set of encoding definitions mapping data to visual components (position, color, …) and with data transformations (aggregation, sorting, …)

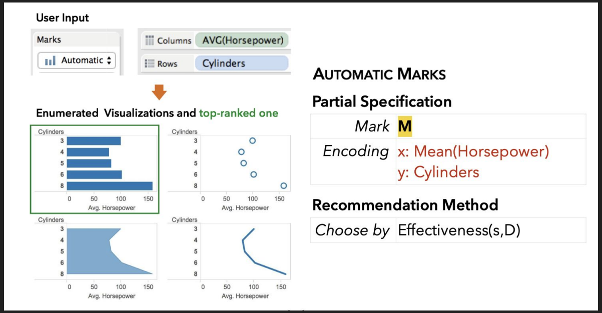
PREVIOUS PAPERS - 2016

Reactive Vega: A streaming dataflow architecture for declarative interactive visualization

- Low-level grammar for explanatory data visualization
- Input events as continuous data streams and uses Event-Driven Functional Reactive Programming (E-FRP) to formulate composable, declarative interaction primitives for data visualization.
- Construct a dataflow graph that can dynamically rewrite itself at runtime by extending or pruning branches in a data-driven fashion.
- Similar to this paper, Vega-Lite uses a portable JSON syntax. A dedicated compiler is used to convert the high-level specifications to the low-end, for Vega-Lite and Vega, respectively. Namely, Vega-Lite specifications are compiled to full Vega specifications.

Towards A General-Purpose Query Language for Visualization Recommendation

- CompassQL, a common framework for visualizing recommender systems in the form of a specification language for querying over the space of visualizations
- CompassQL defines a partial specification that describes enumeration constraints. It extends the Vega-Lite's grammar with explicit enumeration specifiers to define properties that should be enumerated.

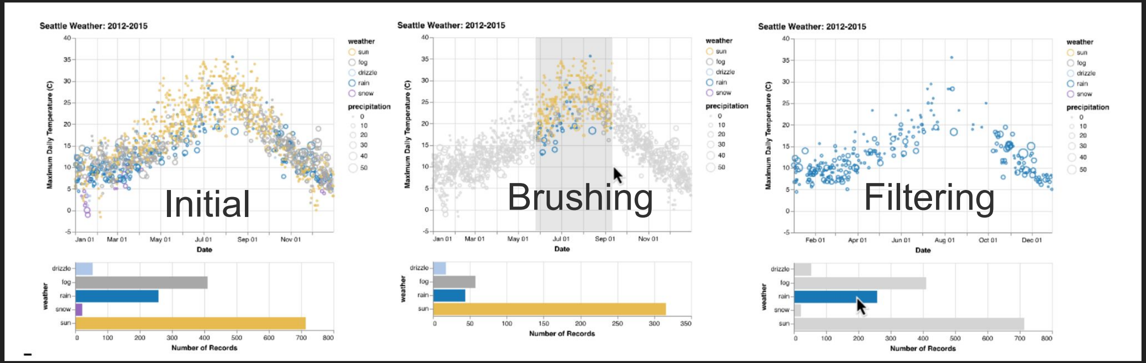


E.g., setting the mark property to M means that the system should enumerate all possible mark types (bar, line, area, point).

FUTURE PAPERS - 2018

Altair: Interactive Statistical Visualizations for Python

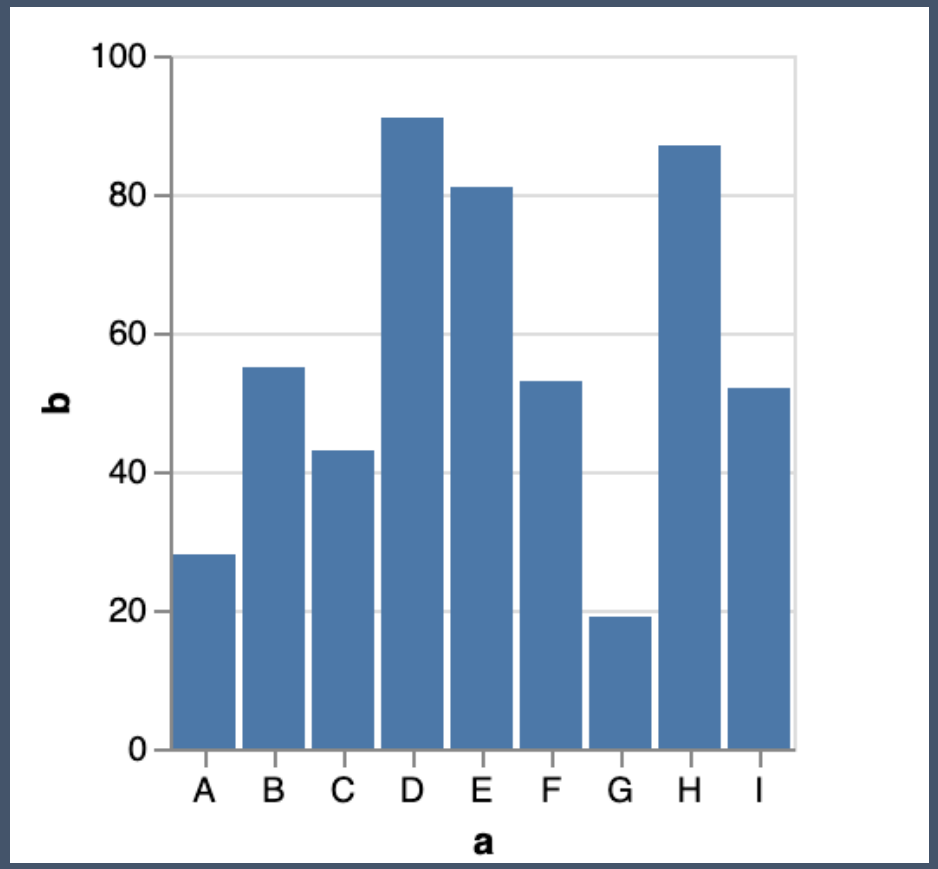
- A declarative statistical visualization library for Python.
- Altair’s Python API emits Vega-Lite JSON data, which is then rendered in a user-interface, such as Jupyter Notebook, JupyterLab, …
- Altair’s Python code are generated from the Vega-Lite JSON schema, ensuring strict compliance with the Vega-Lite specification



Example of an interactive Altair visualization of the weather in Seattle.

THANK YOU

Vega vs Vega-lite



https://vega.github.io/vega/examples/bar-chart/

Vega-Lite JSON Specification

```
{
  "$schema": "https://vega.github.io/schema/vega-lite/v5.json",
  "description": "A simple bar chart with embedded data.",
  "data": {
    "values": [
      {"a": "A", "b": 28}, {"a": "B", "b": 55}, {"a": "C", "b": 43},
      {"a": "D", "b": 91}, {"a": "E", "b": 81}, {"a": "F", "b": 53},
      {"a": "G", "b": 19}, {"a": "H", "b": 87}, {"a": "I", "b": 52}
    ]
  },
  "mark": "bar",
  "encoding": {
    "x": {"field": "a", "type": "nominal", "axis": {"labelAngle": 0}},
    "y": {"field": "b", "type": "quantitative"}
  }
}
```

Next class

Expressive Time Series Querying with Hand-Drawn Scale-Free Sketches

Authors: Harshal, Cangdi

Reviewer: Haotian

Archaeologist: Akshay

Practioner: Siddhi