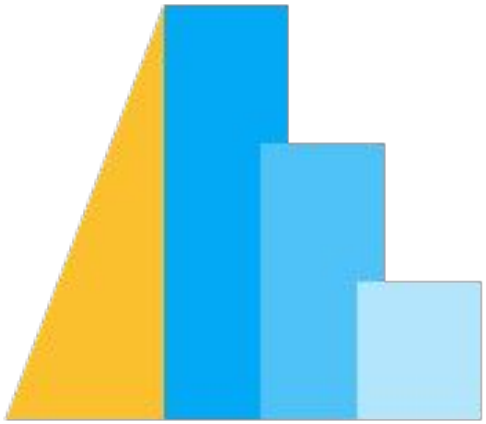
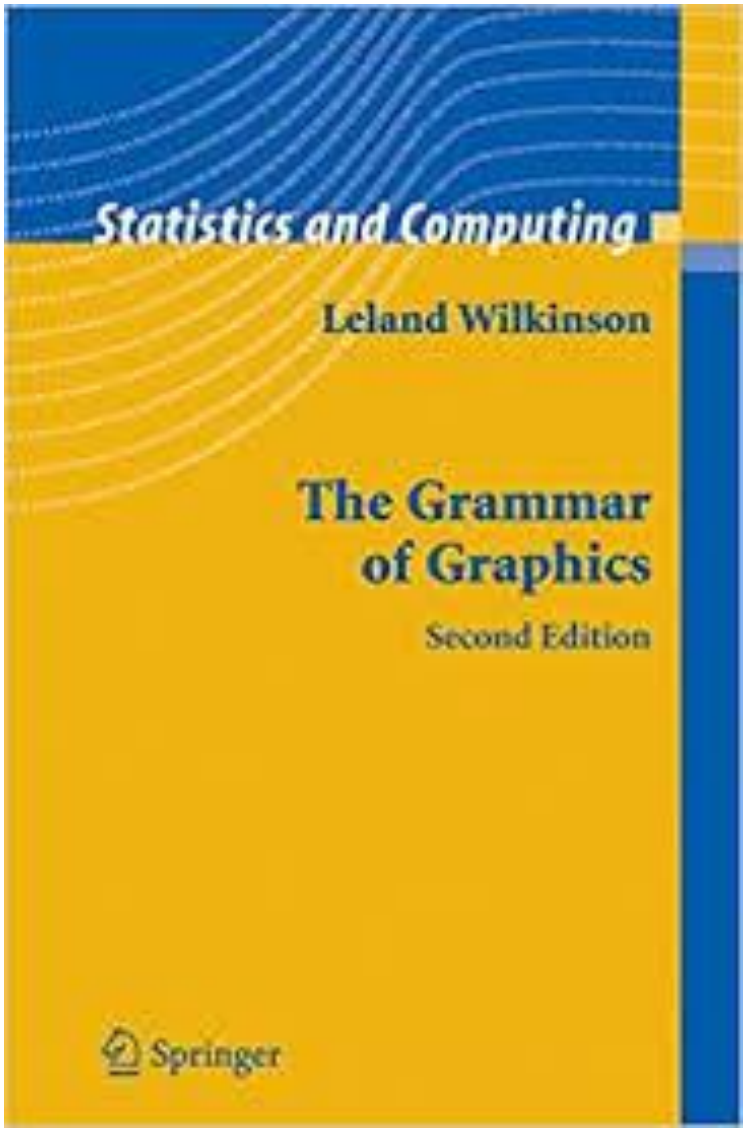


Exploratory Data Visualization with Altair

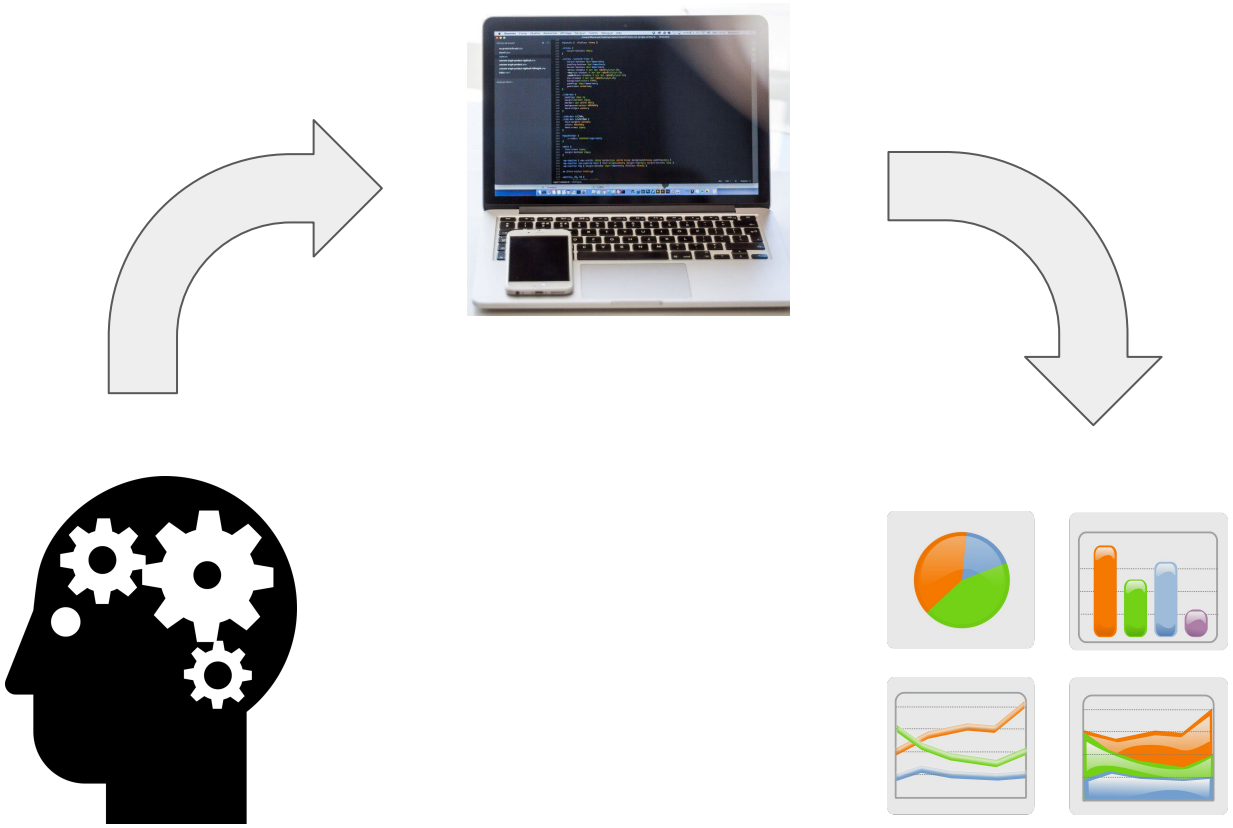
Jake VanderPlas @jakevdp
PyCon 2018

Materials at http://github.com/altair-viz/altair-tutorial

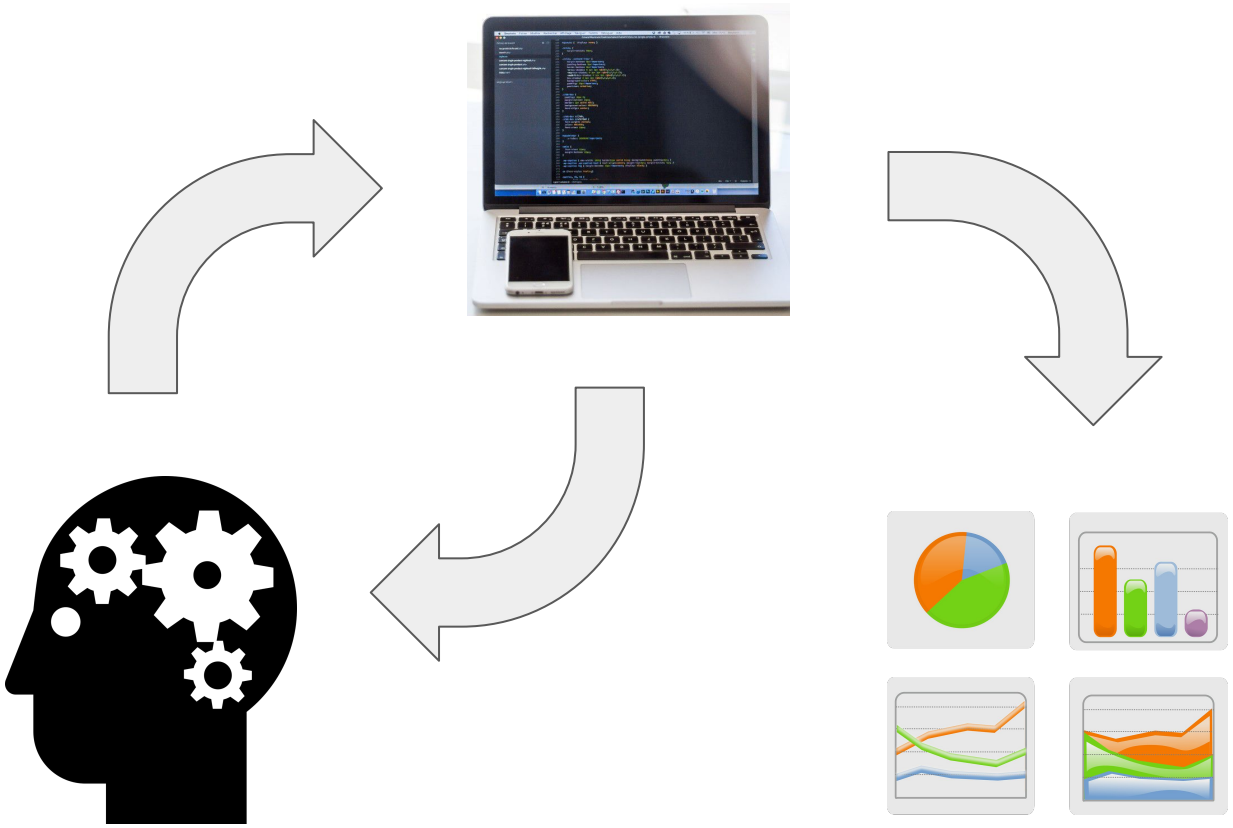


Building Blocks of Visualization:

1. **Data**
2. **Transformation**
3. **Marks**
4. **Encoding** – mapping from fields to mark properties
5. **Scale** – functions that map data to visual scales
6. **Guides** – visualization of scales (axes, legends, etc.)



Key: Visualization *concepts* should map directly to visualization *implementation.*



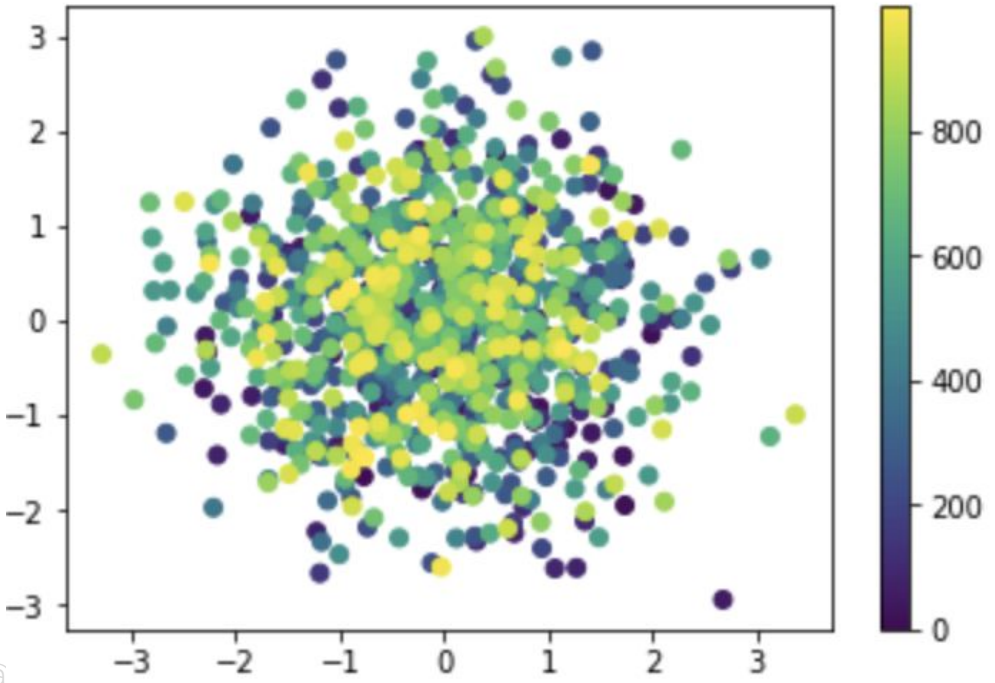
Hypothesis: good *implementation* can influence good *conceptualization.*

~ familiar tools ~

http://matplotlib.org/

Plotting with Matplotlib

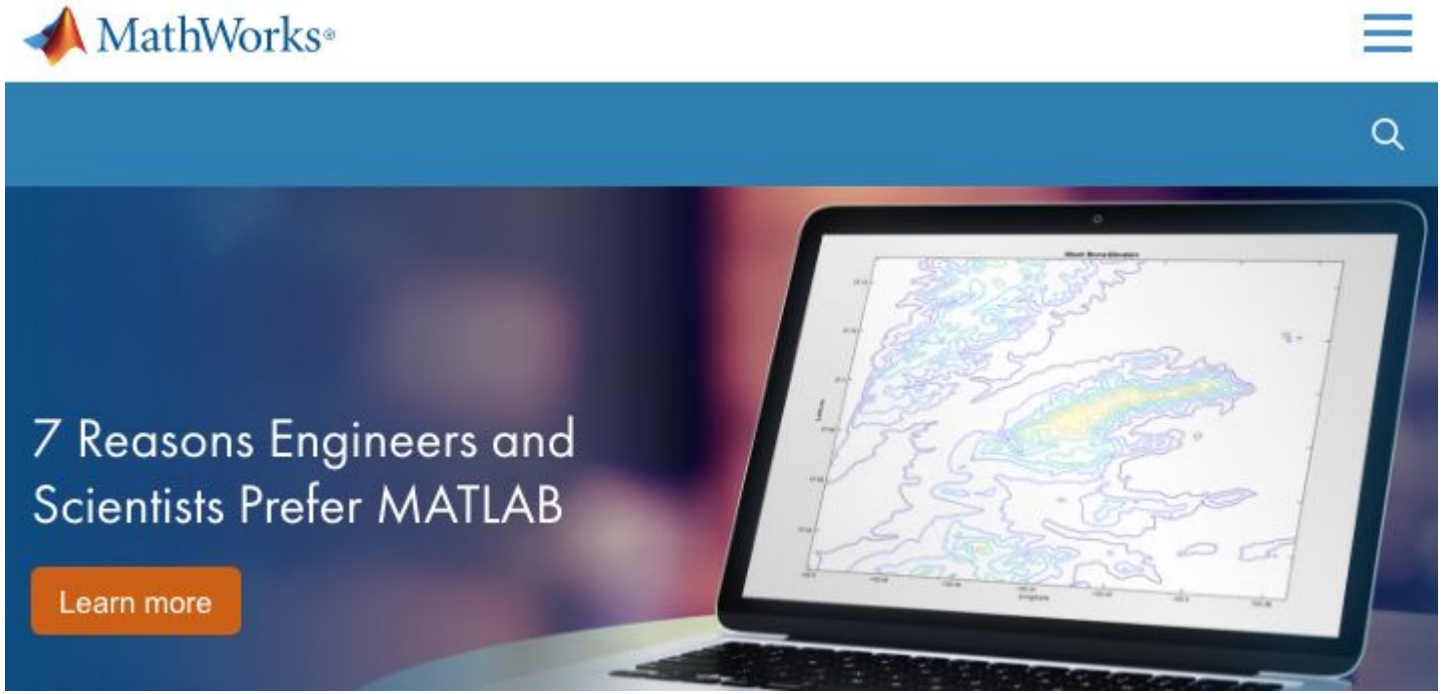
```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.randn(1000)
y = np.random.randn(1000)
color = np.arange(1000)

plt.scatter(x, y, c=color)
plt.colorbar()
```



Plotting with Matplotlib

Strengths:

- Designed like MatLab: switching was easy



For more on the historical perspective, see
https://speakerdeck.com/jakevdp/pydata-101

Plotting with Matplotlib

Strengths:

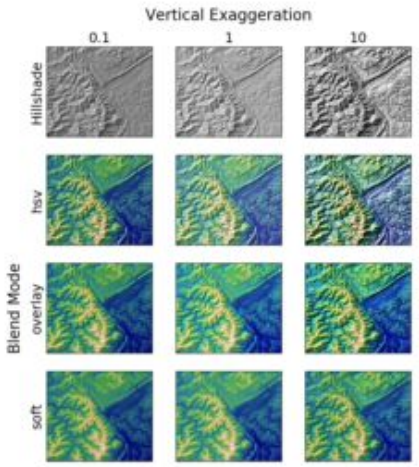
- Designed like MatLab: switching was easy
- Many rendering backends

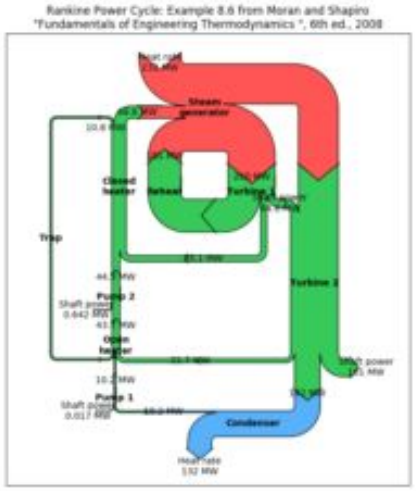
```
In [26]: from matplotlib import rcsetup
         rcsetup.all_backends

Out[26]: ['GTK',
          'GTKAgg',
          'GTKCairo',
          'MacOSX',
          'Qt4Agg',
          'Qt5Agg',
          'TkAgg',
          'WX',
          'WXAgg',
          'GTK3Cairo',
          'GTK3Agg',
          'WebAgg',
          'nbAgg',
          'agg',
          'cairo',
          'gdk',
          'pdf',
          'pgf',
          'ps',
          'svg',
          'template']
```

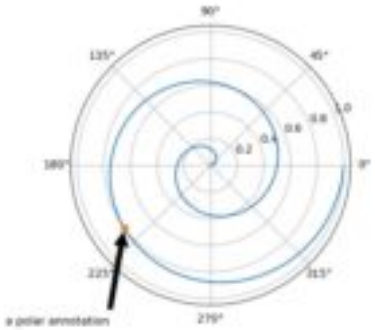
Plotting with Matplotlib

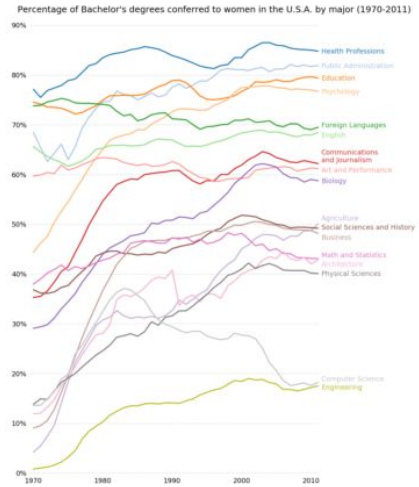
Strengths:

- Designed like MatLab: switching was easy
- Many rendering backends
- Can reproduce just about any plot (with a bit of effort)





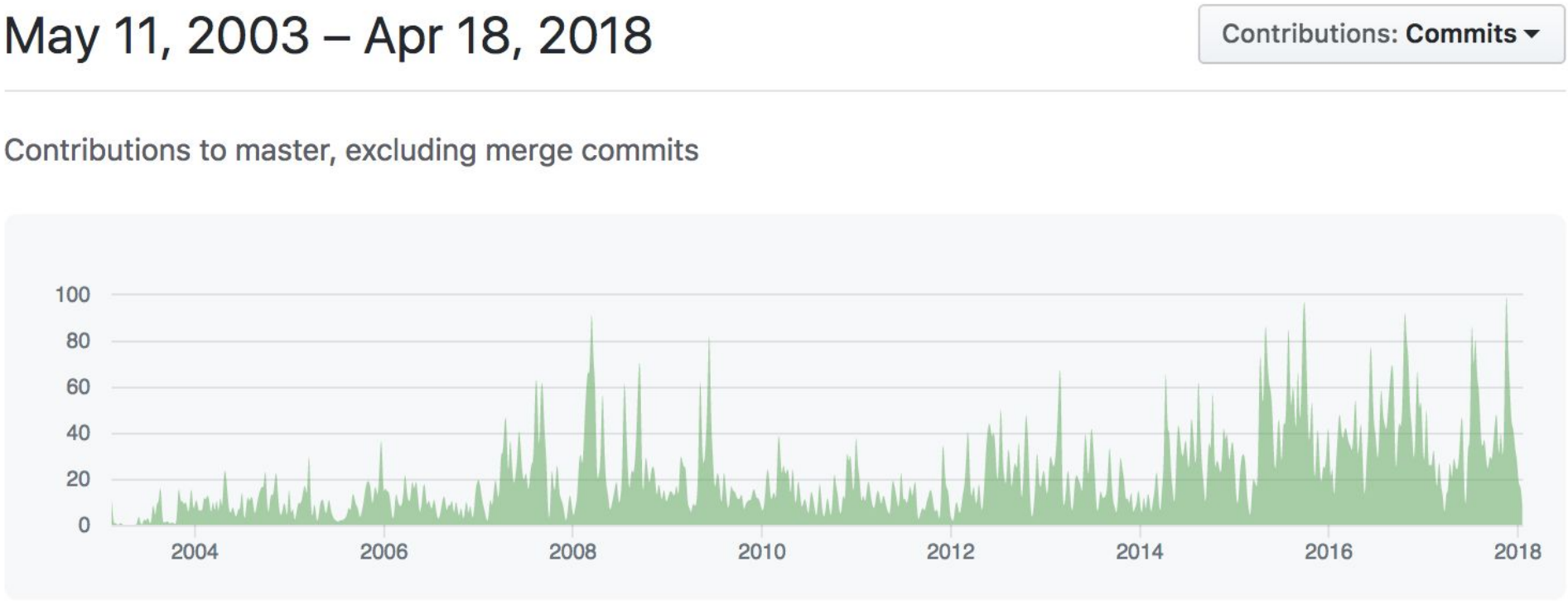




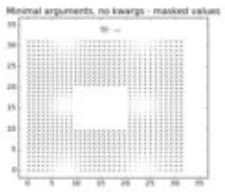
Plotting with Matplotlib

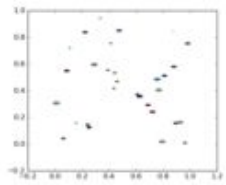
Strengths:

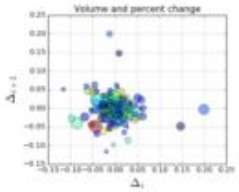
- Designed like MatLab: switching was easy
- Many rendering backends
- Can reproduce just about any plot (with a bit of effort)
- Well-tested, standard tool for 15 years

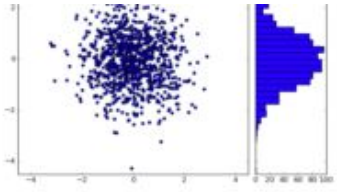


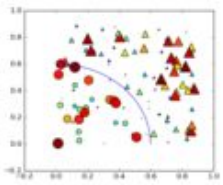
Matplotlib Gallery

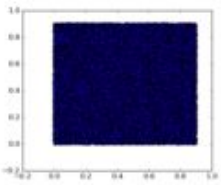

quiver_demo

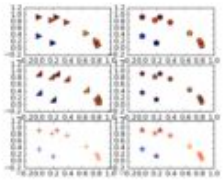

scatter_custom_symbol

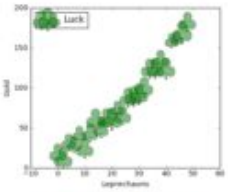

scatter_demo2

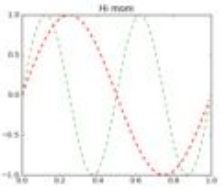

scatter_hist

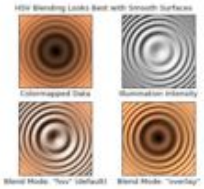

scatter_masked

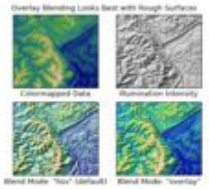

scatter_profile

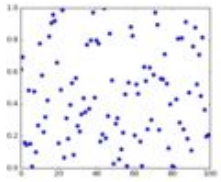

scatter_star_poly

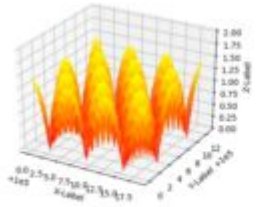

scatter_symbol

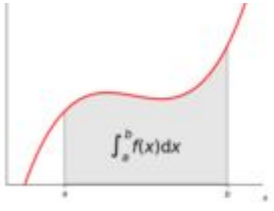

set_and_get


shading_example

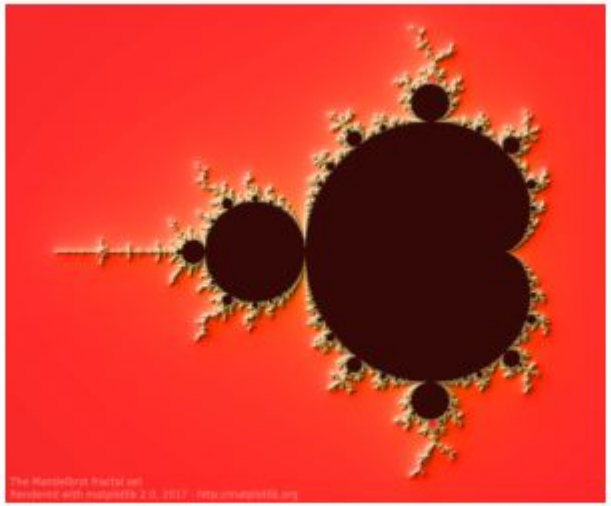

shading_example


shared_axis_across_figures

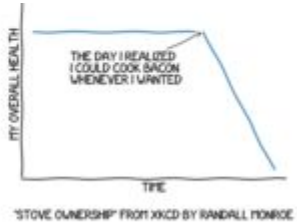

offset_demo



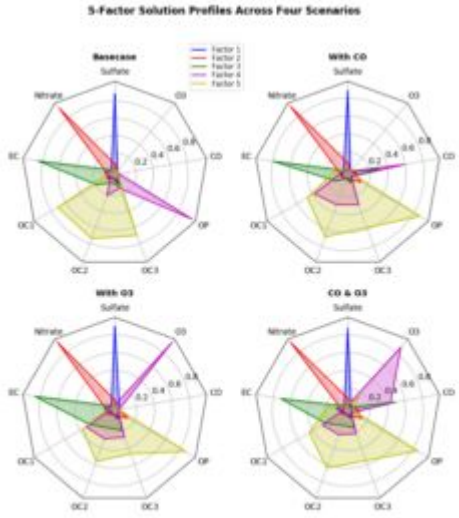
integral_demo


mandelbrot


svg_filter_pie



xkcd



radar_chart

Plotting with Matplotlib

Strengths:

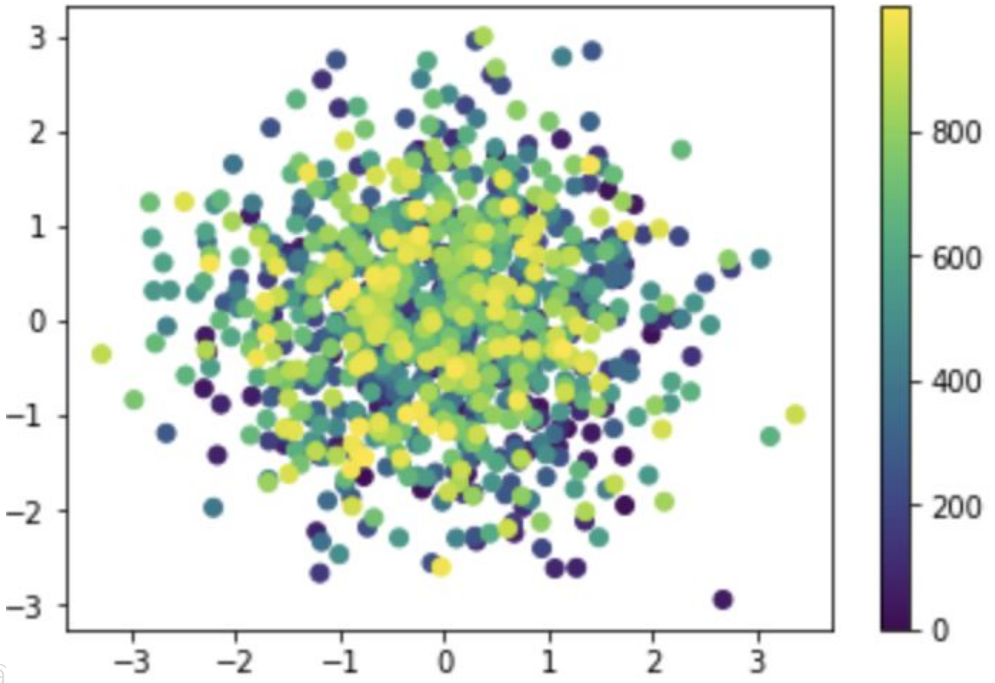
- Designed like MatLab: switching was easy
- Many rendering backends
- Can reproduce just about any plot with a bit of effort
- Well-tested, standard tool for 15 years

Weaknesses:

- API is imperative & often overly verbose
- Poor/no support for interactive/web graphs

Plotting with Matplotlib

```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.randn(1000)
y = np.random.randn(1000)
color = np.arange(1000)

plt.scatter(x, y, c=color)
plt.colorbar()
```



Statistical Visualization

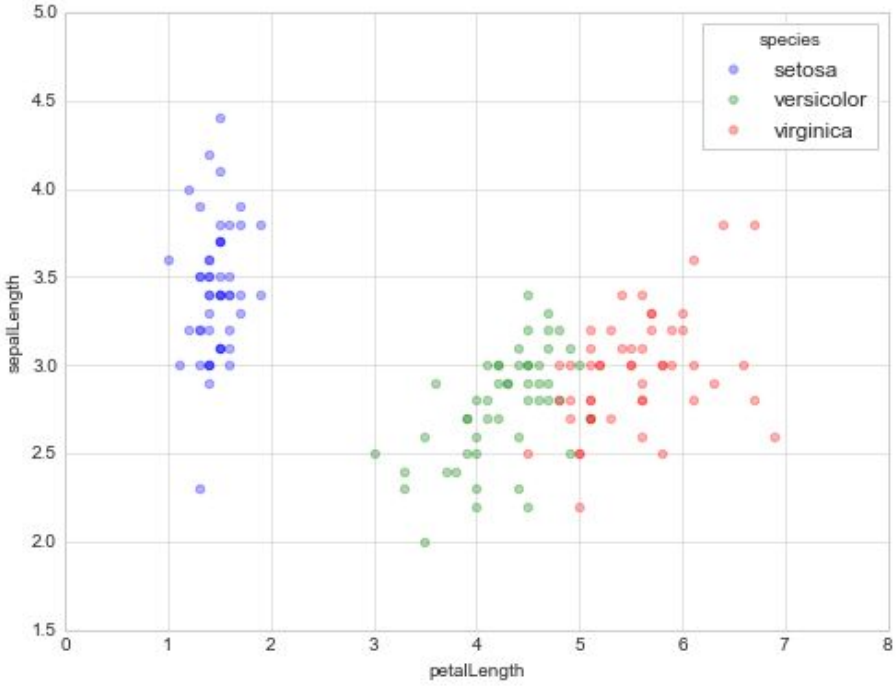
```
from vega datsets import data
iris = data('iris')
iris.head()
```

	petalLength	petalWidth	sepalLength	sepalWidth	species
0	1.4	0.2	5.1	3.5	setosa
1	1.4	0.2	4.9	3.0	setosa
2	1.3	0.2	4.7	3.2	setosa
3	1.5	0.2	4.6	3.1	setosa
4	1.4	0.2	5.0	3.6	setosa

Data in column-oriented format; i.e. rows are samples, columns are features

Statistical Visualization: Grouping

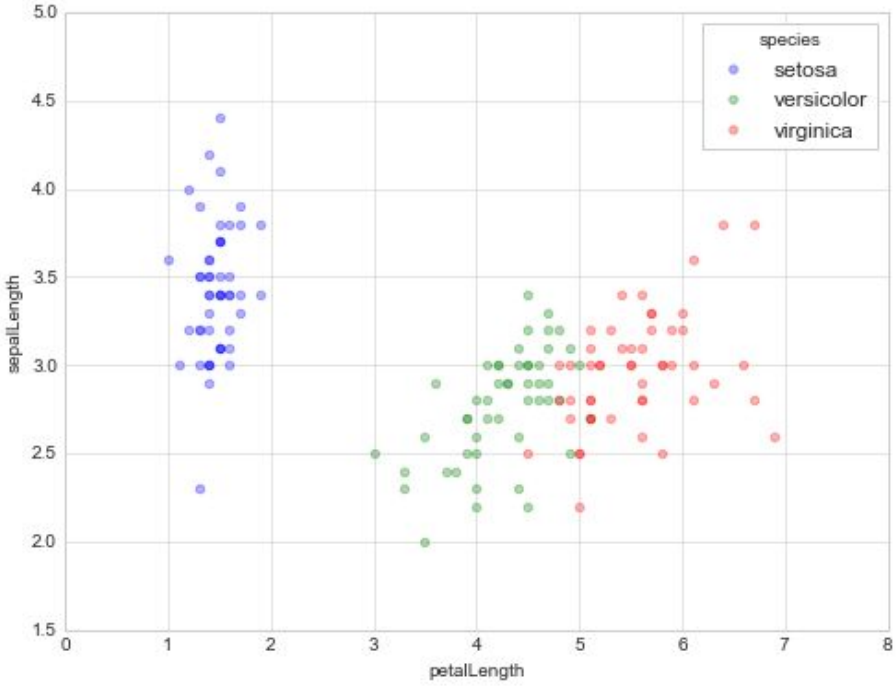
```
color map = dict(zip(iris.species.unique(),
                     ['blue', 'green', 'red']))

for species, group in iris.groupby('species'):
    plt.scatter(group['petalLength'], group['sepalWidth'],
                color=color map[species],
                alpha=0.3, edgecolor=None,
                label=species)

plt.legend(frameon=True, title='species')
plt.xlabel('petalLength')
plt.ylabel('sepalLength')
```



Statistical Visualization: Grouping

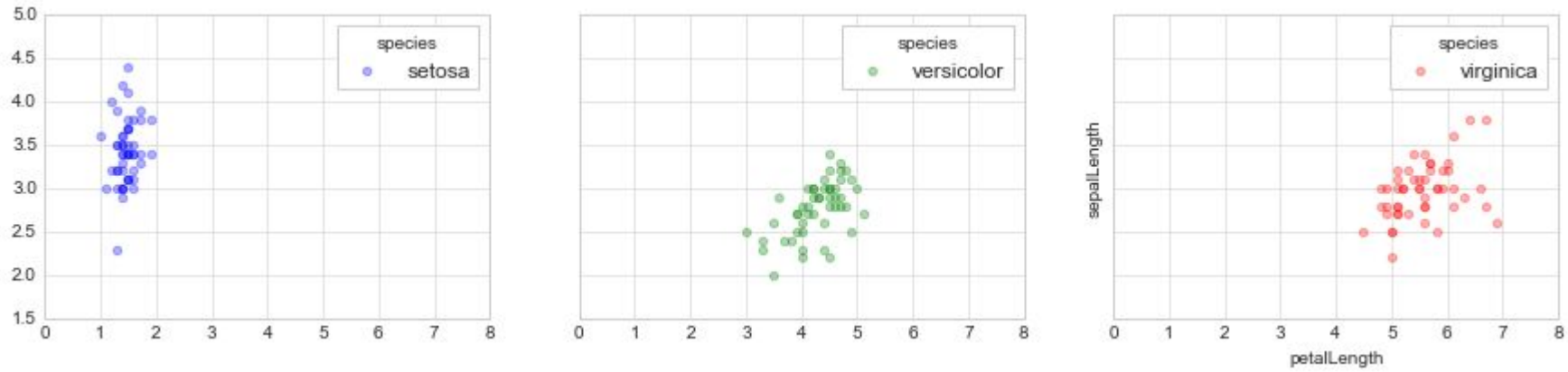
```
color map = dict(zip(iris.species.unique(),
                     ['blue', 'green', 'red']))

for species, group in iris.groupby('species'):
    plt.scatter(group['petalLength'], group['sepalWidth'],
                color=color map[species],
                alpha=0.3, edgecolor=None,
                label=species)

plt.legend(frameon=True, title='species')
plt.xlabel('petalLength')
plt.ylabel('sepalLength')
```

1. Data?
2. Transformation?
3. Marks?
4. Encoding?
5. Scale?
6. Guides?



Statistical Visualization: Faceting

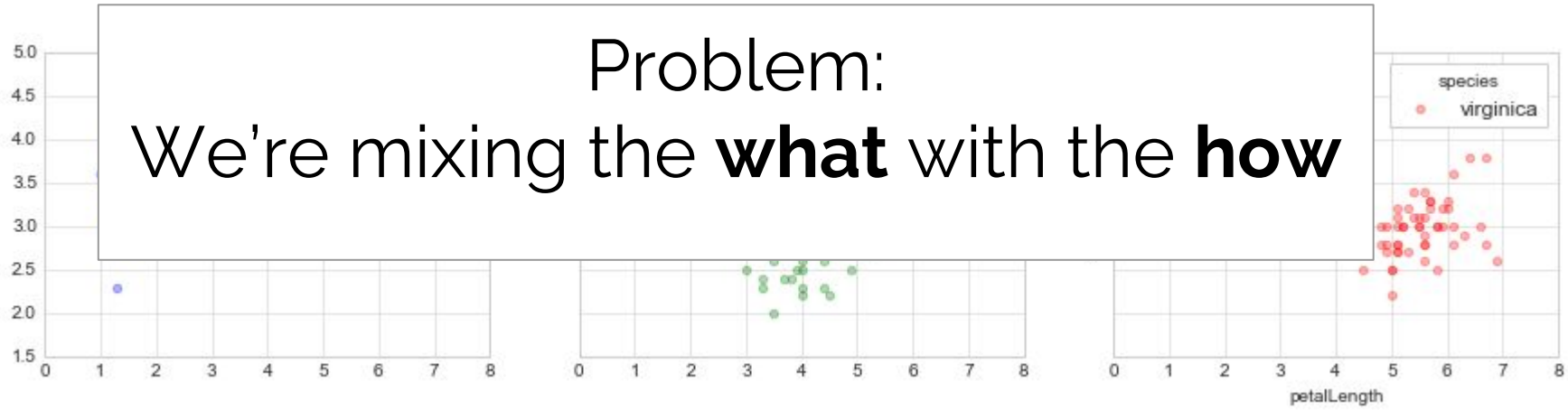
```
color_map = dict(zip(iris.species.unique(),['blue', 'green', 'red']))
n_panels = len(color_map)

fig, ax = plt.subplots(1, n_panels, figsize=(n_panels * 5, 3),
                       sharex=True, sharey=True)

for i, (species, group) in enumerate(iris.groupby('species')):
    ax[i].scatter(group['petalLength'], group['sepalWidth'],
                  color=color_map[species],
                  alpha=0.3, edgecolor=None,
                  label=species)
    ax[i].legend(frameon=True, title='species')

plt.xlabel('petalLength')
plt.ylabel('sepalLength')
```



Statistical Visualization: Faceting

```
color_map = dict(zip(iris.species.unique(),['blue', 'green', 'red']))
n_panels = len(color_map)

fig, ax = plt.subplots(1, n_panels, figsize=(n_panels * 5, 3),
                       sharex=True, sharey=True)

for i, (species, group) in enumerate(iris.groupby('species')):
    ax[i].scatter(group['petalLength'], group['sepalWidth'],
                  color=color_map[species],
                  alpha=0.3, edgecolor=None,
                  label=species)
    ax[i].legend(frameon=True, title='species')

plt.xlabel('petalLength')
plt.ylabel('sepalLength')
```

Problem:
We're mixing the **what** with the **how**



Toward a well-motivated *Declarative Visualization*

Imperative

- Specify *How* something should be done.
- Specification & Execution intertwined.
- *"Put a red circle here and a blue circle here"*

Declarative

- Specify *What* should be done.
- Separates Specification from Execution
- *"Map <x> to a position, and <y> to a color"*

Declarative visualization lets you think about **data** and **relationships**, rather than incidental details.

Toward a well-motivated *Declarative Visualization*

Imperative	**Declarative**
- Specify *How* something should be done.	- Specify *What* should be done.
- Specification & Execution intertwined.	- Separates Specification from Execution
- *"Put a red circle here and a blue circle here"*	- *"Map <x> to a position, and <y> to a color"*

Declarative visualization lets you think about **data** and **relationships**, rather than incidental details.

Altair

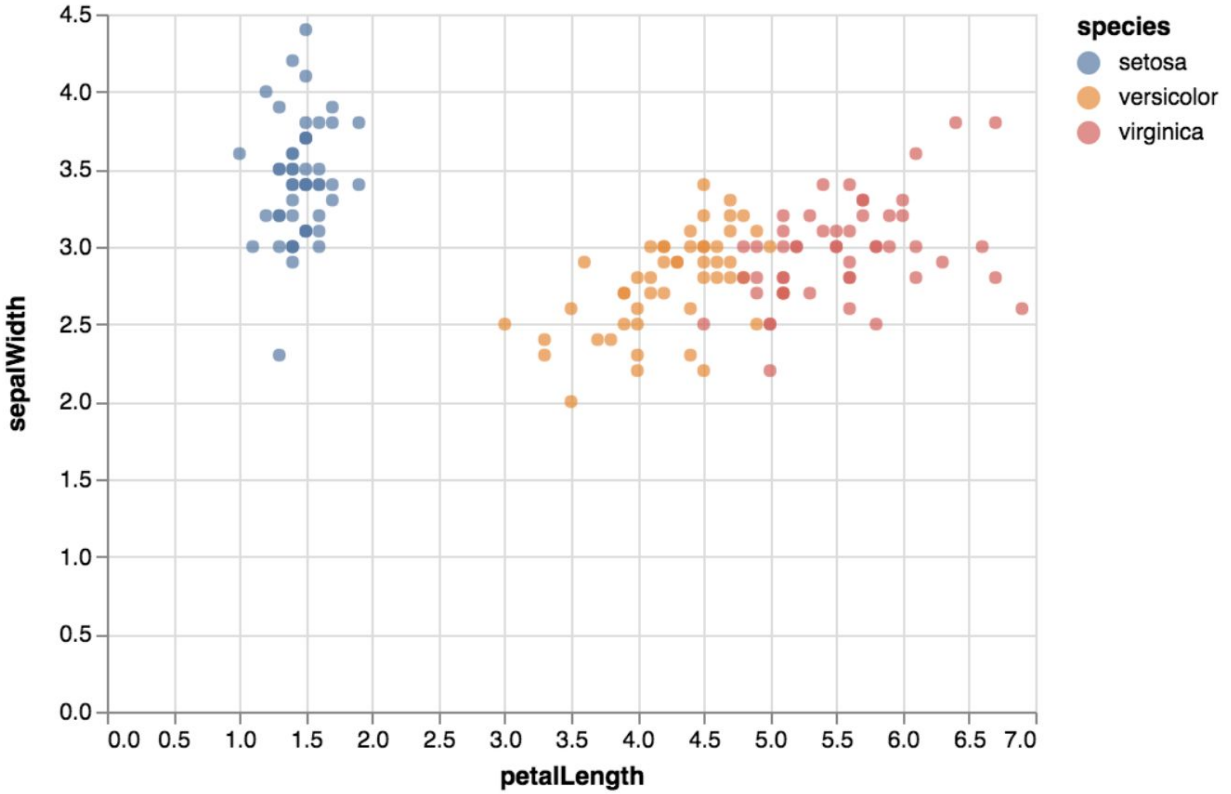
Declarative Visualization in Python

Based on the Vega and Vega-Lite grammars.

http://altair-viz.github.io

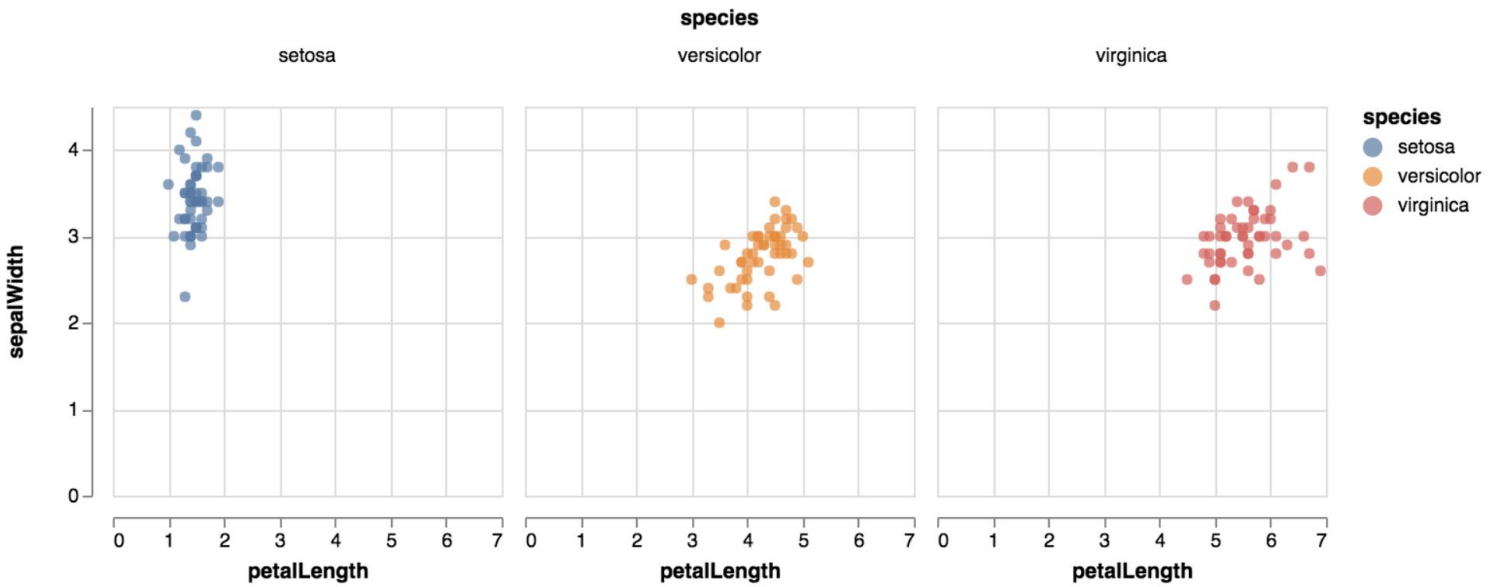
Altair for Statistical Visualization

```
import altair as alt
from vega_datasets import data

iris = data.iris()

alt.Chart(iris).mark_point().encode(
    x='petalLength',
    y='sepalWidth',
    color='species'
)
```



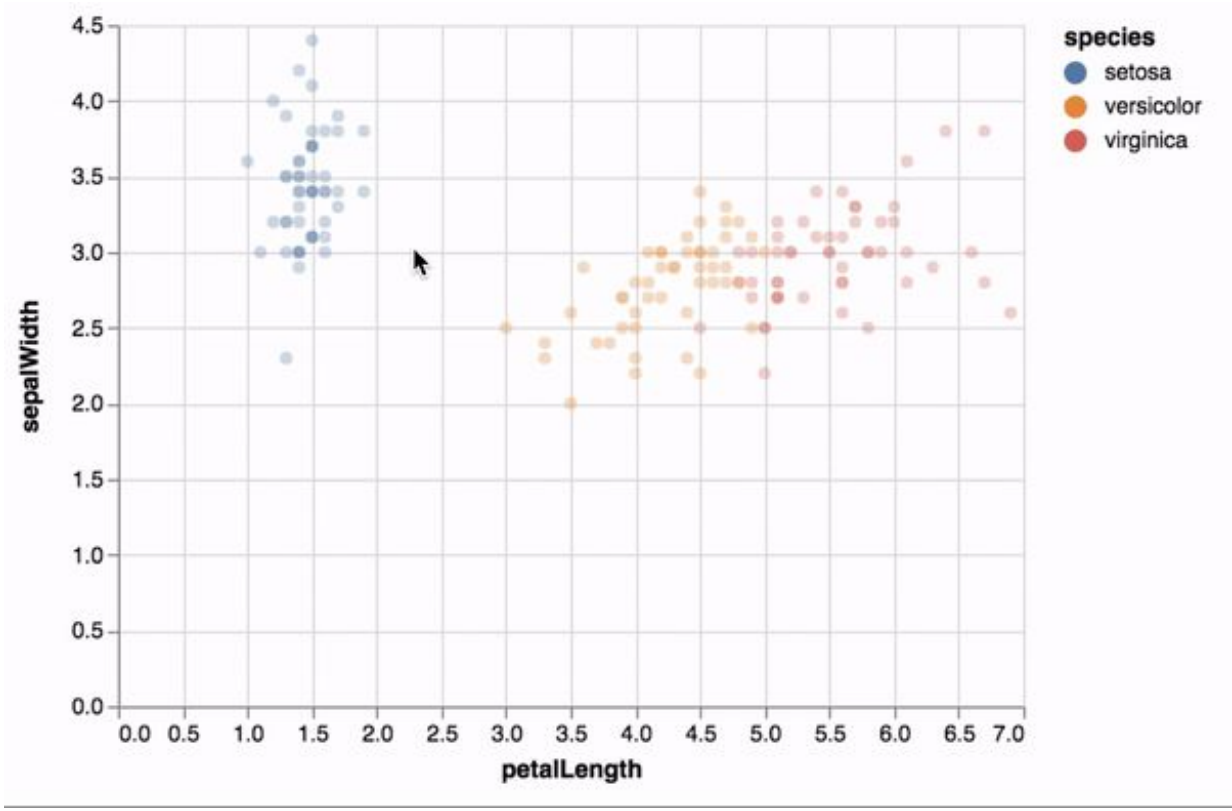
Encodings are Flexible:

```
import altair as alt
from vega_datasets import data

iris = data.iris()

alt.Chart(iris).mark_point().encode(
    x='petalLength',
    y='sepalWidth',
    color='species',
    column='species'
)
```



Altair is Interactive

```
import altair as alt
from vega_datasets import data

iris = data.iris()

alt.Chart(iris).mark_point().encode(
    x='petalLength',
    y='sepalWidth',
    color='species'
).interactive()
```



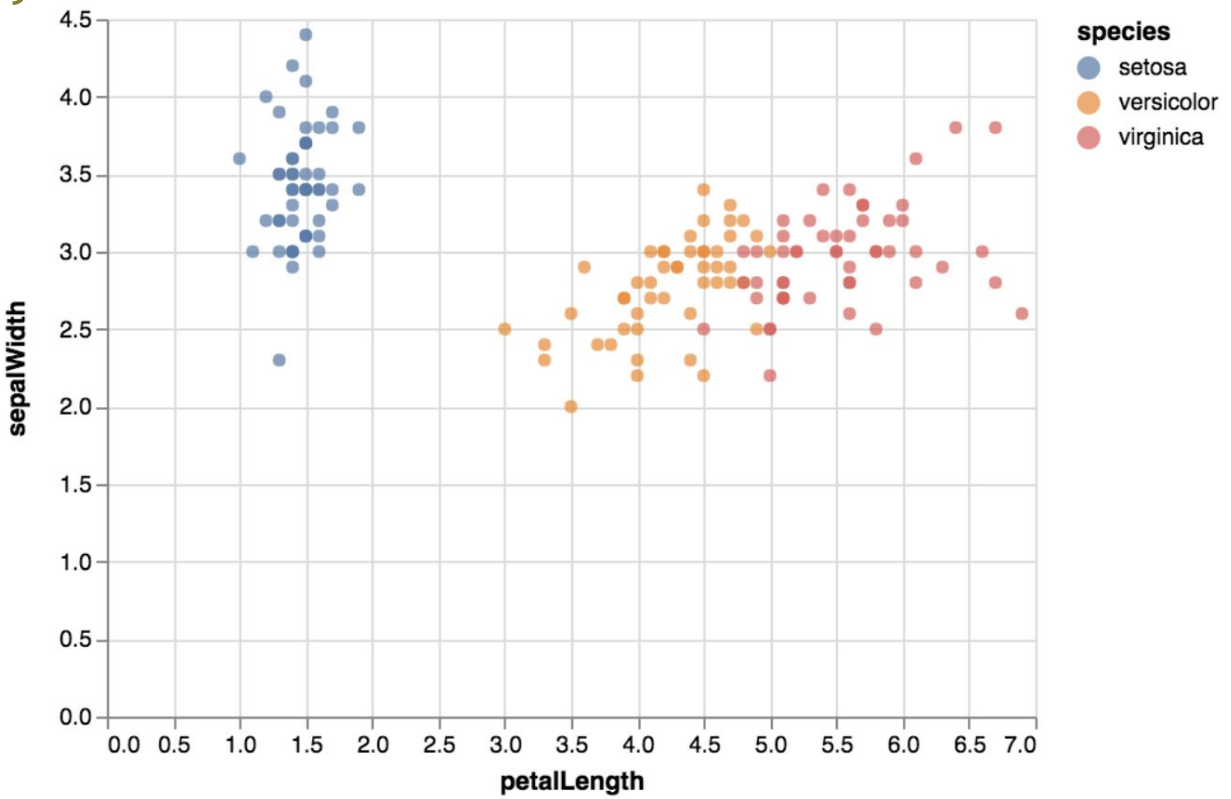
And so much more . . .

See the rest of the tutorial content at
http://github.com/altair-viz/altair-tutorial

Extra Content

Basics of an Altair Chart

```
import altair as alt
from vega_datasets import data

iris = data.iris()

alt.Chart(iris).mark_point().encode(
    x='petalLength:Q',
    y='sepalWidth:Q',
    color='species:N'
)
```



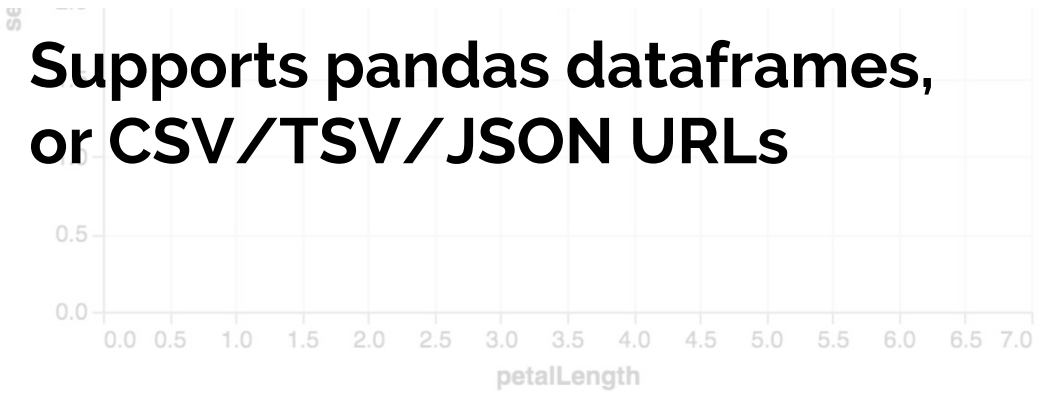
Anatomy of an Altair Chart

```
import altair as Chart
from vega_datasets import data

iris = data.iris()

alt.Chart(iris).mar
    x='petalLength:
    y='sepalWidth:Q
    color='species:
)
```

Chart assumes tabular, column-oriented data

	petalLength	petalWidth	sepalLength	sepalWidth	species
0	1.4	0.2	5.1	3.5	setosa
1	1.4	0.2	4.9	3.0	setosa
2	1.3	0.2	4.7	3.2	setosa
3	1.5	0.2	4.6	3.1	setosa
4	1.4	0.2	5.0	3.6	setosa

Supports pandas dataframes, or CSV/TSV/JSON URLs

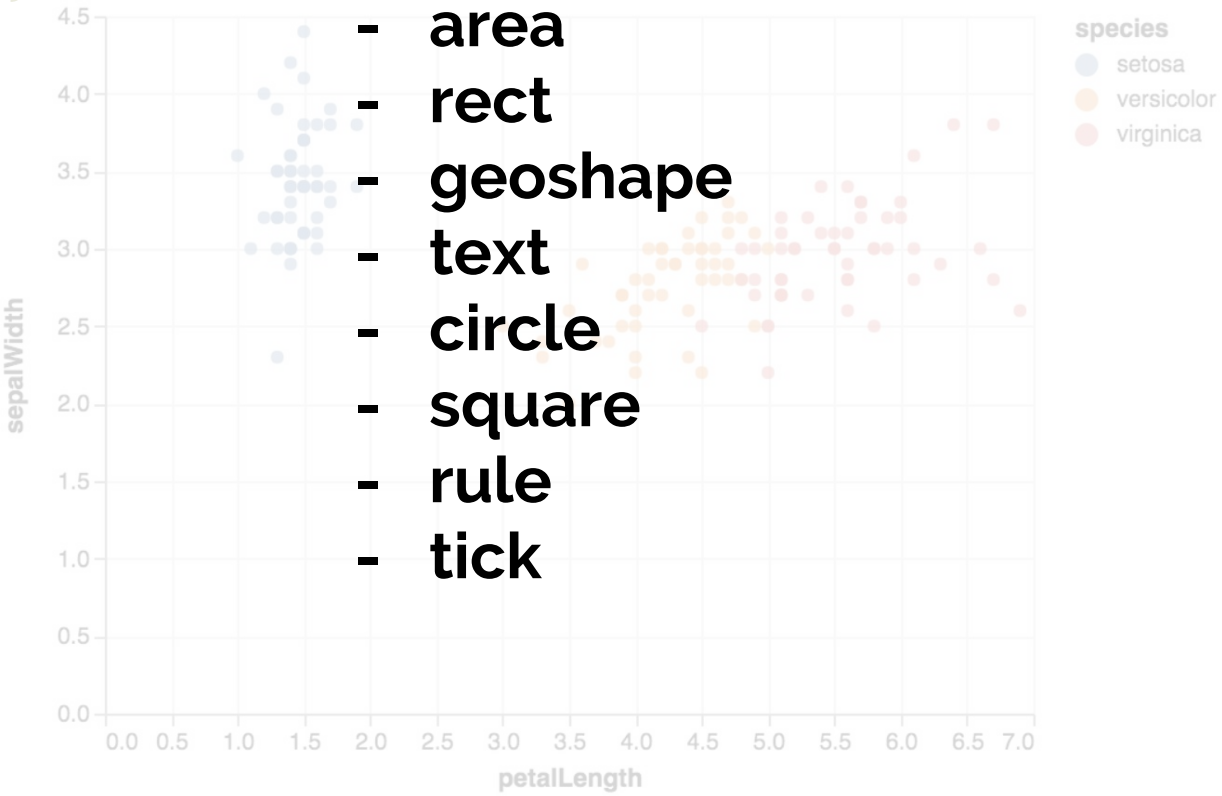


Anatomy of an Altair Chart

```
import altair as Chart
from vega_datasets import data

iris = data.iris()

alt.Chart(iris).mark_point().encode(
    x='petalLength:Q',
    y='sepalWidth:Q',
    color='species:N'
)
```

Chart uses one of several pre-defined marks:

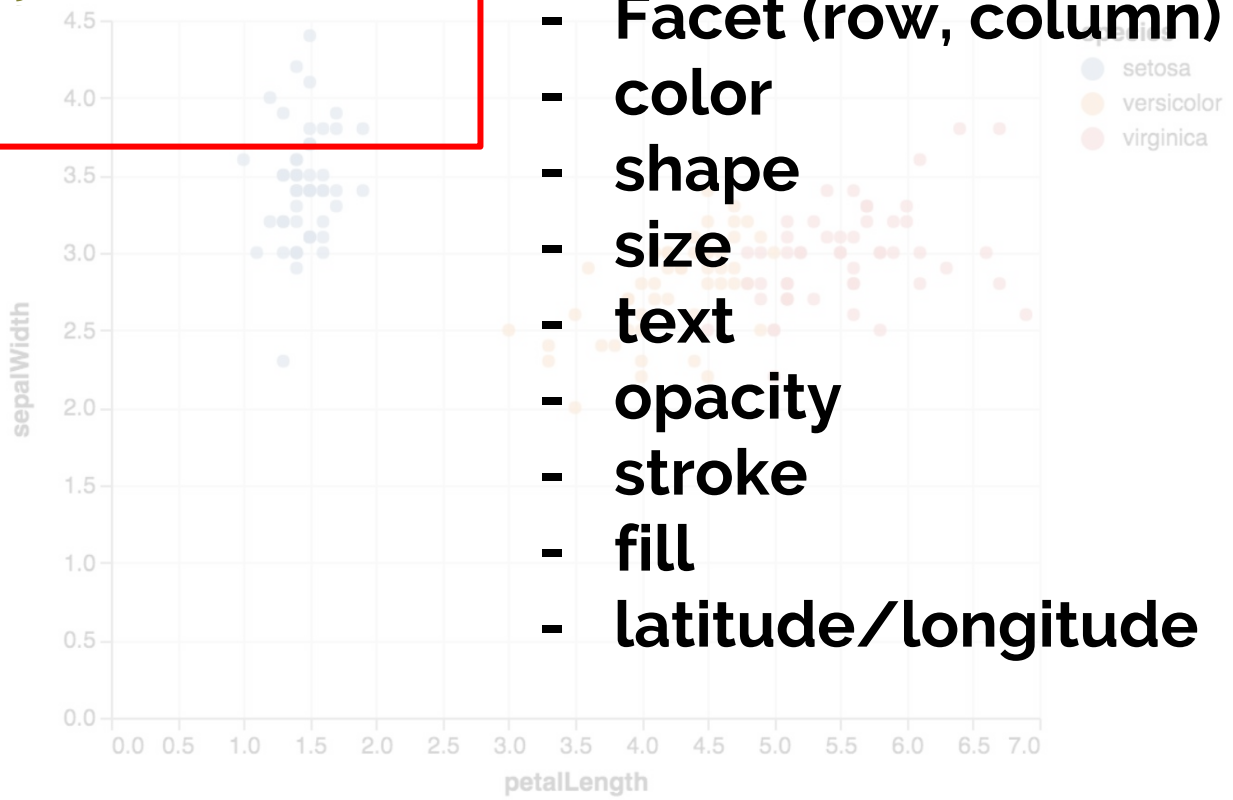
- point
- line
- bar
- area
- rect
- geoshape
- text
- circle
- square
- rule
- tick



Basics of an Altair Chart

- Encodings map *visual channels* to *data columns*,
- Channels are automatically adjusted based on data type (N, O, Q, T)

```
alt.Chart(iris).mark_point().encode(
    x='petalLength:Q',
    y='sepalWidth:Q',
    color='species:N'
)
```



Available channels:

- Position (x, y)
- Facet (row, column)
- color
- shape
- size
- text
- opacity
- stroke
- fill
- latitude/longitude

Anatomy of an Altair Chart

```
import altair as alt
from vega_datasets import data

iris = data.iris()

alt.Chart(iris).mark_point().encode(
    x='petalLength:Q',
    y='sepalWidth:Q',
    color='species:N'
).to_json()
```

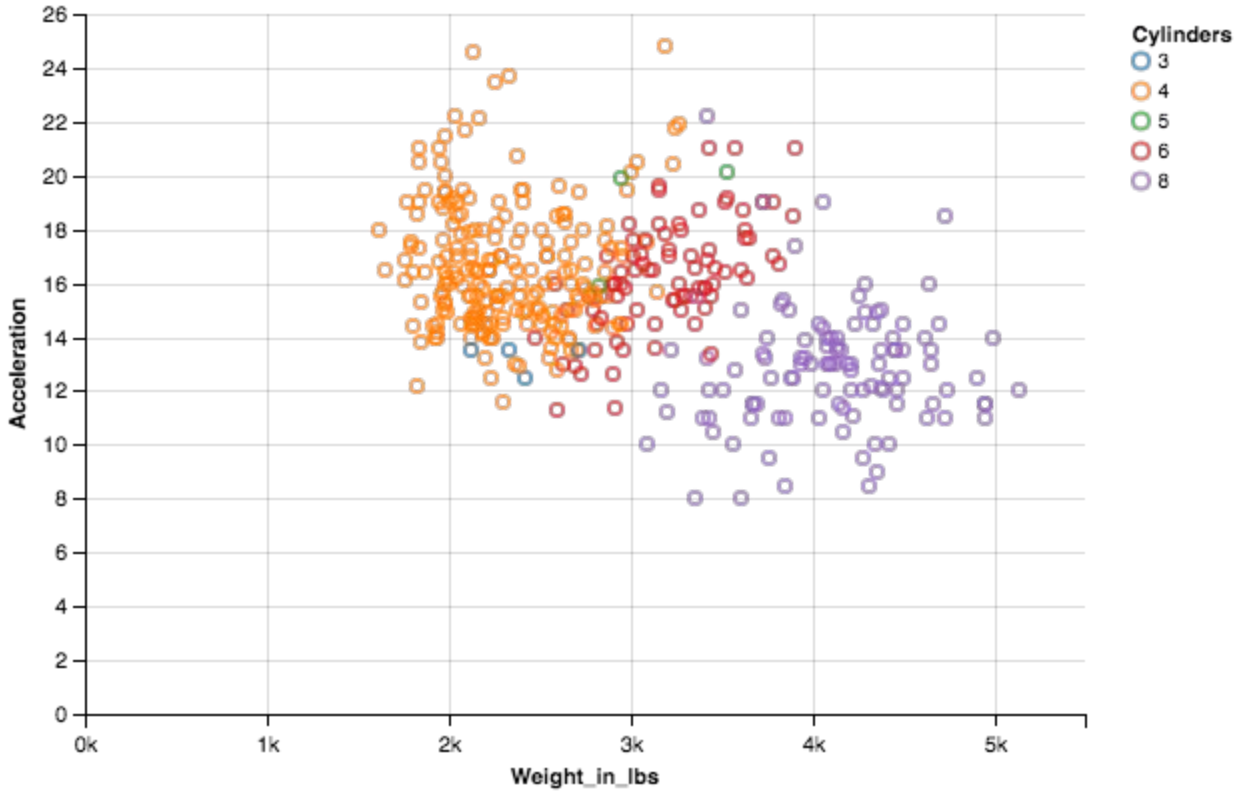
Altair produces specifications following the *Vega-Lite* grammar.

```
{ "data": {"values": [...]},
  "encoding": {
    "color": {"field": "species", "type": "nominal"},
    "x": {"field": "petalLength", "type": "quantitative"},
    "y": {"field": "sepalWidth", "type": "quantitative"}
  },
  "mark": "point"
}
```

http://vega.github.io/vega-lite/

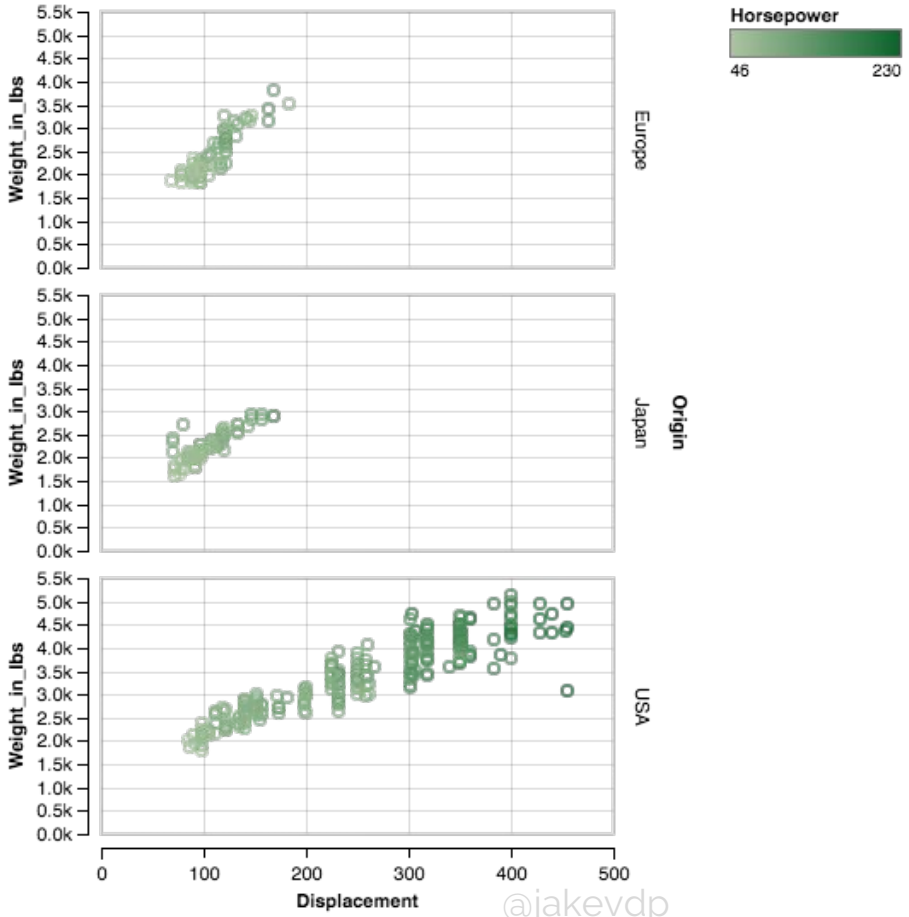
Examples:

```
In [4]: Chart(cars).mark_point().encode(
            x='Weight_in_lbs',
            y='Acceleration',
            color='Cylinders:N'
        )
```



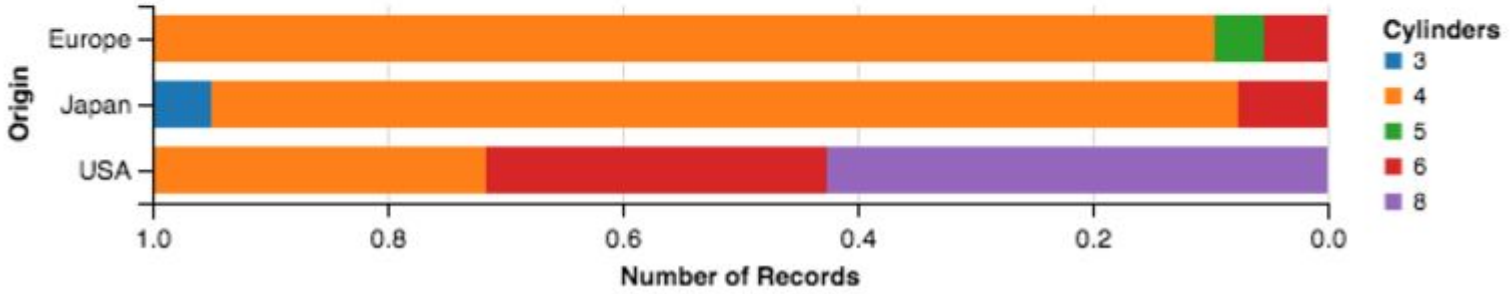
Examples:

```
In [6]: Chart(cars).mark_point().encode(
            x='Displacement',
            y='Weight_in_lbs',
            color='Horsepower',
            row='Origin'
        ).configure_cell(width=300, height=150)
```



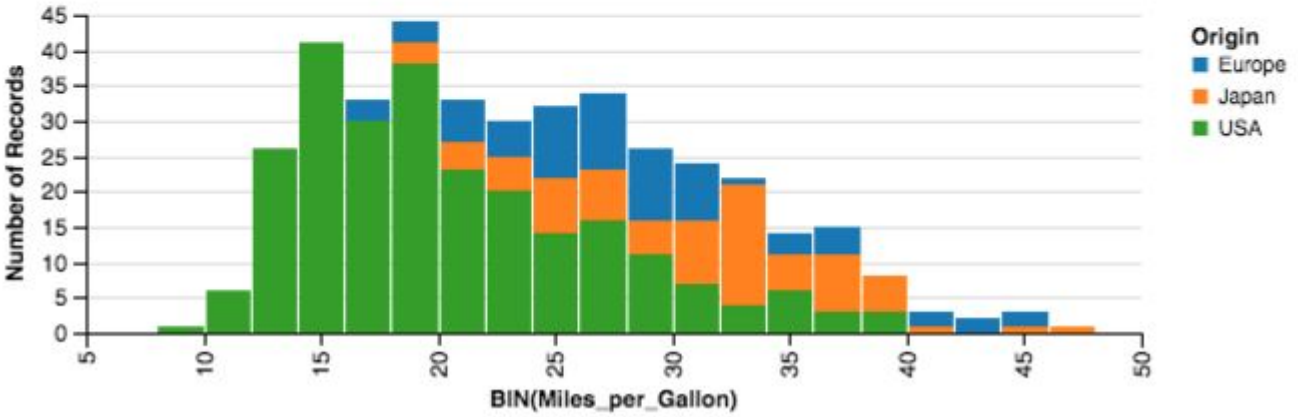
Examples:

In [7]:
```
Chart(cars).mark_bar(stacked='normalize').encode(
    Y('Origin'),
    X('*:Q', aggregate='count', sort='descending'),
    Color('Cylinders:N')
)
```



Examples:

In [8]:
```
Chart(cars).mark_bar().encode(
    X('Miles_per_Gallon', bin=Bin(maxbins=20)),
    Y('*:Q', aggregate='count'),
    Color('Origin')
).configure_cell(height=150)
```

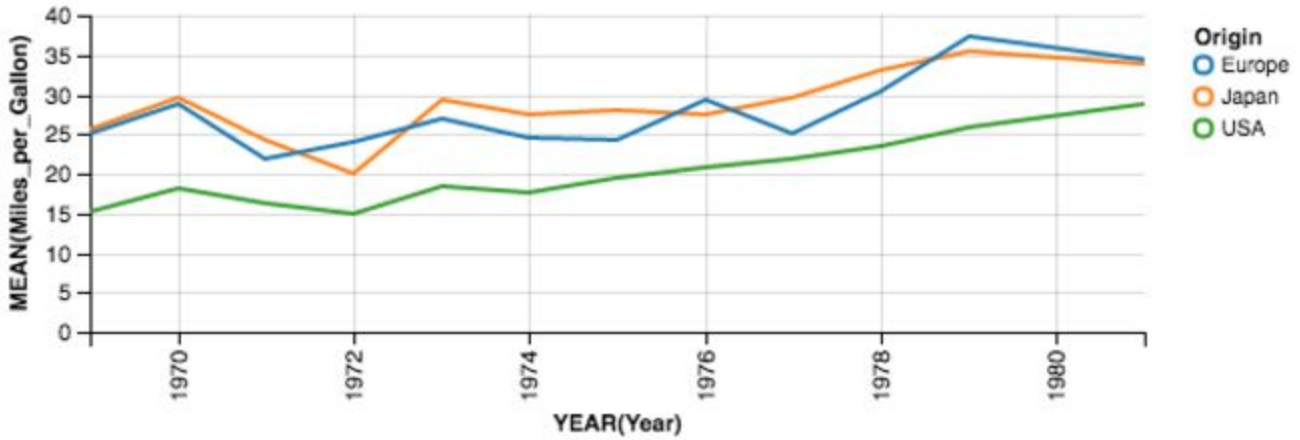


Examples:

```
In [9]: Chart(cars).mark_text(applyColorToBackground=True).encode(
            Row('Origin:O'),
            Column('Cylinders:O'),
            Color('mean(Miles_per_Gallon):Q', sort='descending'),
            Text('mean(Miles_per_Gallon):Q')
        )
```

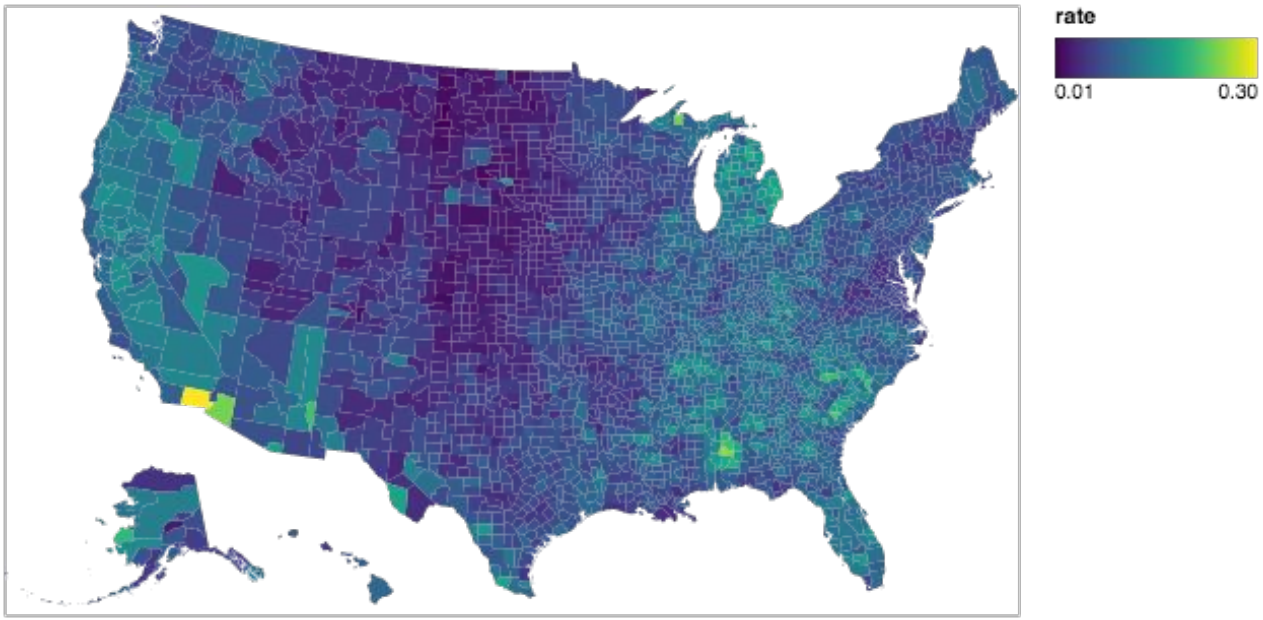
Examples:

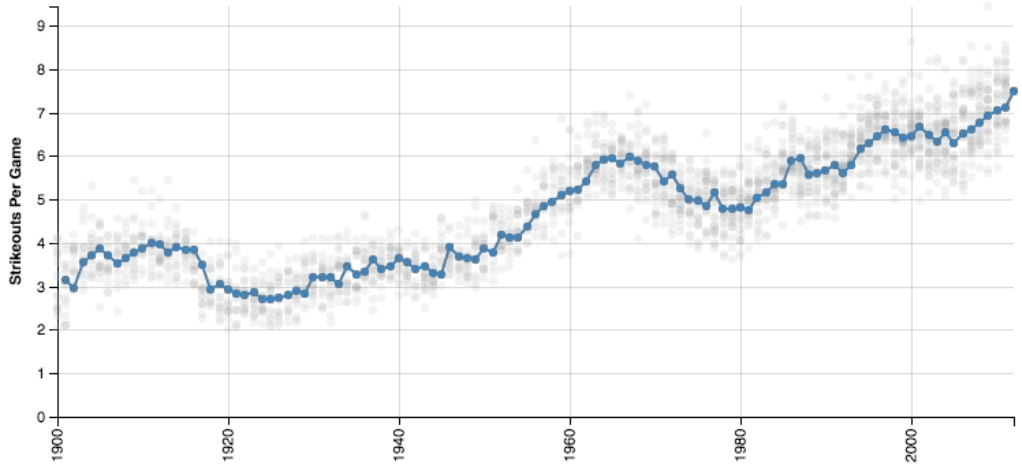
In [10]:
```
Chart(cars).mark_line().encode(
    X('Year:T', timeUnit='year'),
    Y('Miles_per_Gallon:Q', aggregate='mean'),
    Color('Origin:N')
).configure_cell(height=150)
```

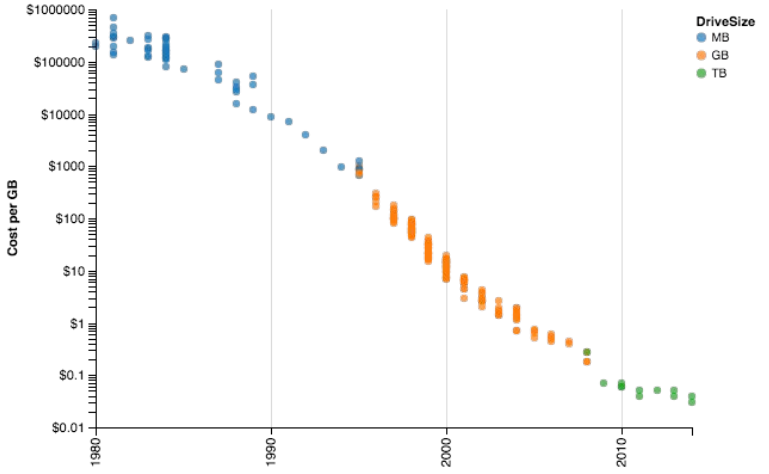


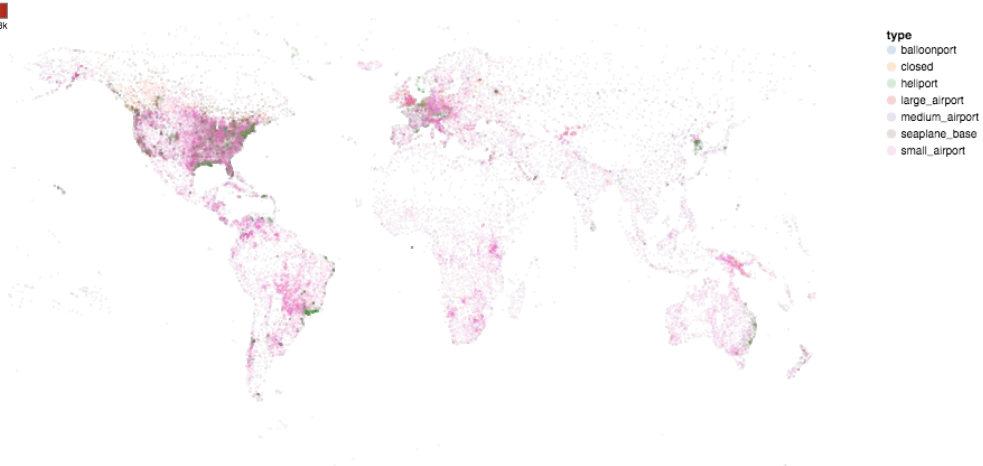
Examples:

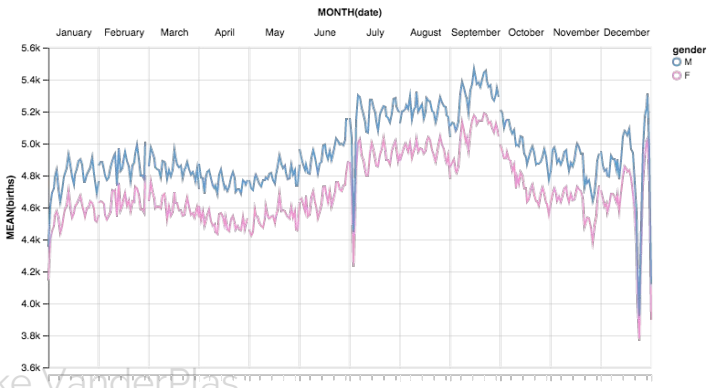
In [11]:
```
counties = alt.topo_feature(data.us_10m.url, 'counties')
unemp_data = data.unemployment.url

alt.Chart(counties).mark_geoshape().properties(
    projection={'type': 'albersUsa'},
    width=500, height=300
).encode(
    color='rate:Q'
).transform_lookup(
    lookup='id', from_=alt.LookupData(unemp_data, 'id', ['rate'])
)
```

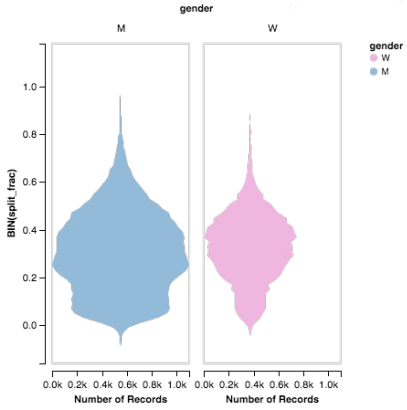






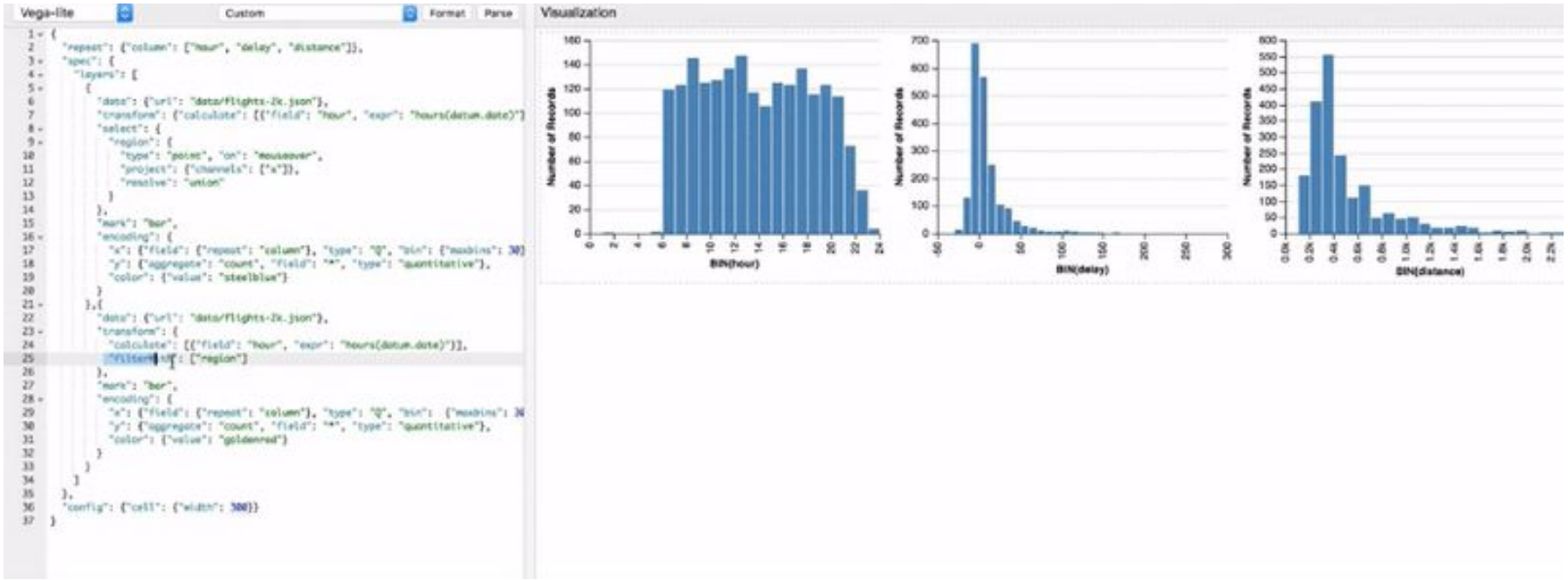






(Visualizations from jakevdp/altair-examples).

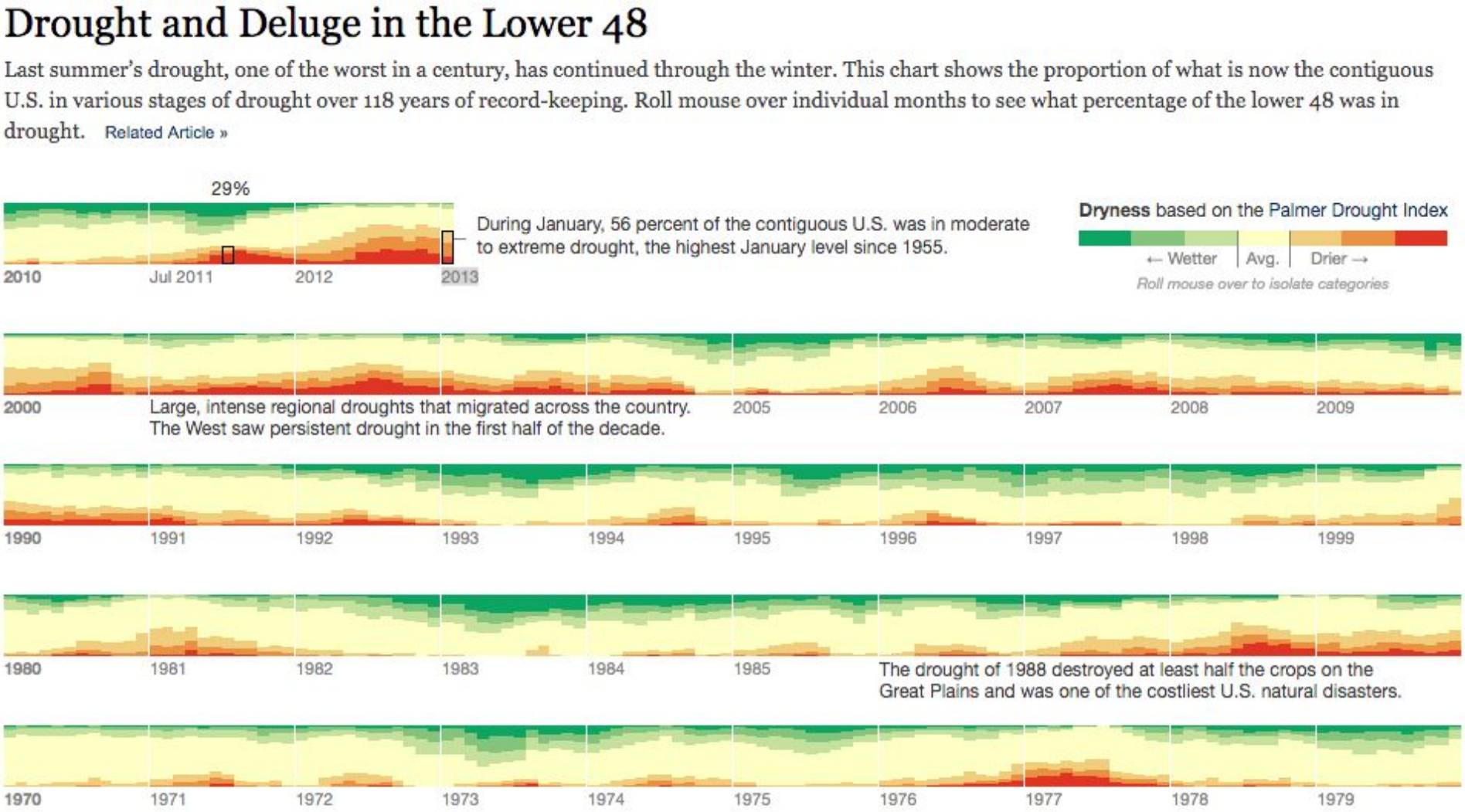
Altair 2.0: a Grammar of Interaction



~ From D3 to Vega to Altair ~

So what is Vega-Lite?

D3 is Everywhere . . .

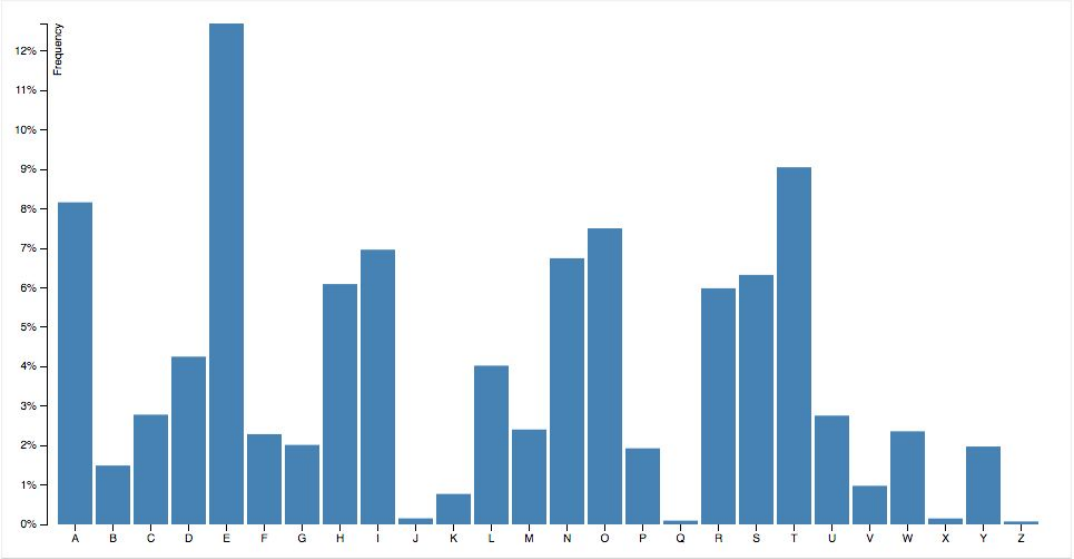


(live version at NYT)

But working in D3 can be challenging . . .

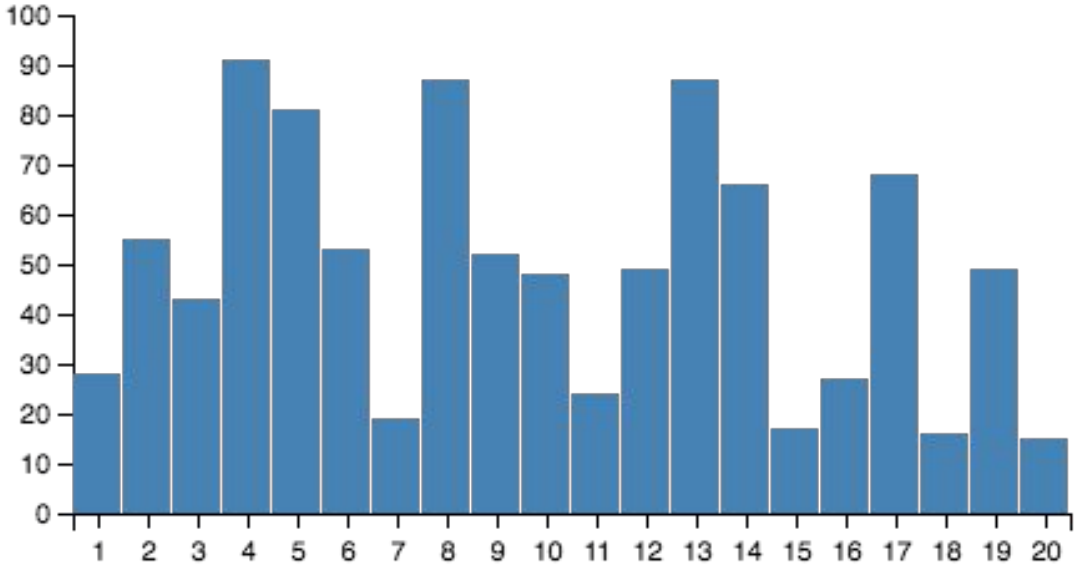
```
var margin = {top: 20, right: 20, bottom: 30, left: 40},
    width = 960 - margin.left - margin.right,
    height = 500 - margin.top - margin.bottom;

var x = d3.scale.ordinal()
    .rangeRoundBands([0, width], .1);

var y = d3.scale.linear()
    .range([height, 0]);

var xAxis = d3.svg.axis()
    .scale(x)
    .orient("bottom");

var yAxis = d3.svg.axis()
    .scale(y)
    .orient("left")
    .ticks(10, "%");

var svg = d3.select("body").append("svg")
    .attr("width", width + margin.left + margin.right)
    .attr("height", height + margin.top + margin.bottom)
  .append("g")
    .attr("transform", "translate(" + margin.left + "," + margin.top + ")");

d3.tsv("data.tsv", type, function(error, data) {
  if (error) throw error;

  x.domain(data.map(function(d) { return d.letter; }));
  y.domain([0, d3.max(data, function(d) { return d.frequency; })]);

  svg.append("g")
      .attr("class", "x axis")
      .attr("transform", "translate(0," + height + ")")
      .call(xAxis);

  svg.append("g")
      .attr("class", "y axis")
      .call(yAxis)
    .append("text")
      .attr("transform", "rotate(-90)")
      .attr("y", 6)
      .attr("dy", ".71em")
      .style("text-anchor", "end")
      .text("Frequency");

  svg.selectAll(".bar")
      .data(data)
    .enter().append("rect")
      .attr("class", "bar")
      .attr("x", function(d) { return x(d.letter); })
      .attr("width", x.rangeBand())
      .attr("y", function(d) { return y(d.frequency); })
      .attr("height", function(d) { return height - y(d.frequency); });
});

function type(d) {
  d.frequency = +d.frequency;
  return d;
}
```

Bar Chart: d3



D3 is a Javascript package that streamlines manipulation of objects on a webpage.

Bar Chart: Vega

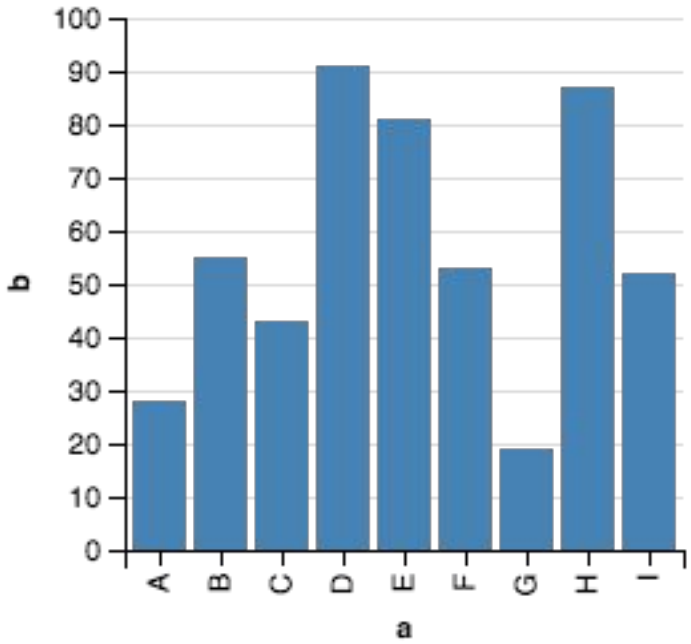


Vega is a detailed declarative specification for visualizations, built on D3.

```
{
  "width": 400,
  "height": 200,
  "padding": {"top": 10, "left": 30, "bottom": 30, "right": 10},
  "data": [
    {
      "name": "table",
      "values": [
        {"x": 1,  "y": 28}, {"x": 2,  "y": 55},
        {"x": 3,  "y": 43}, {"x": 4,  "y": 91},
        {"x": 5,  "y": 81}, {"x": 6,  "y": 53},
        {"x": 7,  "y": 19}, {"x": 8,  "y": 87},
        {"x": 9,  "y": 52}, {"x": 10, "y": 48},
        {"x": 11, "y": 24}, {"x": 12, "y": 49},
        {"x": 13, "y": 87}, {"x": 14, "y": 66},
        {"x": 15, "y": 17}, {"x": 16, "y": 27},
        {"x": 17, "y": 68}, {"x": 18, "y": 16},
        {"x": 19, "y": 49}, {"x": 20, "y": 15}
      ]
    }
  ],
  "scales": [
    {
      "name": "x",
      "type": "ordinal",
      "range": "width",
      "domain": {"data": "table", "field": "x"}
    },
    {
      "name": "y",
      "type": "linear",
      "range": "height",
      "domain": {"data": "table", "field": "y"},
      "nice": true
    }
  ],
  "axes": [
    {"type": "x", "scale": "x"},
    {"type": "y", "scale": "y"}
  ],
  "marks": [
    {
      "type": "rect",
      "from": {"data": "table"},
      "properties": {
        "enter": {
          "x": {"scale": "x", "field": "x"},
          "width": {"scale": "x", "band": true, "offset": -1},
          "y": {"scale": "y", "field": "y"},
          "y2": {"scale": "y", "value": 0}
        },
        "update": {
          "fill": {"value": "steelblue"}
        }
```

```
{
  "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": "ordinal"},
    "y": {"field": "b", "type": "quantitative"}
  }
}
```

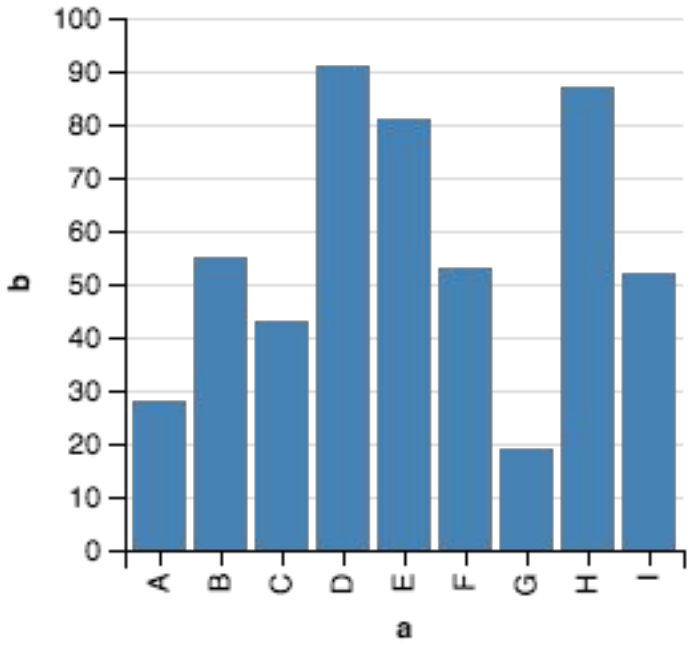
Bar Chart: Vega-Lite



Vega-Lite is a simpler declarative specification aimed at statistical visualization.

Bar Chart: Altair

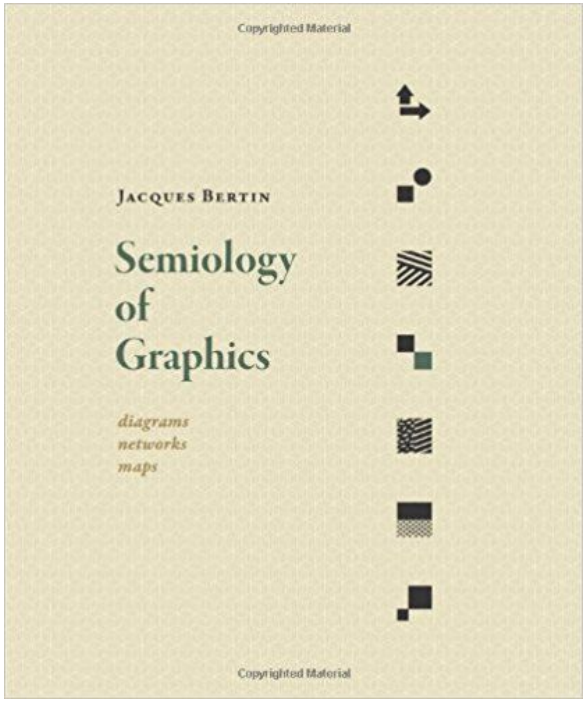
```
import pandas as pd
from altair import Chart

data = pd.DataFrame({'a': ['A', 'B', 'C', 'D', 'E',
                           'F', 'G', 'H', 'I'],
                     'b': [28, 55, 43, 91, 81,
                           53, 19, 87, 52]})

Chart(data).mark_bar().encode(
    x='a',
    y='b',
)
```

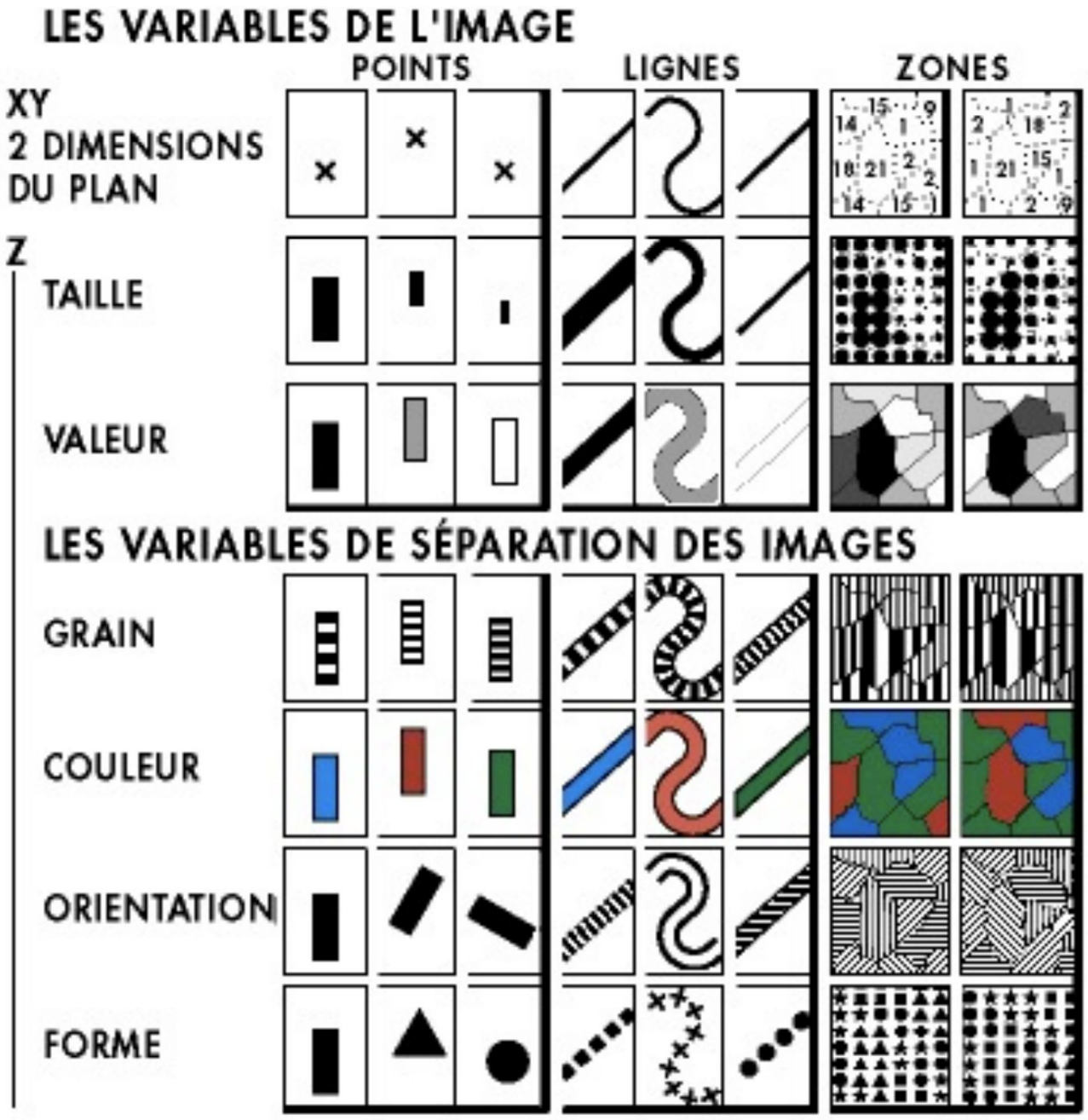


Altair is a Python API for creating Vega-Lite specifications.

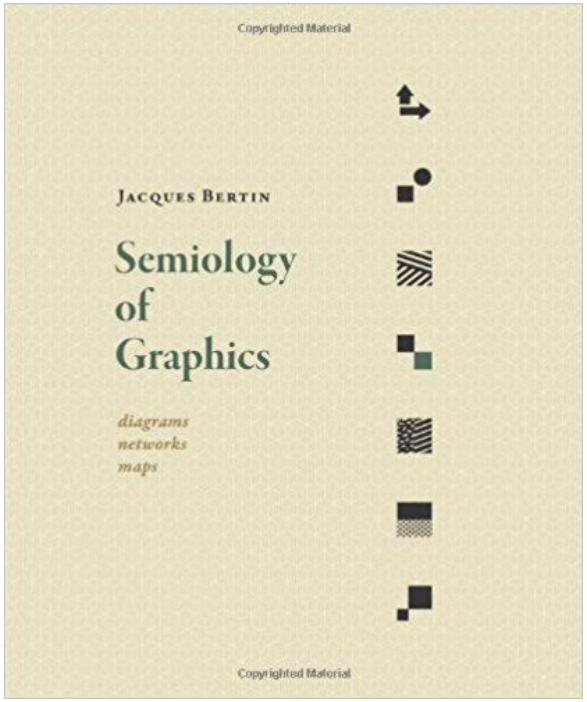
~ Thinking about Visualization ~

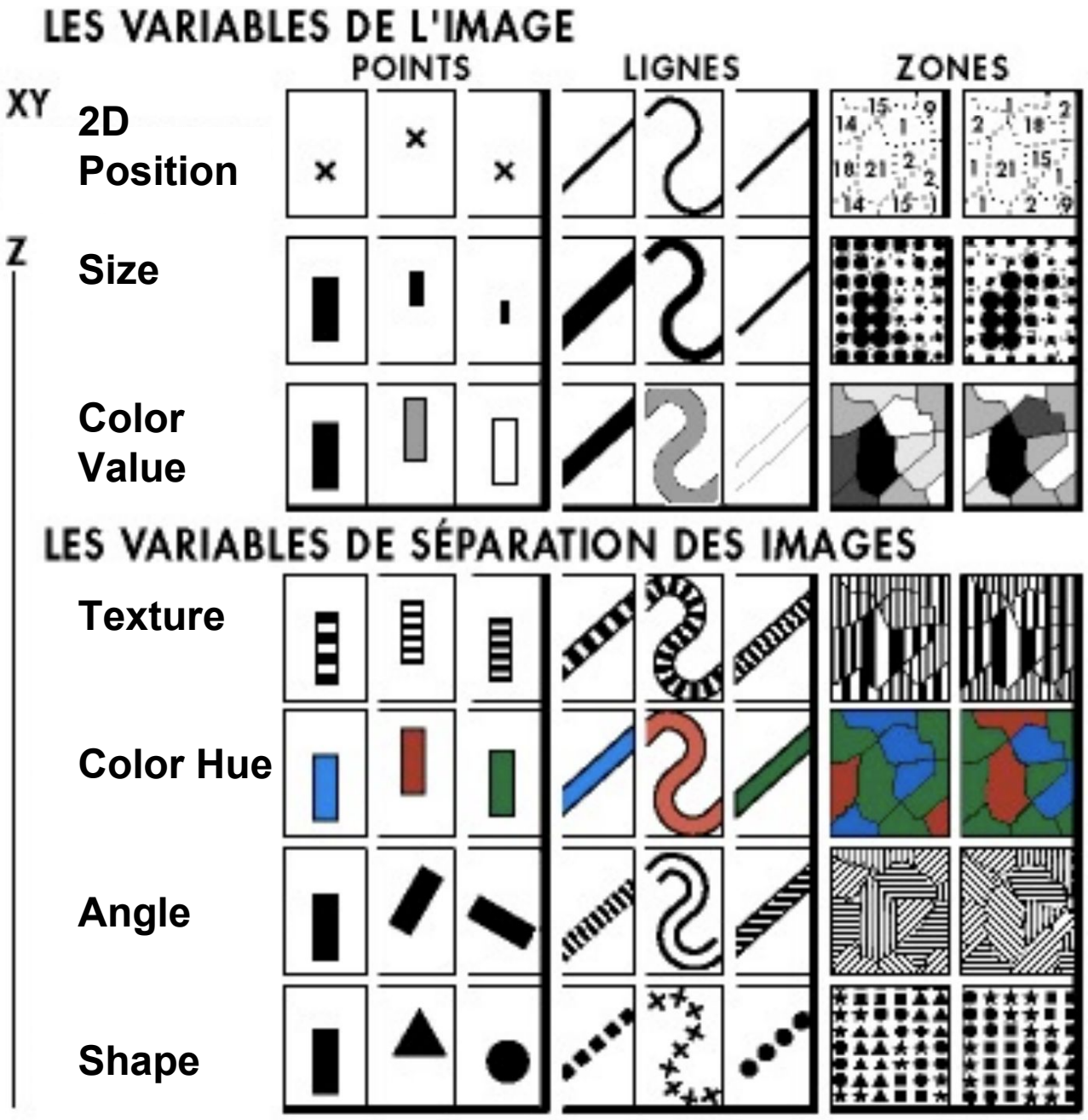
Bertin's Semiology of Graphics (1967)





Bertin's Semiology of Graphics (1967)

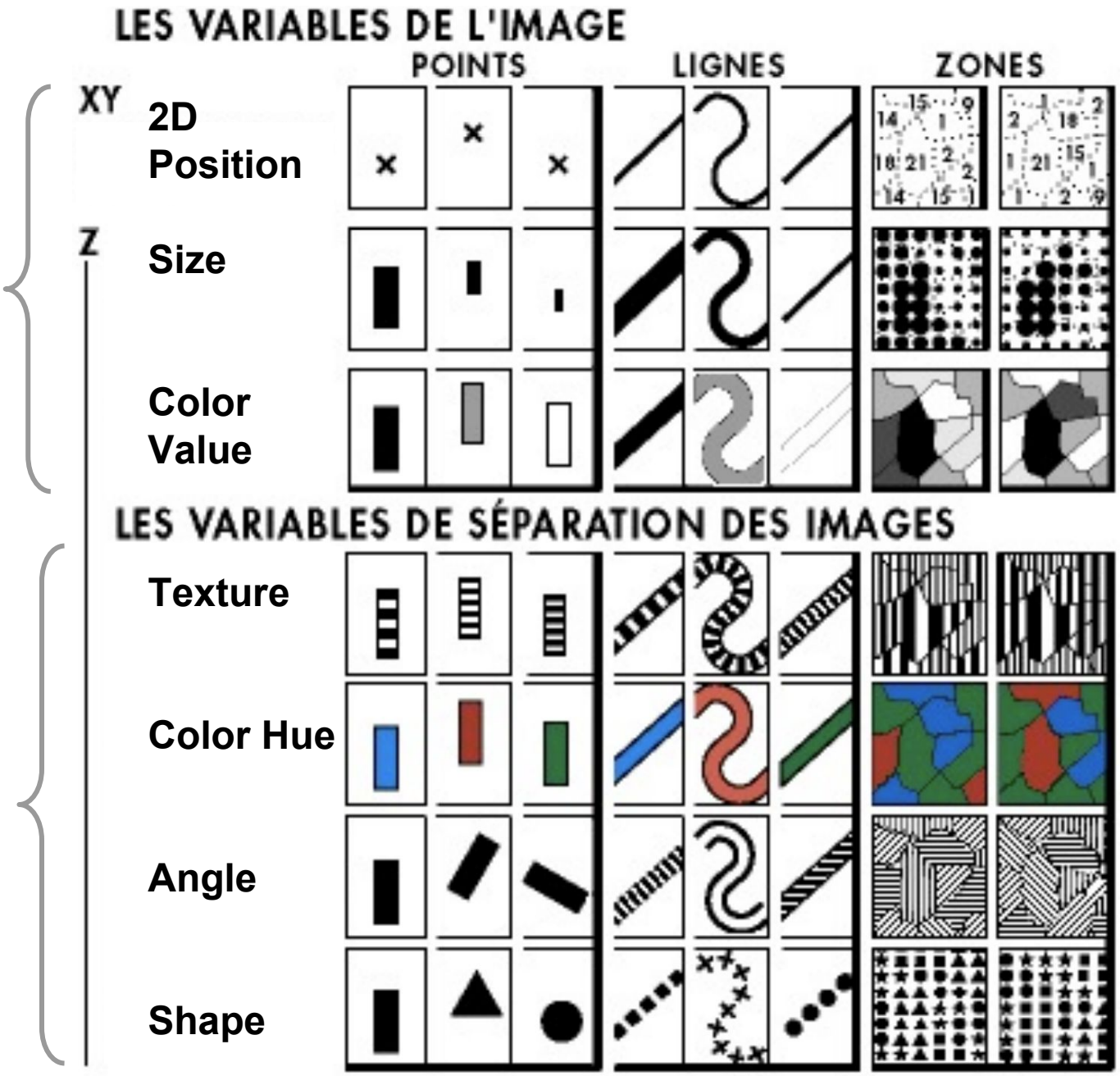




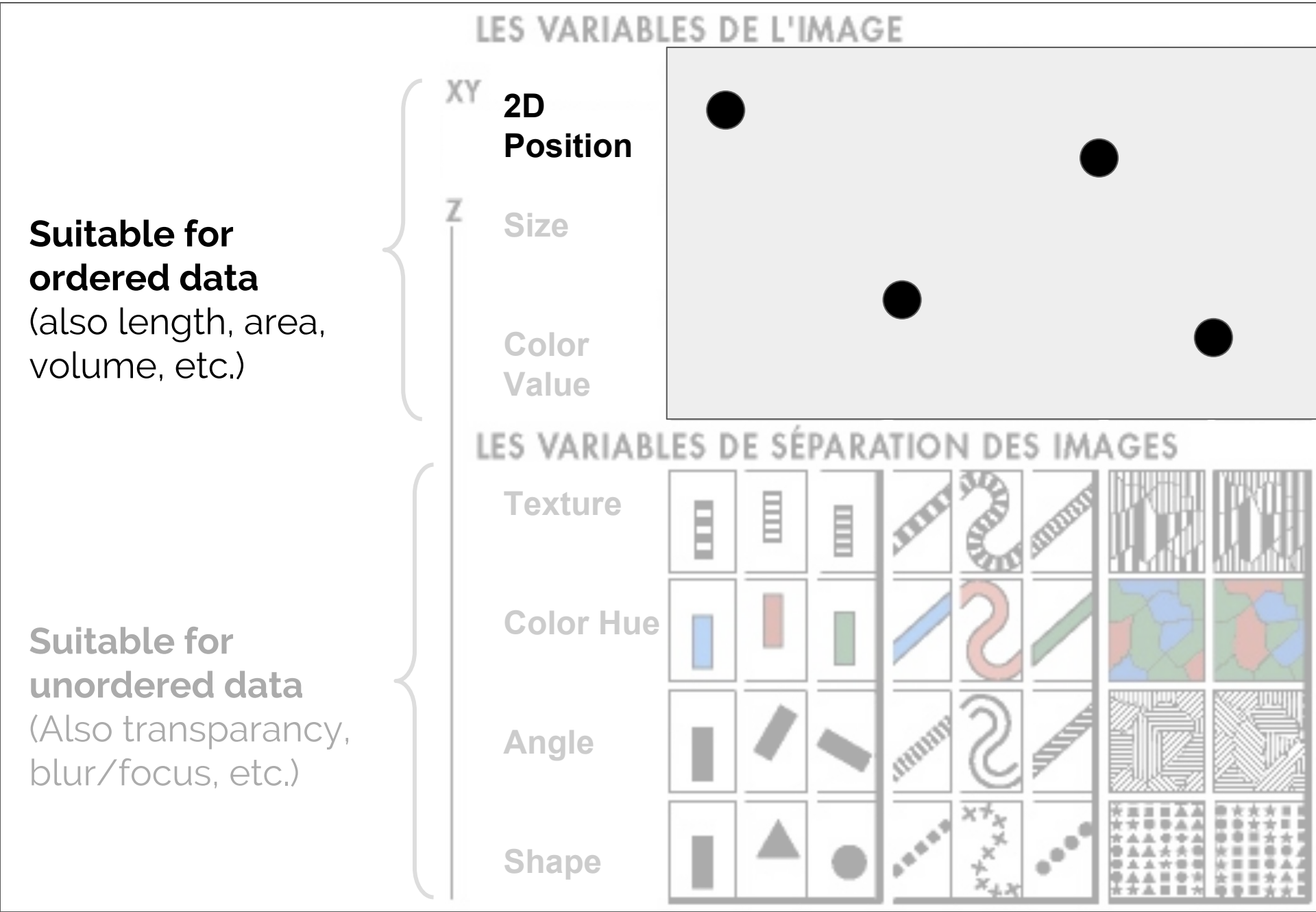
Bertin's Semiology of Graphics (1967)

Suitable for ordered data
(also length, area, volume, etc.)

Suitable for unordered data
(Also transparancy, blur/focus, etc.)



Bertin's Semiology of Graphics (1967)



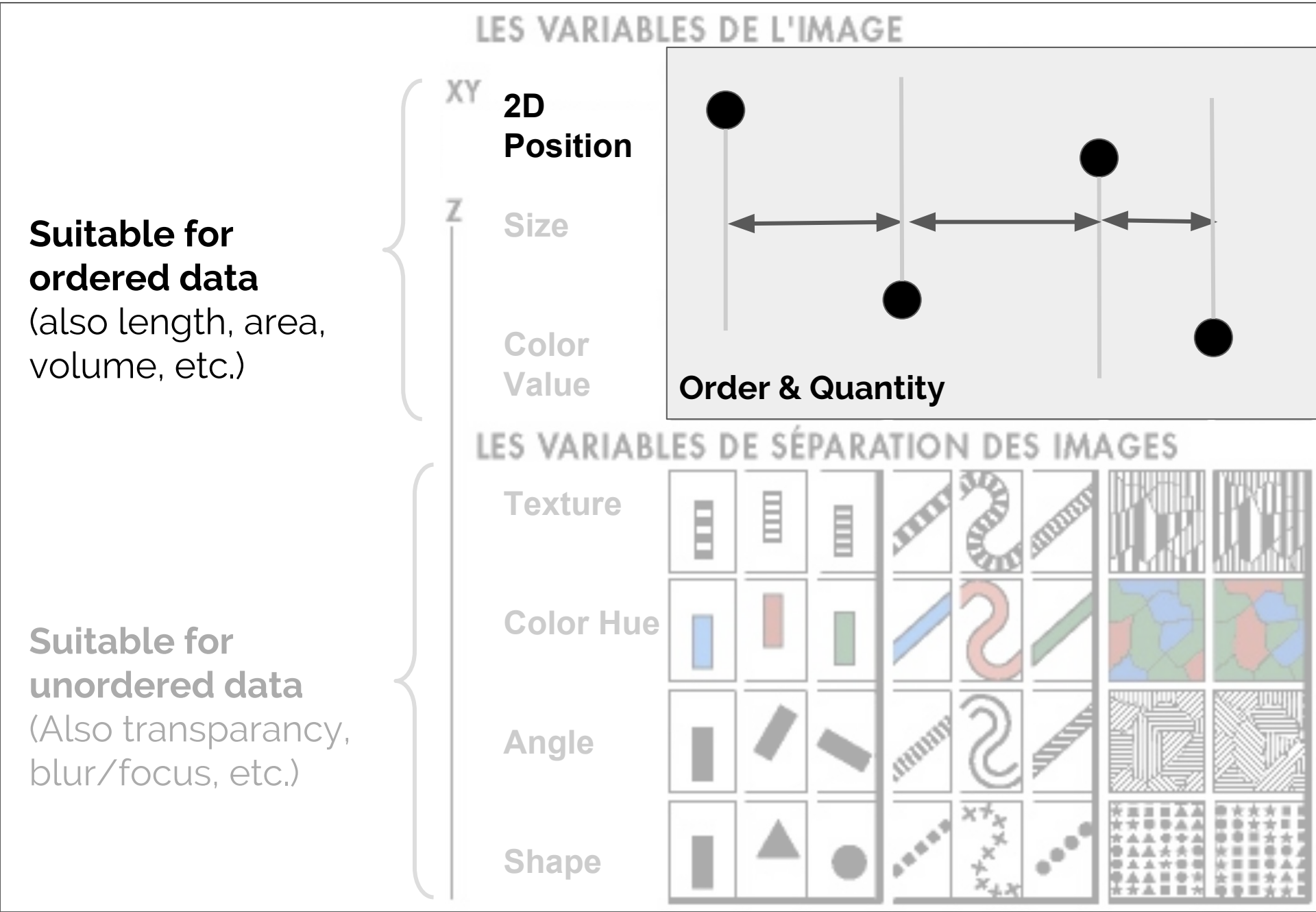
Suitable for ordered data
(also length, area, volume, etc.)

Suitable for unordered data
(Also transparancy, blur/focus, etc.)

Bertin's Semiology of Graphics (1967)

Suitable for ordered data
(also length, area, volume, etc.)

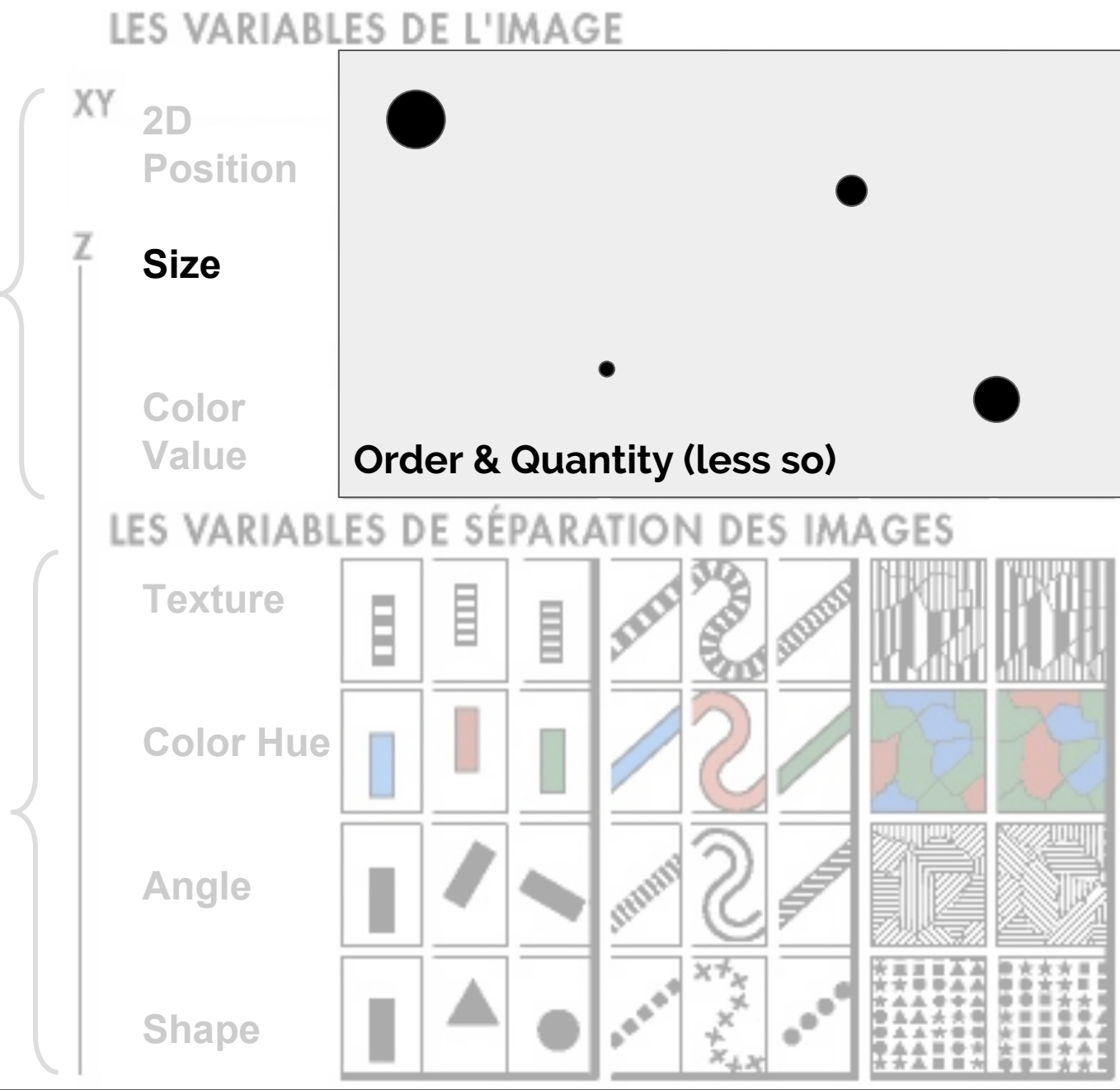
Suitable for unordered data
(Also transparancy, blur/focus, etc.)



Bertin's Semiology of Graphics (1967)

Suitable for ordered data
(also length, area, volume, etc.)

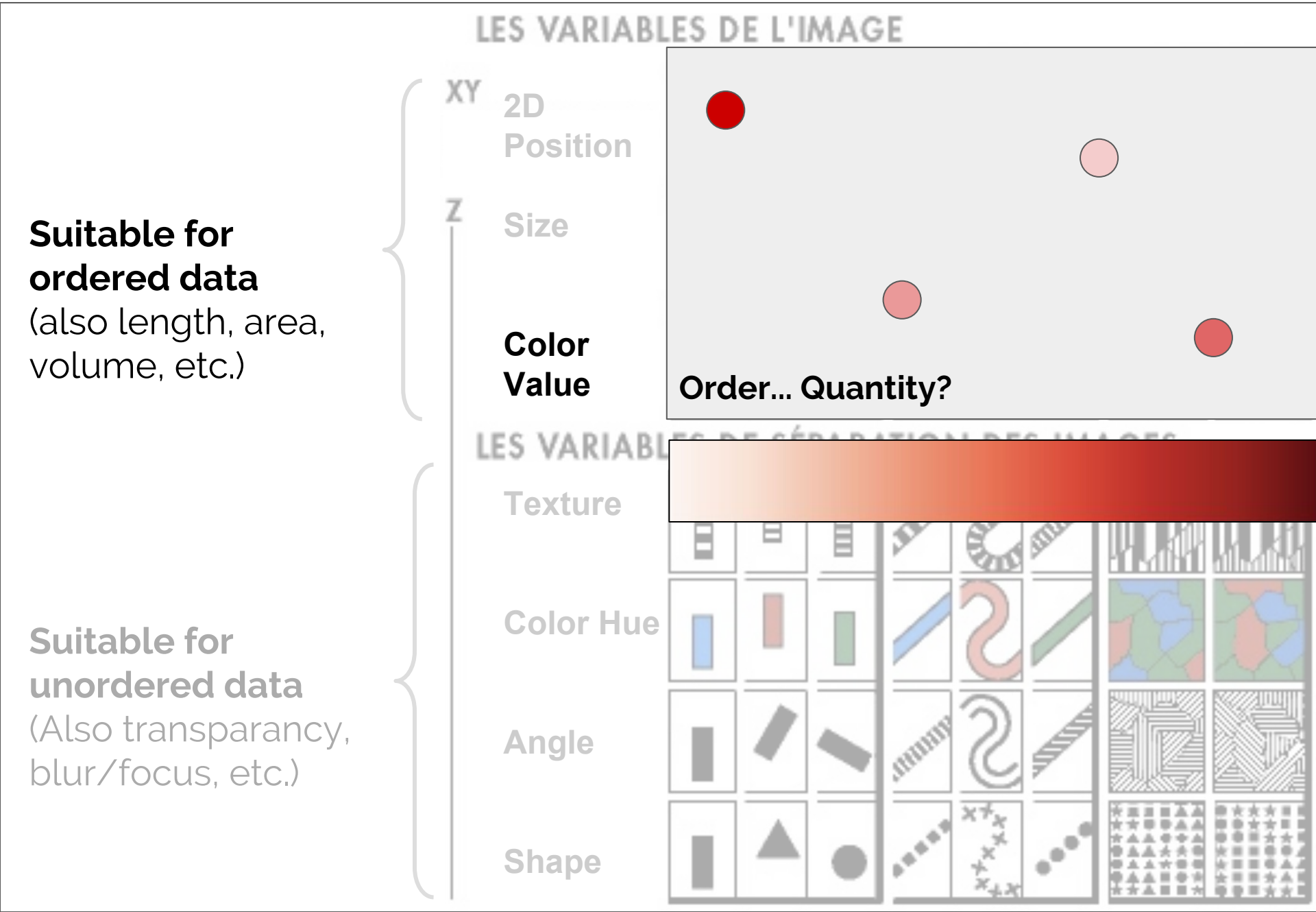
Suitable for unordered data
(Also transparancy, blur/focus, etc.)



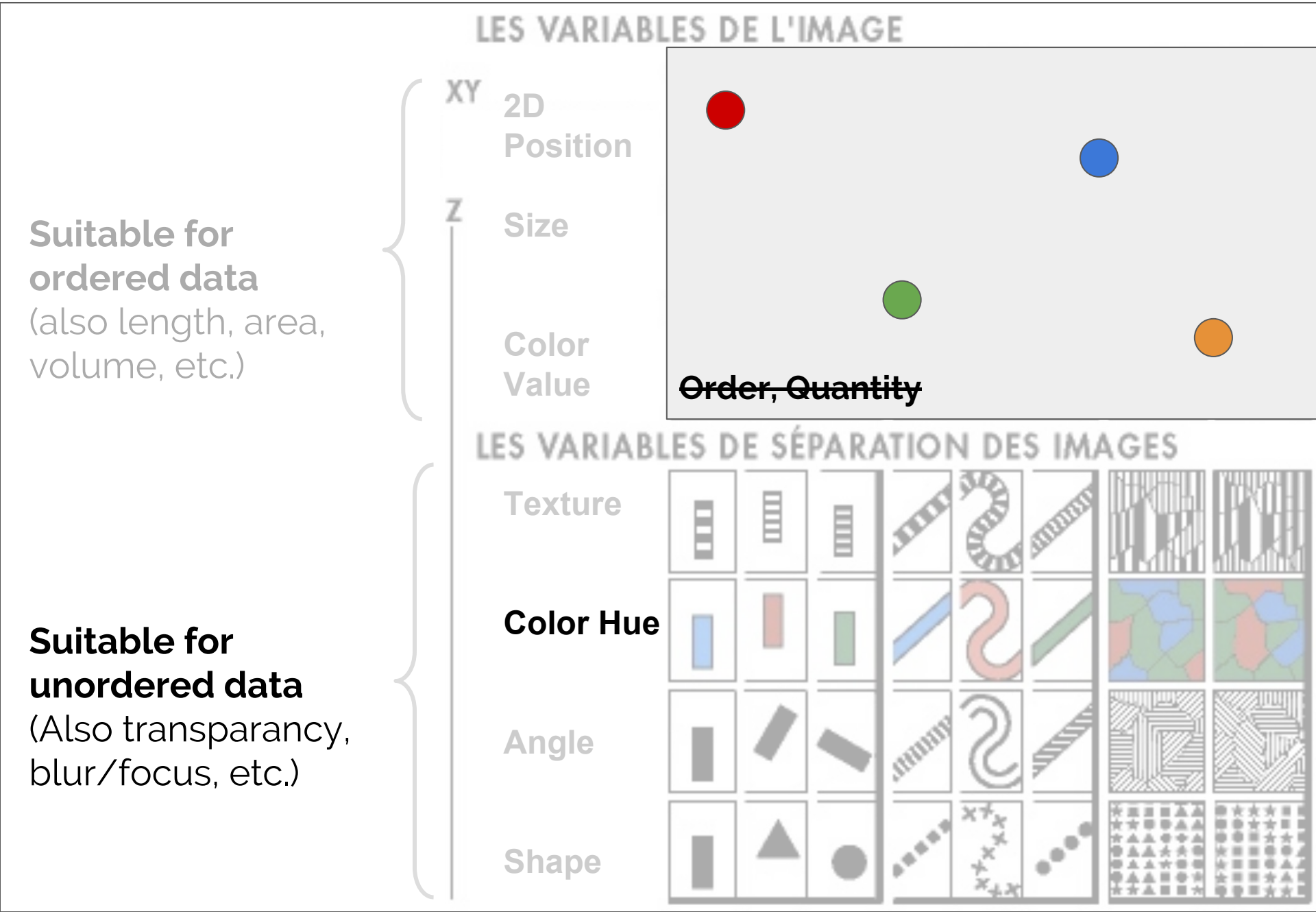
Bertin's Semiology of Graphics (1967)

Suitable for ordered data
(also length, area, volume, etc.)

Suitable for unordered data
(Also transparancy, blur/focus, etc.)



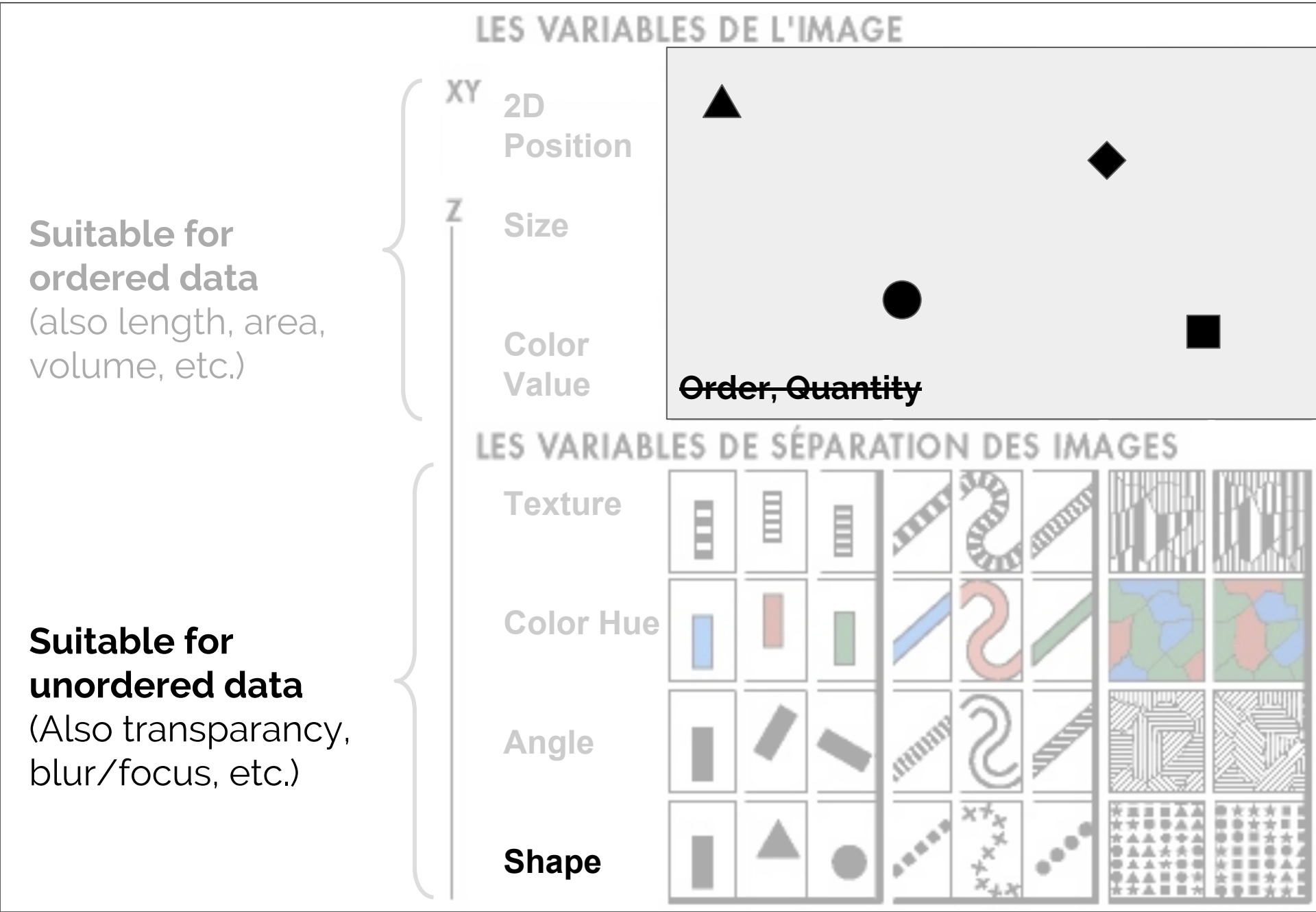
Bertin's Semiology of Graphics (1967)



Suitable for ordered data
(also length, area, volume, etc.)

Suitable for unordered data
(Also transparancy, blur/focus, etc.)

Bertin's Semiology of Graphics (1967)



Suitable for ordered data
(also length, area, volume, etc.)

Suitable for unordered data
(Also transparancy, blur/focus, etc.)

Bertin's Semiology o

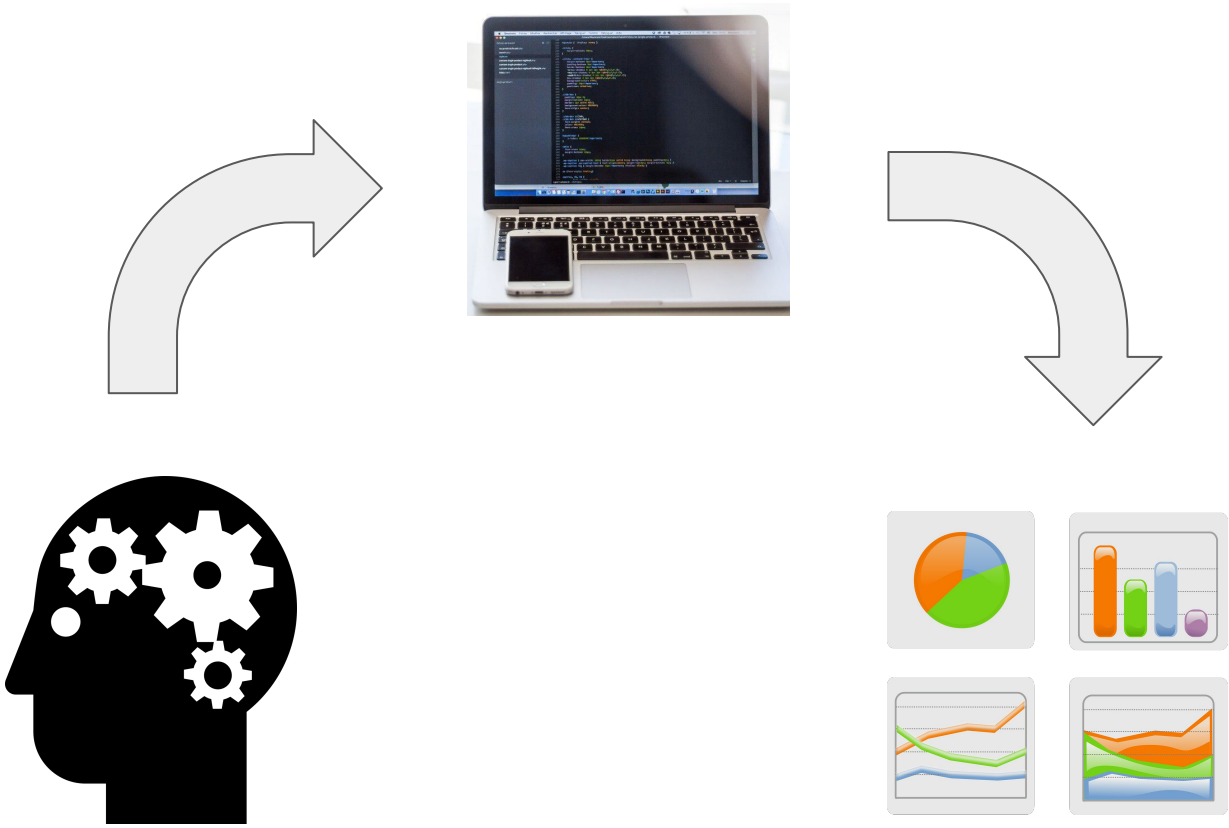
Bertin's "Levels of Organization"

N = Nominal (named category)
O = Ordinal (ordered category)
Q = Quantitative (ordered continuous)

Suitable for ordered data
(also length, area, volume, etc.)

Suitable for unordered data
(Also transparancy, blur/focus, etc.)

Position	**N**	**O**	**Q**
Size	**N**	**O**	**Q**
Color Value	**N**	**O**	**Q**
Texture	**N**	**O**	
Color Hue	**N**		
Angle	**N**		
Shape	**N**		



Key: Visualization *concepts* should map directly to visualization *implementation.*

Great resource is Jeff Heer's viz course: https://courses.cs.washington.edu/courses/cse512/16sp/