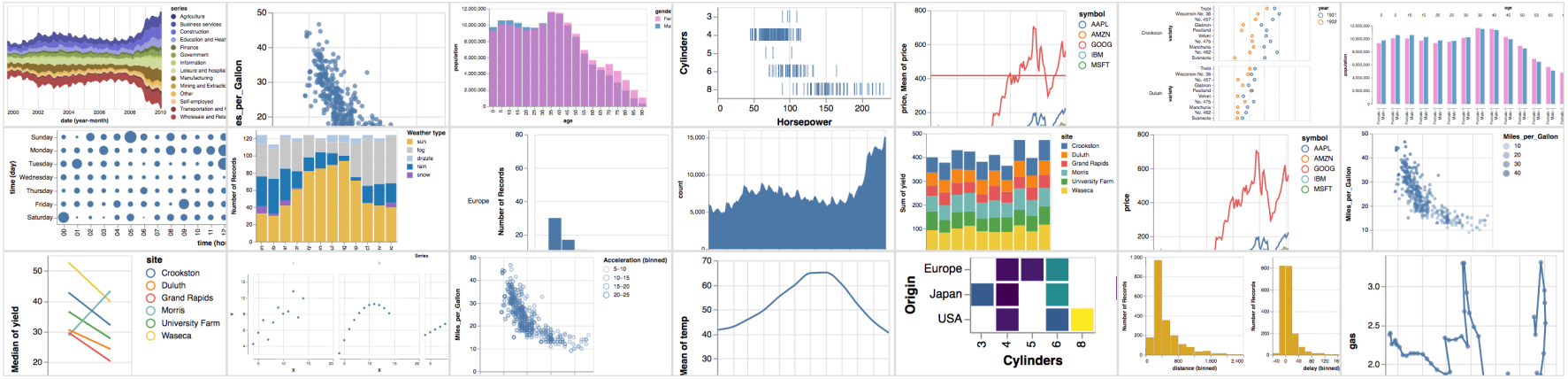
A Tutorial on High-level Grammars for Visualization and Visual Analytics

Gustavo Moreira, University of Illinois Chicago
Marcos Lage, Universidade Federal Fluminense
Nivan Ferreira, Universidade Federal de Pernambuco
Fabio Miranda, University of Illinois Chicago

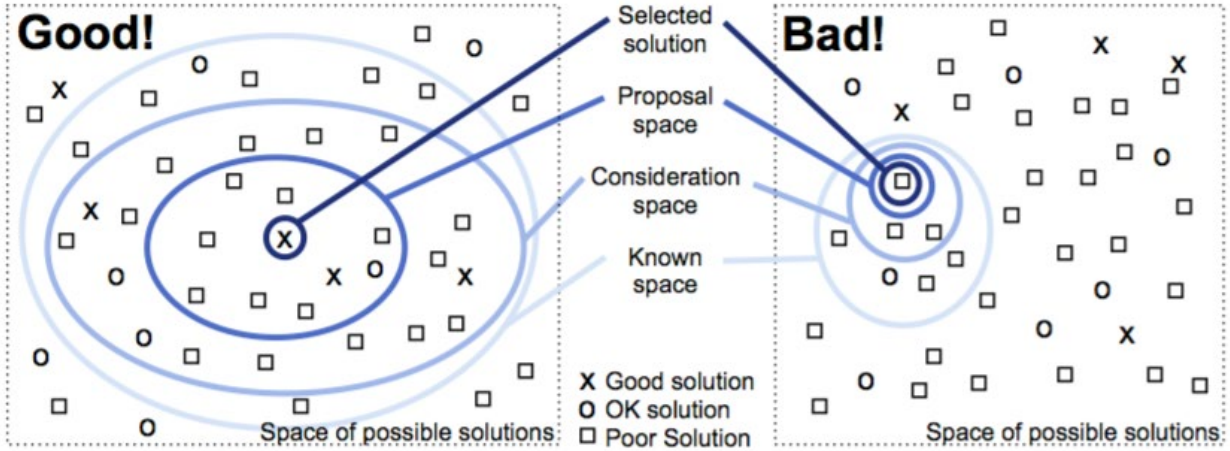
Spectrum of visualization tools

Easy to learn — Tableau — Vega-Lite — Vega — D3 — JavaScript — Difficult to learn

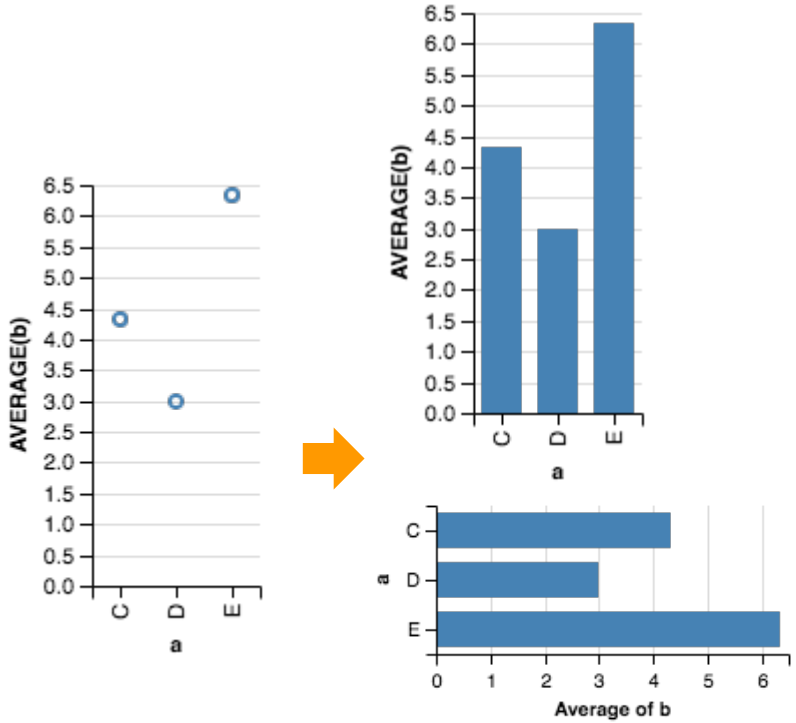
More templated — More flexible



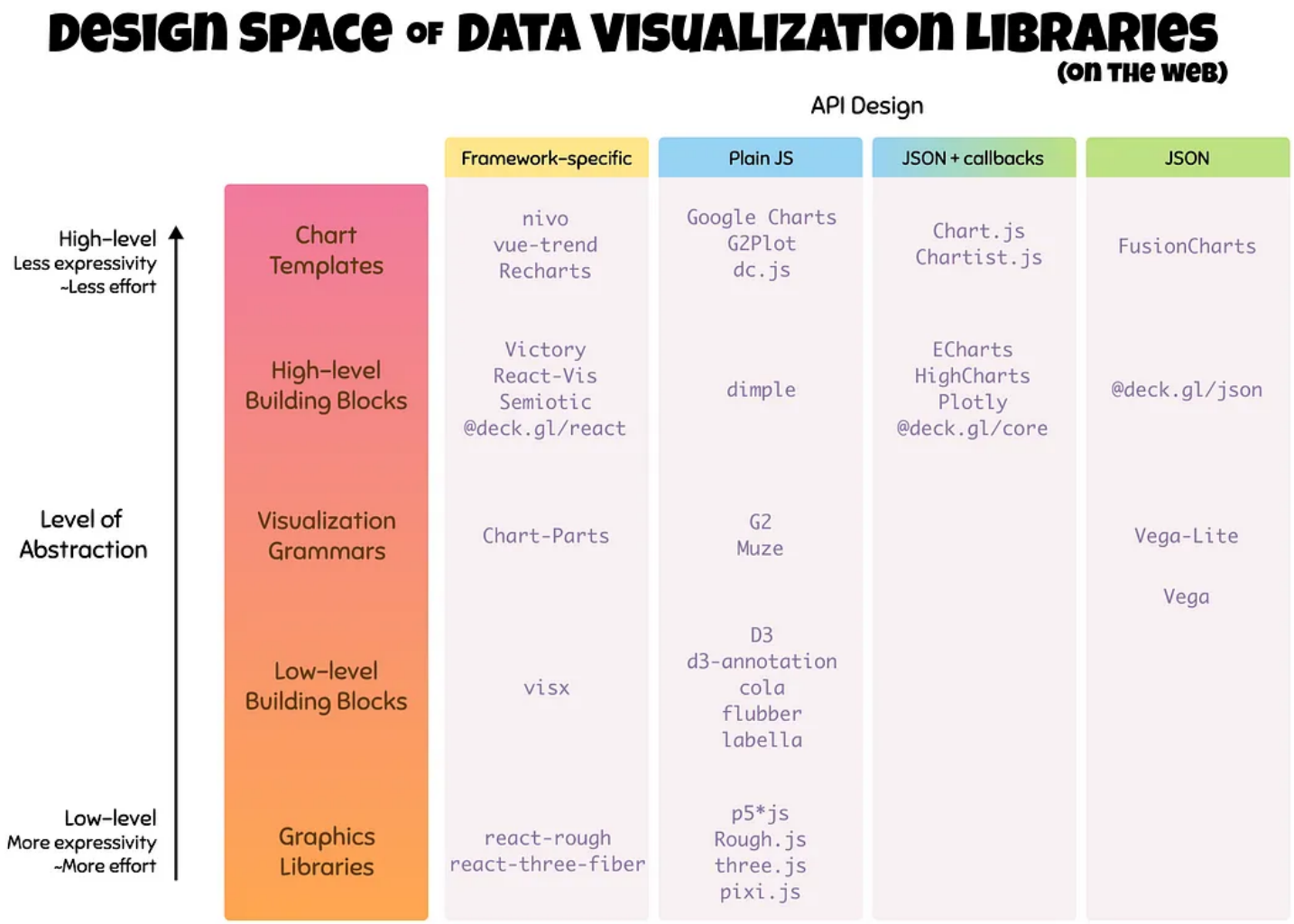
Visualization design space



[Munzner, 2015]



Visualization tools



Level of abstraction:

1) **Effort** required from the developers to create a visualization.

Higher-level libraries:

- Fewer lines of code
- Fewer concepts to learn

2) **Expressivity**, or how much you can customize.

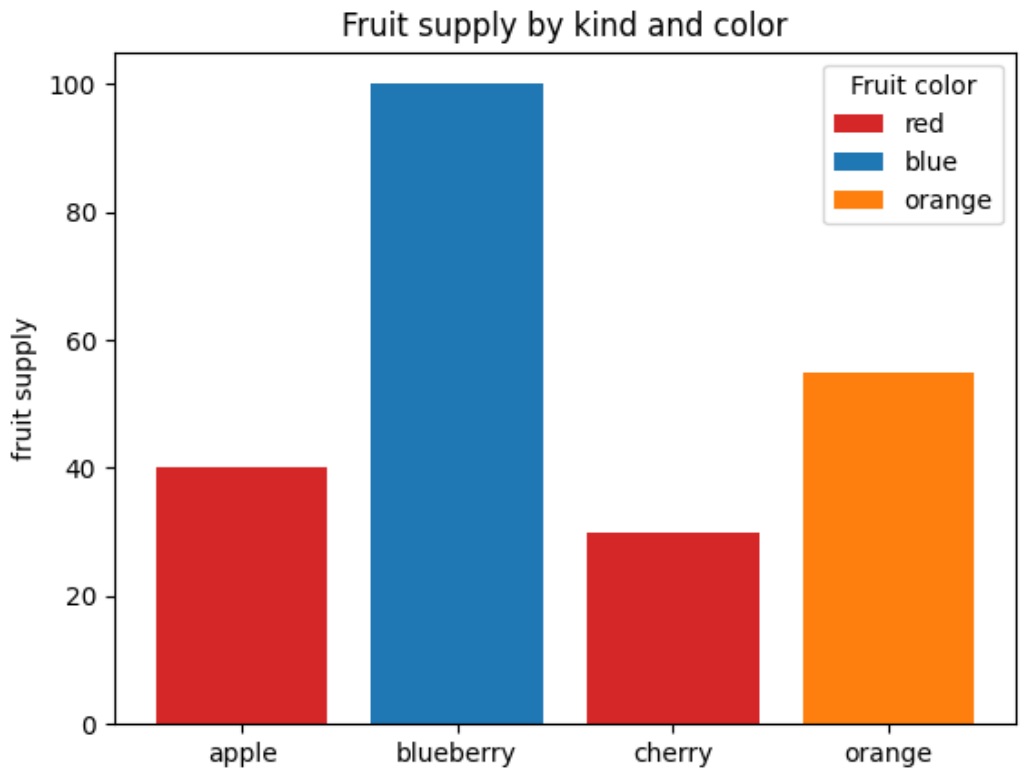
Higher-level libraries:

- Limited customization options

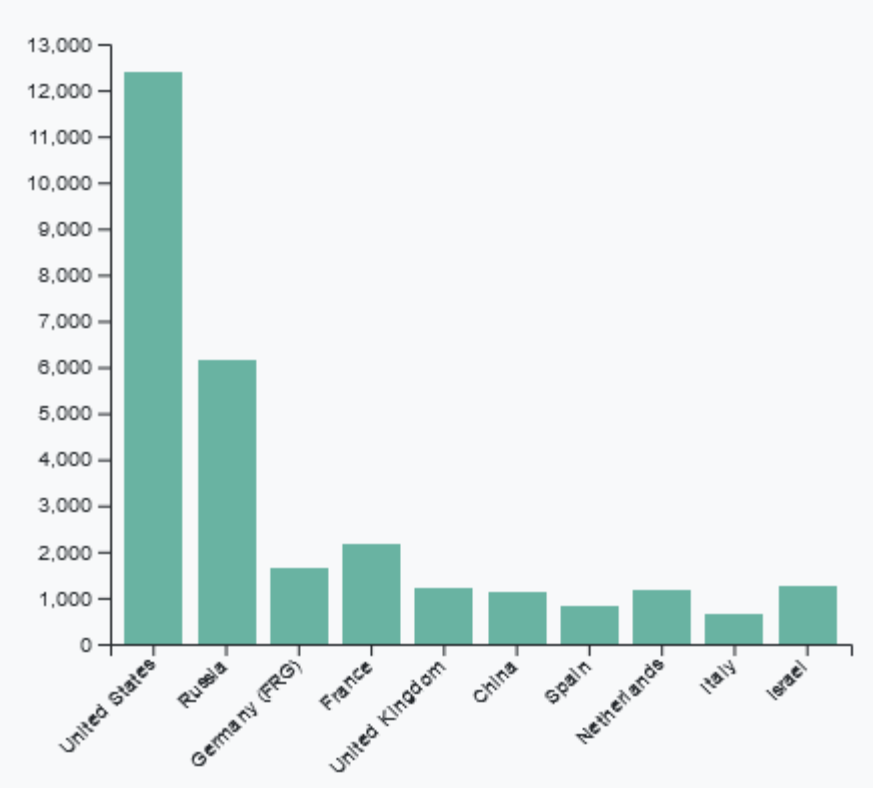
[Wongsuphasawat, 2020]

Visualization tools

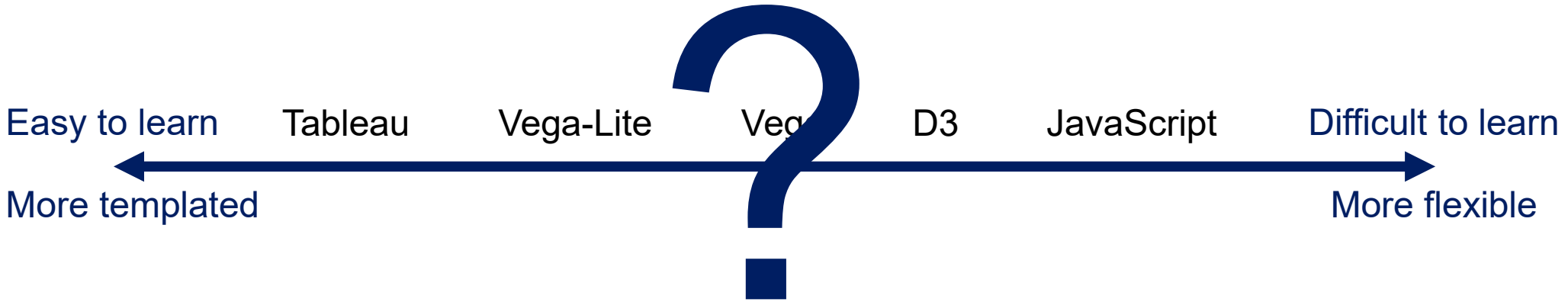
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

fruits = ['apple', 'blueberry', 'cherry', 'orange']
counts = [40, 100, 30, 55]
bar_labels = ['red', 'blue', '_red', 'orange']
bar_colors = ['tab:red', 'tab:blue', 'tab:red',
'tab:orange']

ax.bar(fruits, counts, label=bar_labels,
color=bar_colors)

ax.set_ylabel('fruit supply')
ax.set_title('Fruit supply by kind and color')
ax.legend(title='Fruit color')

plt.show()
```



```
// set the dimensions and margins of the graph
const margin = {top: 30, right: 30, bottom: 70, left: 60},
    width = 460 - margin.left - margin.right,
    height = 400 - margin.top - margin.bottom;

// append the svg object to the body of the page
const svg = d3.select("#my_dataviz")
  .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})`);

// Parse the Data
d3.csv("data.csv").then( function(data) {

  // X axis
  const x = d3.scaleBand().range([ 0, width ]).domain(data.map(d => d.Country)).padding(0.2);
  svg.append("g")
    .attr("transform", `translate(0, ${height})`)
    .call(d3.axisBottom(x))
    .selectAll("text")
      .attr("transform", "translate(-10,0)rotate(-45)")
      .style("text-anchor", "end");

  // Add Y axis
  const y = d3.scaleLinear().domain([0, 13000]).range([ height, 0]);
  svg.append("g").call(d3.axisLeft(y));

  // Bars
  svg.selectAll("mybar")
    .data(data)
    .join("rect").attr("x", d => x(d.Country))
      .attr("y", d => y(d.Value))
      .attr("width", x.bandwidth())
      .attr("height", d => height - y(d.Value))
      .attr("fill", "#69b3a2")

})
```



Spectrum of visualization tools



Grammars

"Grammar is defined as a system of language rules that allows you to combine individual words to make complex meanings."

(Grammarly)

A grammar is a formal system of rules for generating lawful statements in a language.

"By a generative grammar I mean simply a system of rules that in some explicit and well-defined way assigns structural descriptions to sentences. Obviously, every speaker of a language has mastered and internalized a generative grammar that expresses his knowledge of his language. This is not to say that he is aware of the rules of the grammar or even that he can become aware of them, or that his statements about his intuitive knowledge of the language are necessarily accurate. "

(Noam Chomsky)

Expressivity

- “Grammar makes language expressive.”
- Language with just words and no grammar expresses only as many ideas as there are words.

Expressivity

- “Grammar makes language expressive.”
- **Language with just words and no grammar expresses only as many ideas as there are words.**

Expressivity

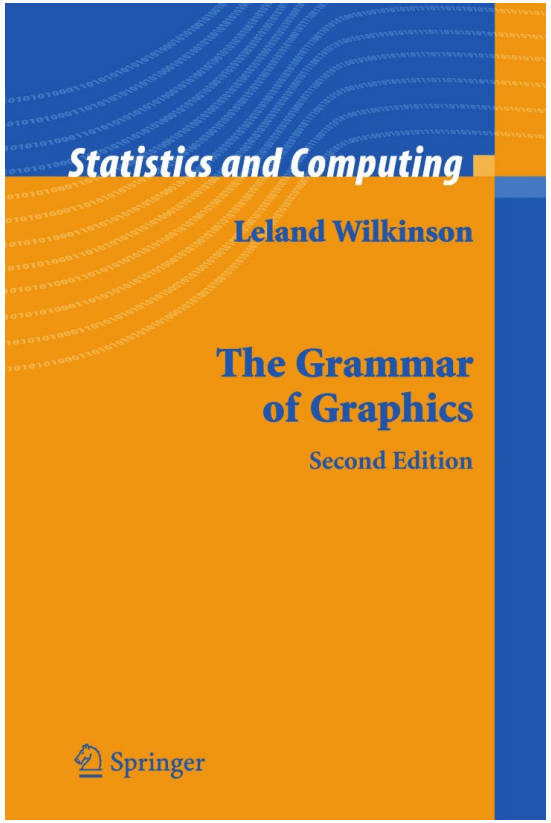
- “Grammar makes language expressive.”
- **Language with just words and no grammar expresses only as many ideas as there are words.**
- “A grammar expands a language’s scope by specifying how words are combined in a statement.”

Grammars

(…) a grammar, a stupendously sophisticated system of logical principles and parameters. This ***grammar can be understood as an expression of the innate, genetically installed "operating system" that endows humans with the capacity to generate complex sentences and long trains of thought****. When linguists seek to develop a theory for why a given language works as it does ("Why are these — but not those — sentences considered grammatical?"), they are building consciously and laboriously an explicit version of the grammar that the child builds instinctively and with minimal exposure to information.*

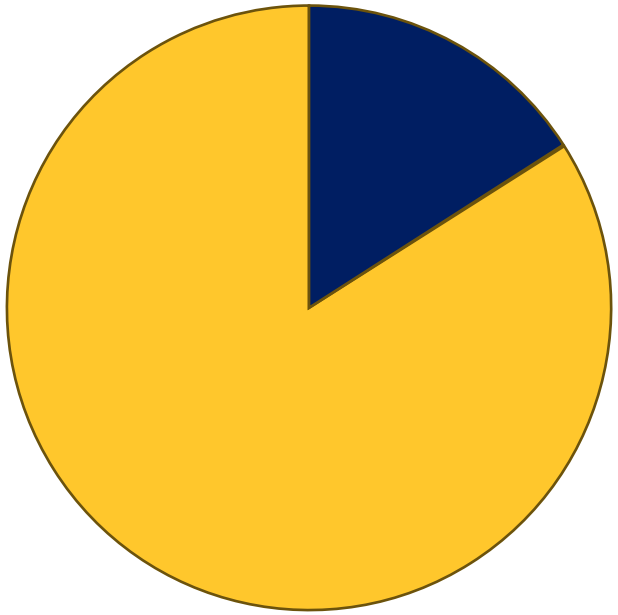
[Noam Chomsky: The False Promise of ChatGPT]

Grammar of graphics

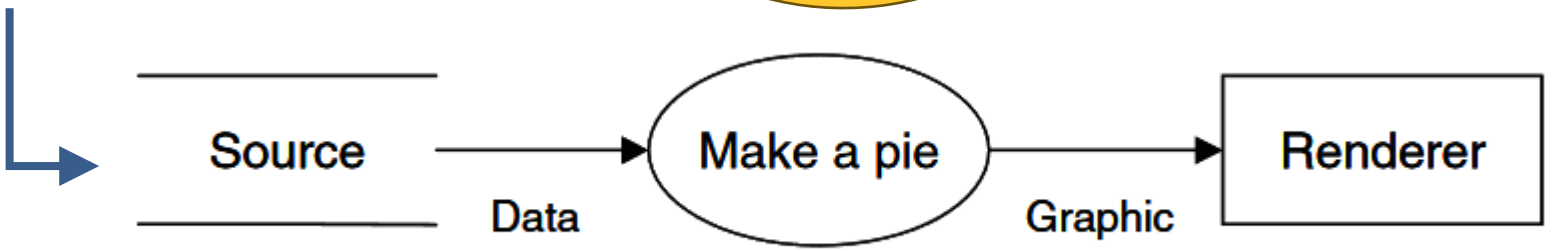


- A grammar to described and create a wide range of statistical graphics.
- Concisely describe the components of a graphic.

How to make a pie (chart)

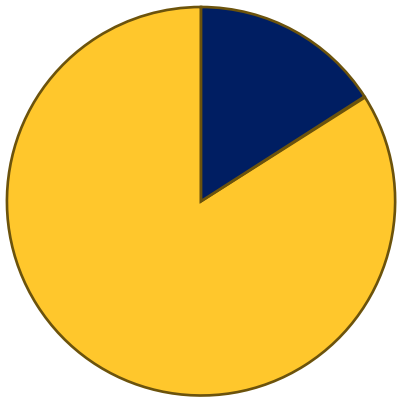


Data flow to making a pie (chart)



Graphics vs charts

- Charts: pie charts, bar charts, line charts, …
  - Typology of charts.
  - Simply instances of much more general objects.
- Pie chart:

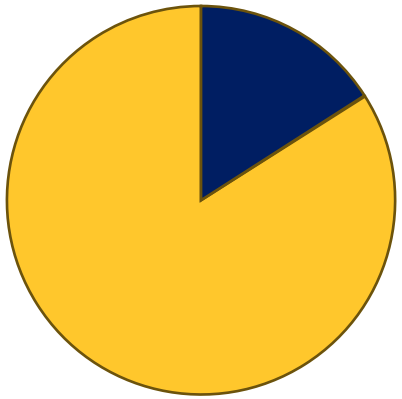


Option 1: `make_pie_chart`

- No deep structure
- Program will be unnecessary complex (fail to reuse objects or routines that function similarly in different charts)
- No way to add new charts without adding complex new code.

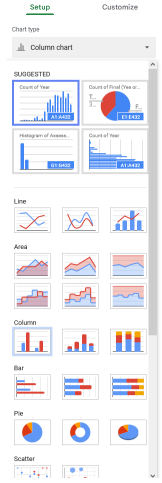
Graphics vs charts

- Charts: pie charts, bar charts, line charts, …
  - Typology of charts.
  - Simply instances of much more general objects.
- Pie chart:

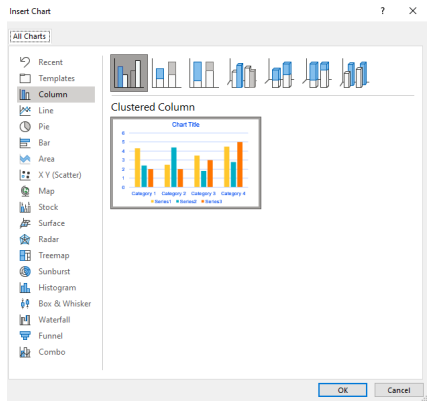


Option 1: `make_pie_chart`

Google

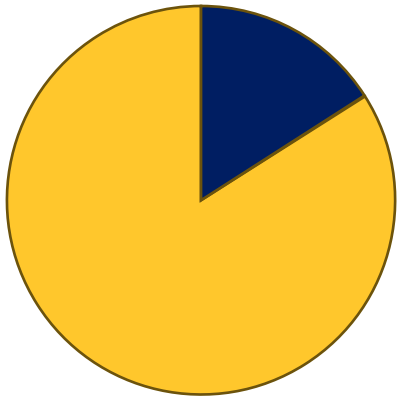


Powerpoint

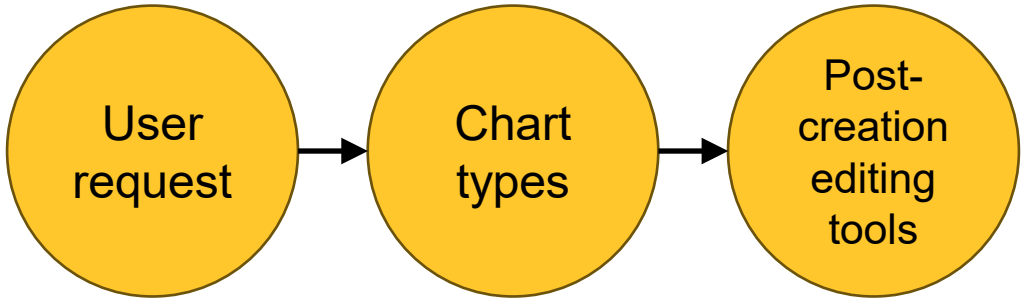


Graphics vs charts

- Charts: pie charts, bar charts, line charts, …
  - Typology of charts.
  - Simply instances of much more general objects.
- Pie chart:

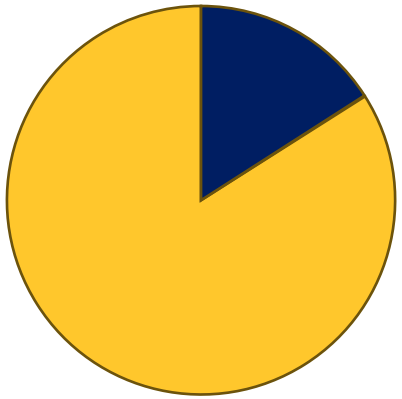


Option 1: `make_pie_chart`



Graphics vs charts

- Charts: pie charts, bar charts, line charts, …
  - Typology of charts.
  - Simply instances of much more general objects.
- Pie chart:

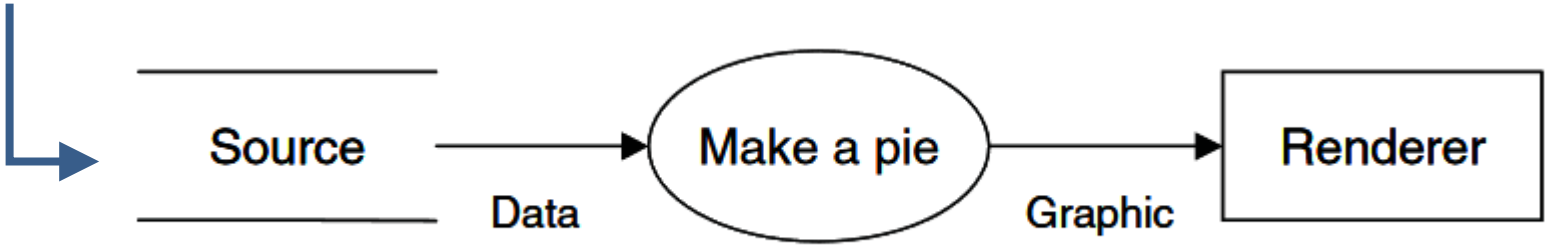


Option 1: `make_pie_chart`

"… give the user the impression of having explored data rather than the experience."

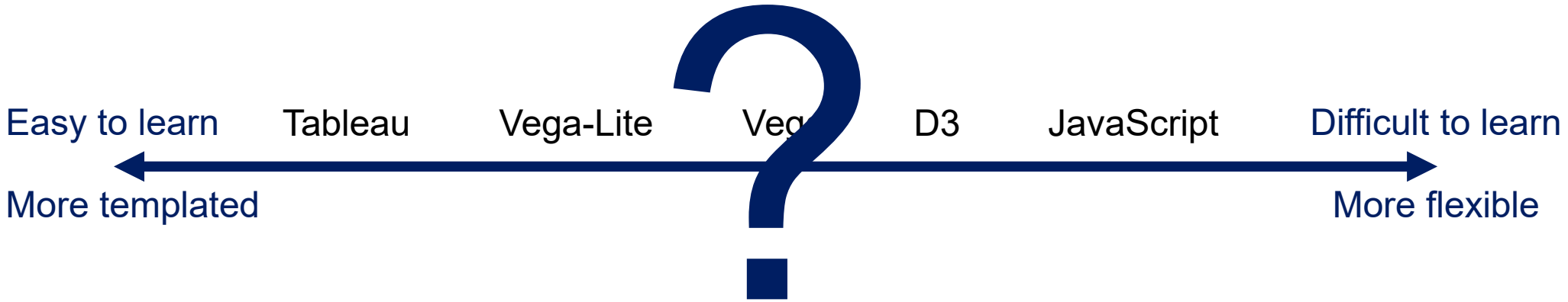
How to make a pie (chart)

Data flow to making a pie (chart)



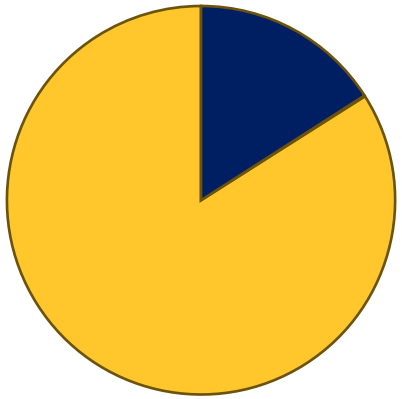
- Data flow to making a pie (chart):
  - What is the format of the data?
  - How are the pie pieces colored?
  - What variables should be used to label the pie?
  - …
- Goal of grammar of graphics is to create new charts flexibly.

Graphics vs charts



Graphics vs charts

- Elegant design requires a theory of graphics not charts.
  - Specification: translation of user actions into a formal language.
  - Grammar rules to describe and create a wide range of graphics.
- Pie chart:

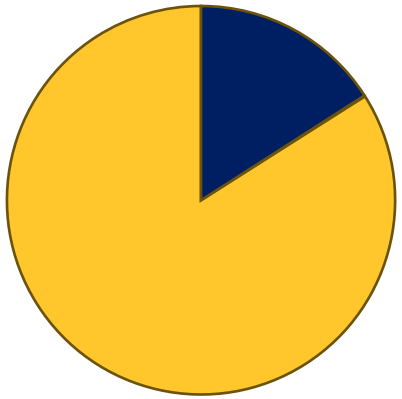


Option 2:

What makes a pie chart a pie chart?

Graphics vs charts

- Elegant design requires a theory of graphics not charts.
  - Specification: translation of user actions into a formal language.
  - Grammar rules to describe and create a wide range of graphics.
- Pie chart:

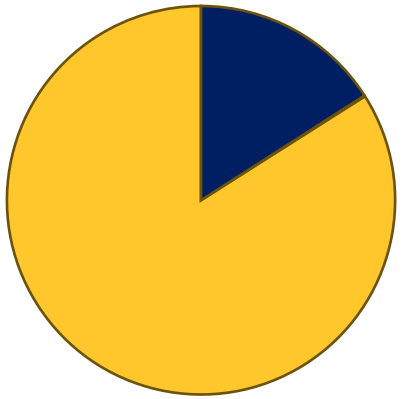


Option 2:

Angle θ and distance to the origin r

Graphics vs charts

- Elegant design requires a theory of graphics not charts.
  - Specification: translation of user actions into a formal language.
  - Grammar rules to describe and create a wide range of graphics.
- Pie chart:



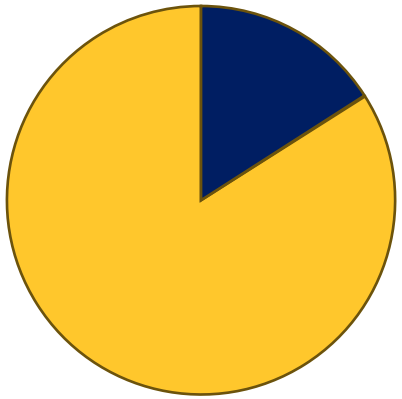
Option 2:

- What is the format of the data?
- How are the pie pieces colored?
- What variables should be used to label the pie?

…

Graphics vs charts

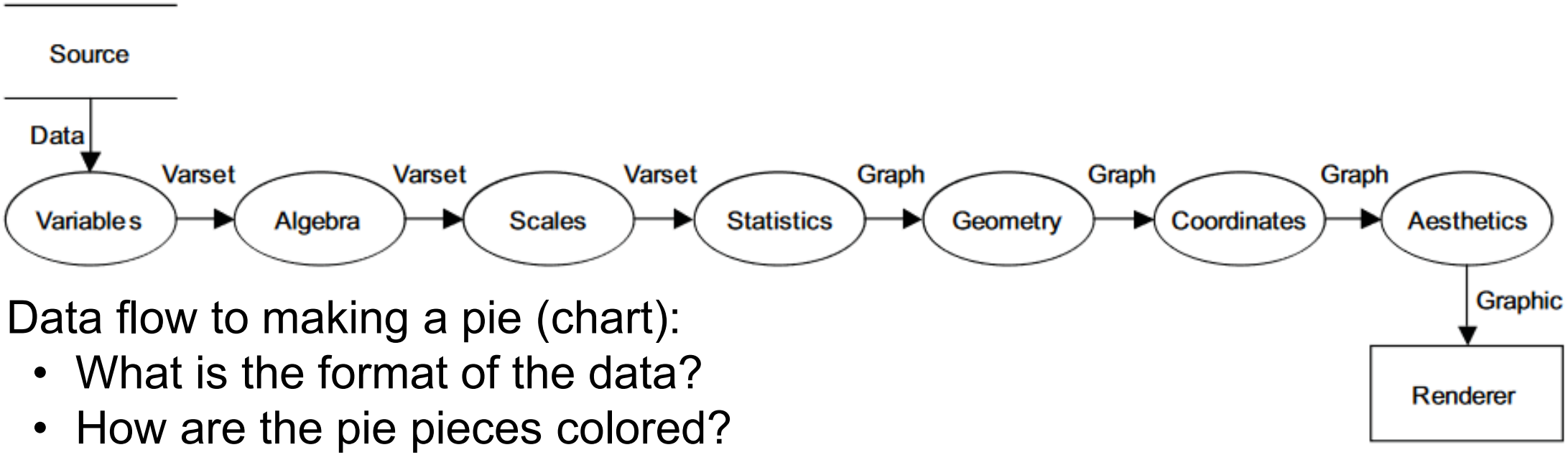
- Elegant design requires a theory of graphics not charts.
  - Specification: translation of user actions into a formal language.
  - Grammar rules to describe and create a wide range of graphics.
- Pie chart:



Option 2:

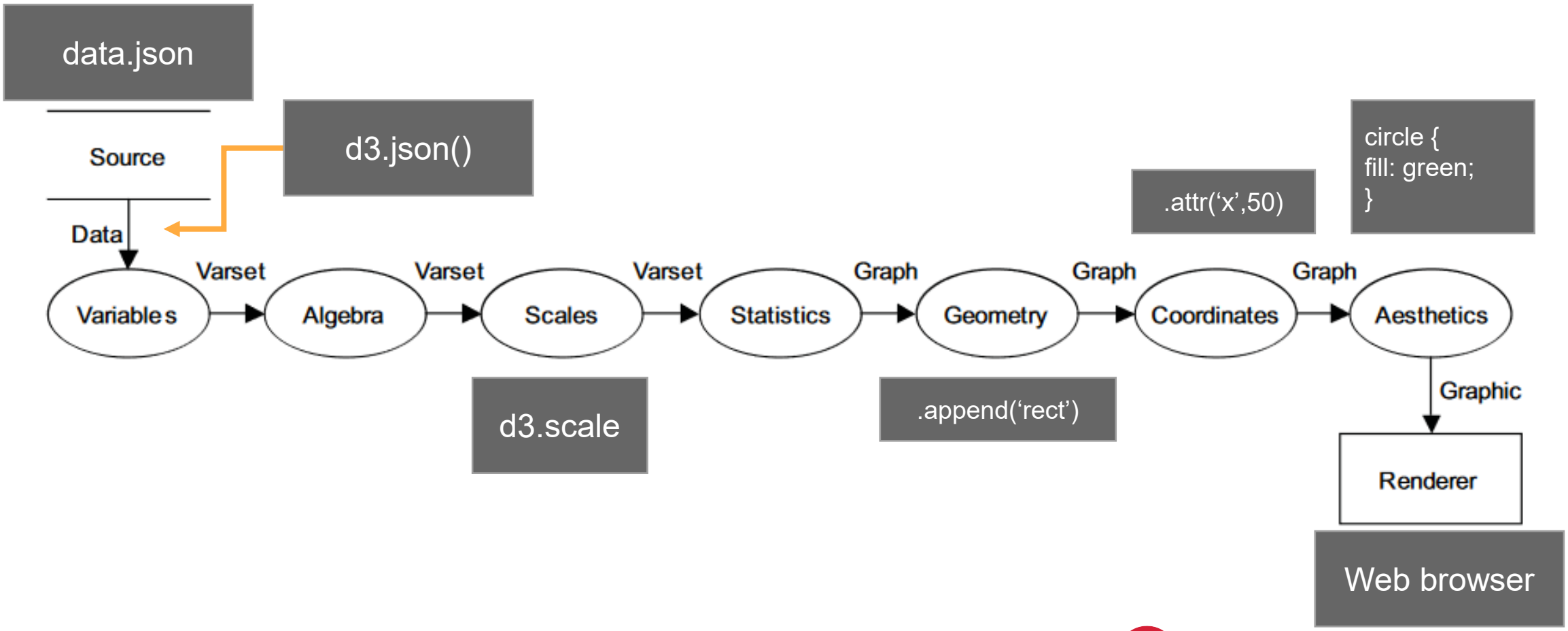
- Deep structure
- Programs that reuse objects or routines that function similarly in different charts
- Can cover the design space of visualizations

How to make a pie (chart)



- Data flow to making a pie (chart):
  - What is the format of the data?
  - How are the pie pieces colored?
  - What variables should be used to label the pie?
  - …
- Goal of grammar of graphics is to create new charts flexibly.

How to make a pie (chart)

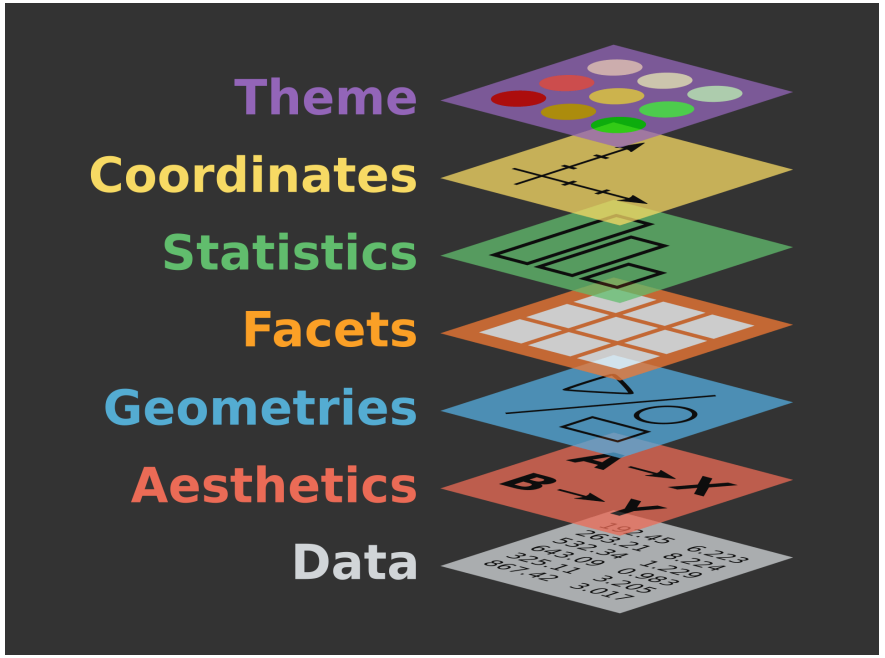


Grammar of graphics

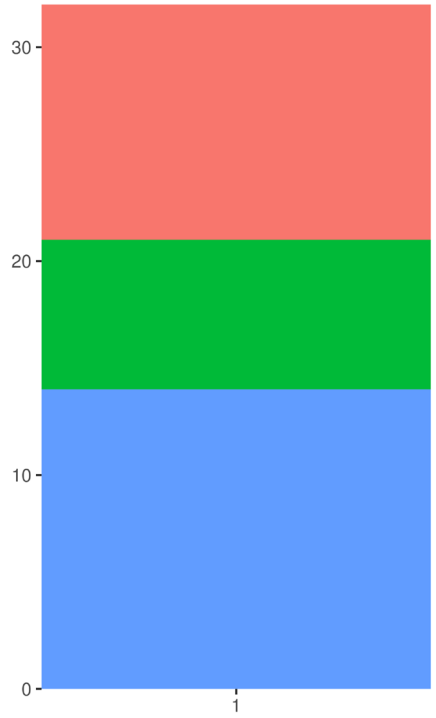
- In a nutshell: “A grammar of graphics is a tool that enables us to concisely describe the components of a graphic. (…) allows us to move beyond named graphics (e.g., the ‘scatterplot’) and **gain insight into the deep structure that underlies statistical graphics**.” (Wickham, 2010)

A layered grammar of graphics

- ggplot2: an implementation of the grammar of graphics.
- Graphs are built step by step, through flexible and customizable layers.

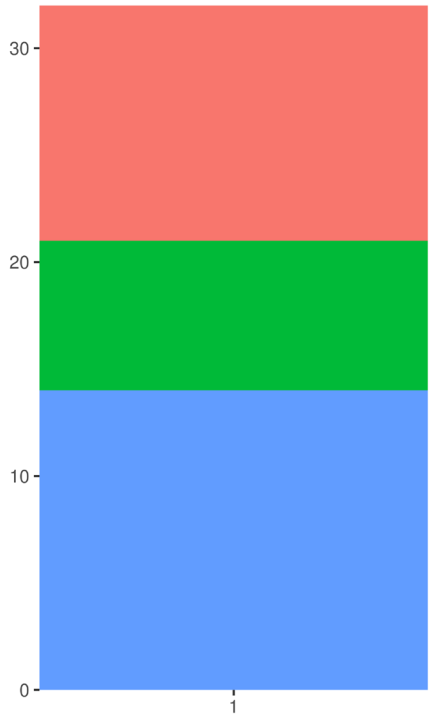


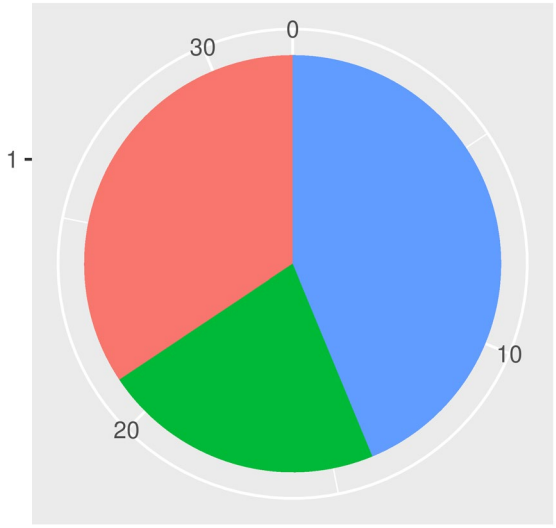
A layered grammar of graphics




```
base <- ggplot(mtcars, aes(factor(1), fill = factor(cyl))) +
  geom_bar(width = 1)
```

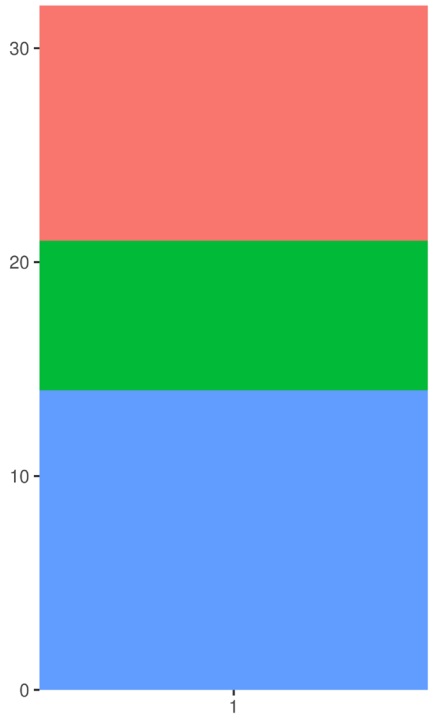
A layered grammar of graphics



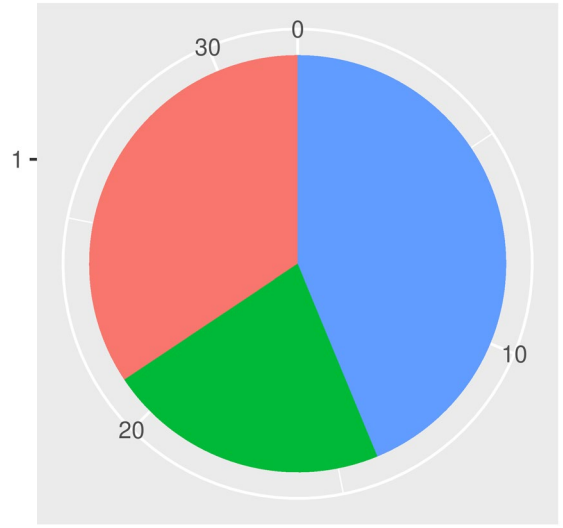


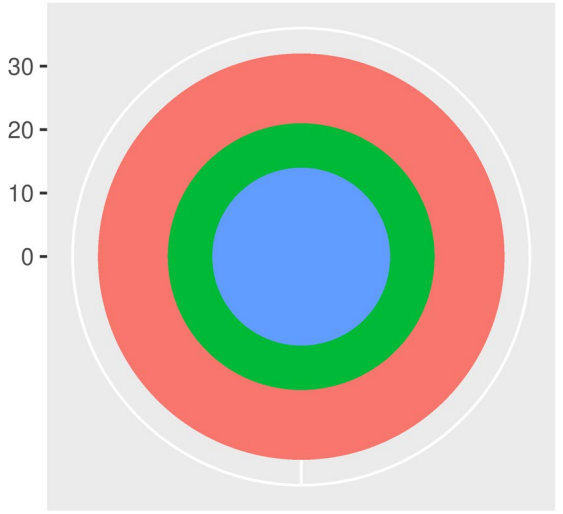

```
base + coord_polar(theta = "y")
```

A layered grammar of graphics



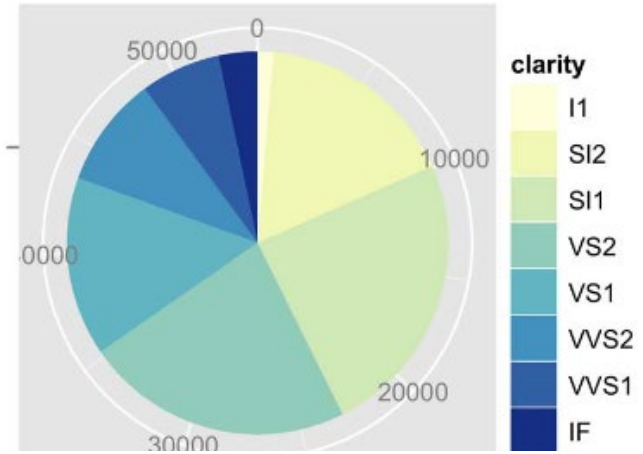

```
base + coord_polar(theta = "x")
```

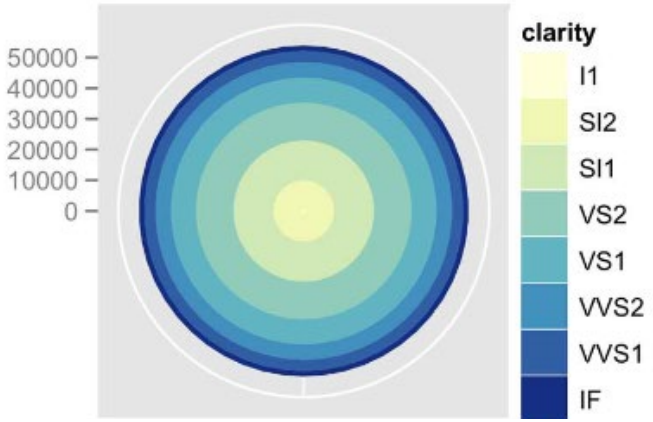


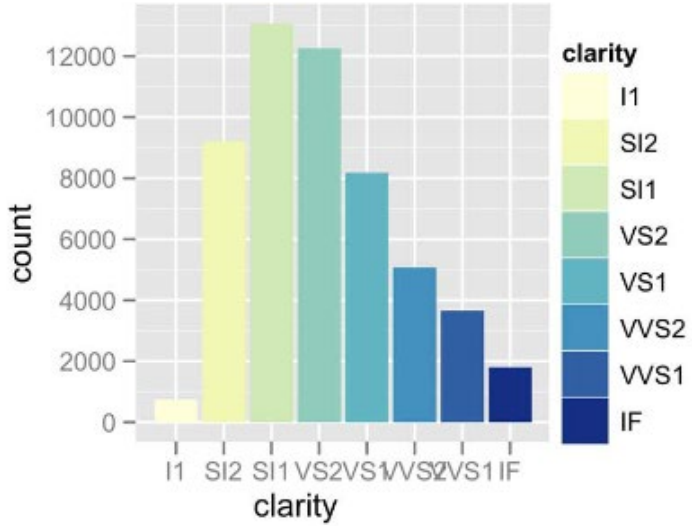


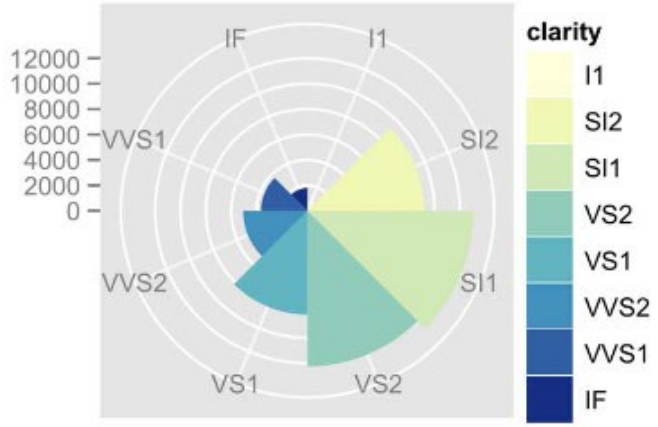

```
base + coord_polar(theta = "y")
```

Expressivity



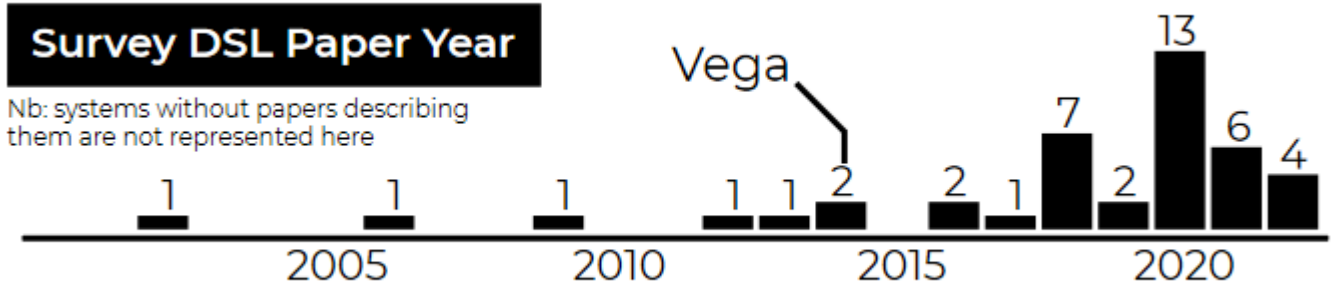






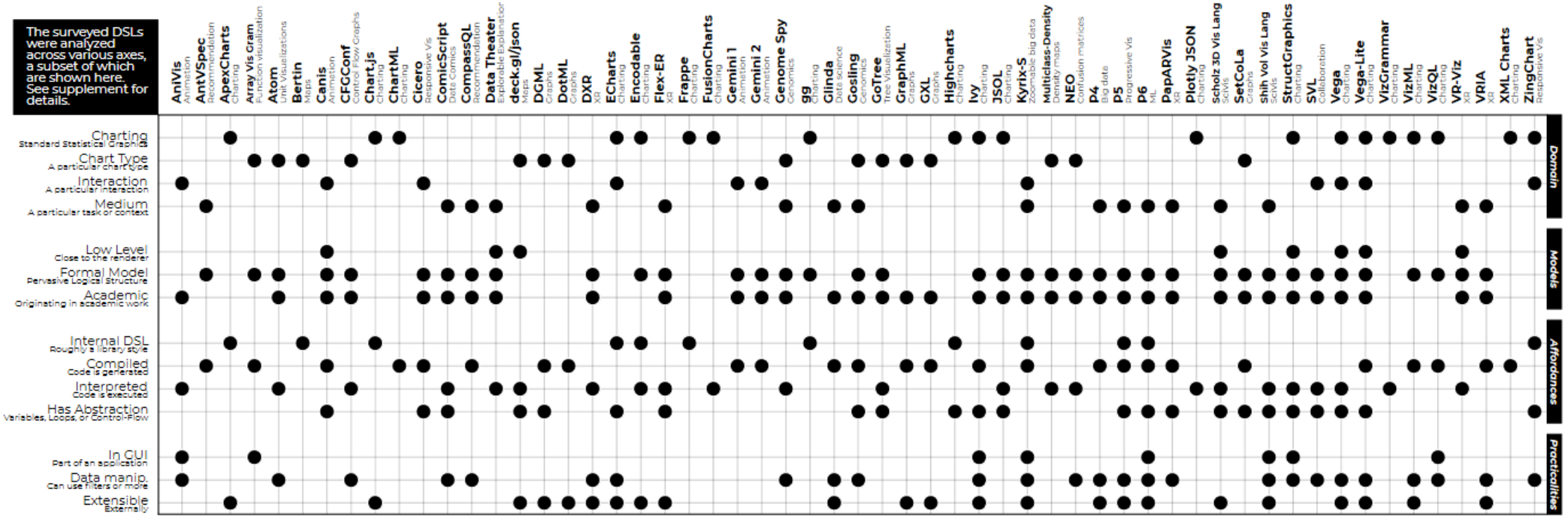
JSON-based grammars

- JSON-based grammars have been employed in a number of visualization and visual analytics systems.



[McNutt, 2022]

JSON-based grammars



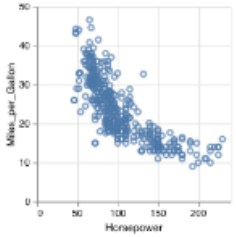
[McNutt, 2022]

JSON-based grammars

Vega-Lite

```
{data: {url: "data/cars.json"},
 mark: "point",
 encoding: {
    x: {field: "Horsepower", type: "quantitative"},
    y: {field: "Miles_per_Gallon",
        type: "quantitative"}}}
```

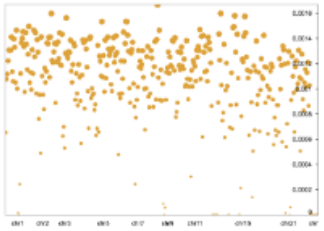
GoG-style mappings (as here) provide a fast-flexible control over the chart design space allowing smooth exploration and clear documentation of intent.



Gosling

```
{tracks: [{layout: "linear", width: 600, height: 400,
 data: {url: "<URL>", type: "multivec",
   row: "sample", column: "position",
   value: "peak", categories: ["sample 1"]},
 mark: "point",
 x: {field: "position", type: "genomic", axis: "bottom"},
 y: {field: "peak", type: "quantitative", axis: "right"},
 size: {field: "peak", type: "quantitative"}}]}
```

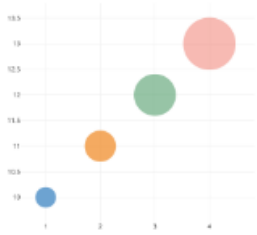
GoG-style languages can be augmented with domain specific content and abstractions, such as in the notion of tracks and genomic coordinates shown here.



Plotly JSON

```
[{x: [1, 2, 3, 4], y: [10, 11, 12, 13], mode: "markers",
  marker: {
    color: ["rgb(93, 164, 214)", "rgb(255, 144, 14)",
     "rgb(44, 160, 101)", "rgb(255, 65, 54)"],
    opacity: [1, 0.8, 0.6, 0.4],
    size: [40, 60, 80, 100]}}]
```

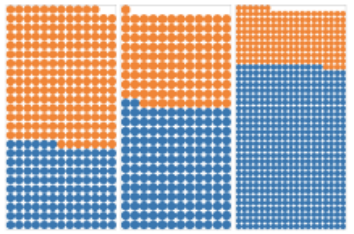
Properties (such as color) in series-models can be applied directly to the relevant component, giving a good *closeness of mapping*.



Atom

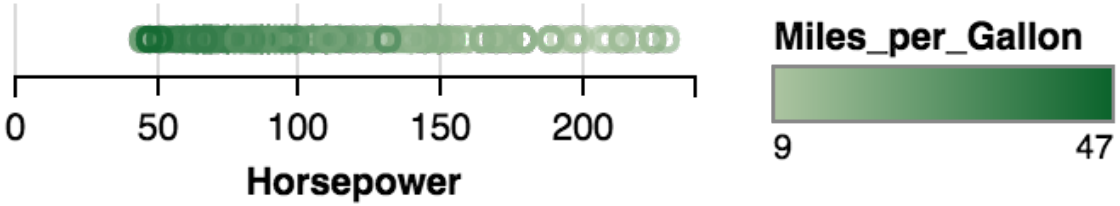
```
{data: {url: "data/titanic3.csv"},
 layouts: [
    {type: "gridxy", aspect_ratio: "fillX",
      subgroup: {type: "groupby", key: "pclass"}},
    {type: "gridxy", aspect_ratio: "maxfill",
      subgroup: {type: "flatten"},
      size: {type: "uniform", isShared: false}}],
 mark: {shape: "circle", color: {key: "survived"}}}
```

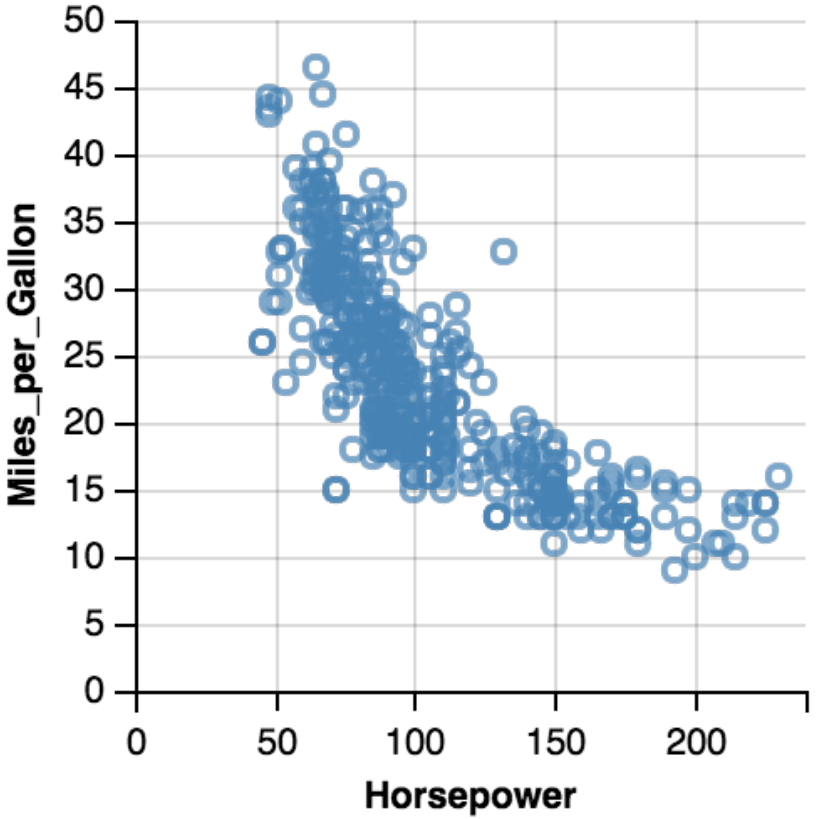
Alternative formal models (such as the L-System inspired framing used here) can motivate alternative analyses and creation of novel chart forms.



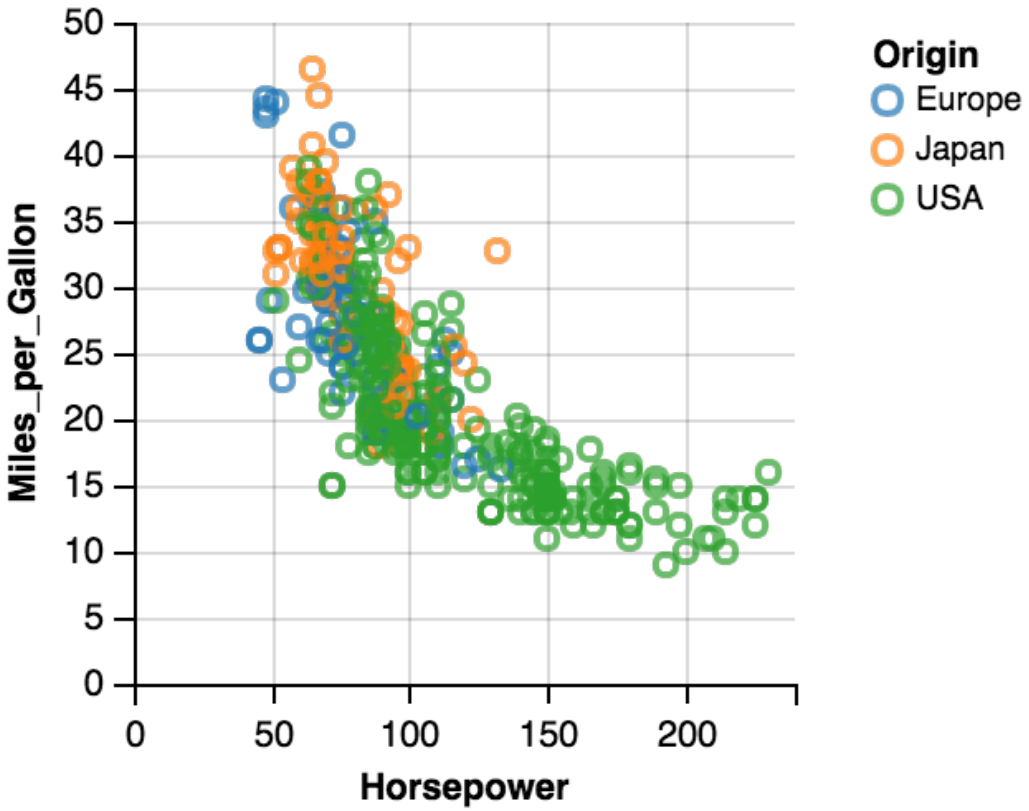
[McNutt, 2022]

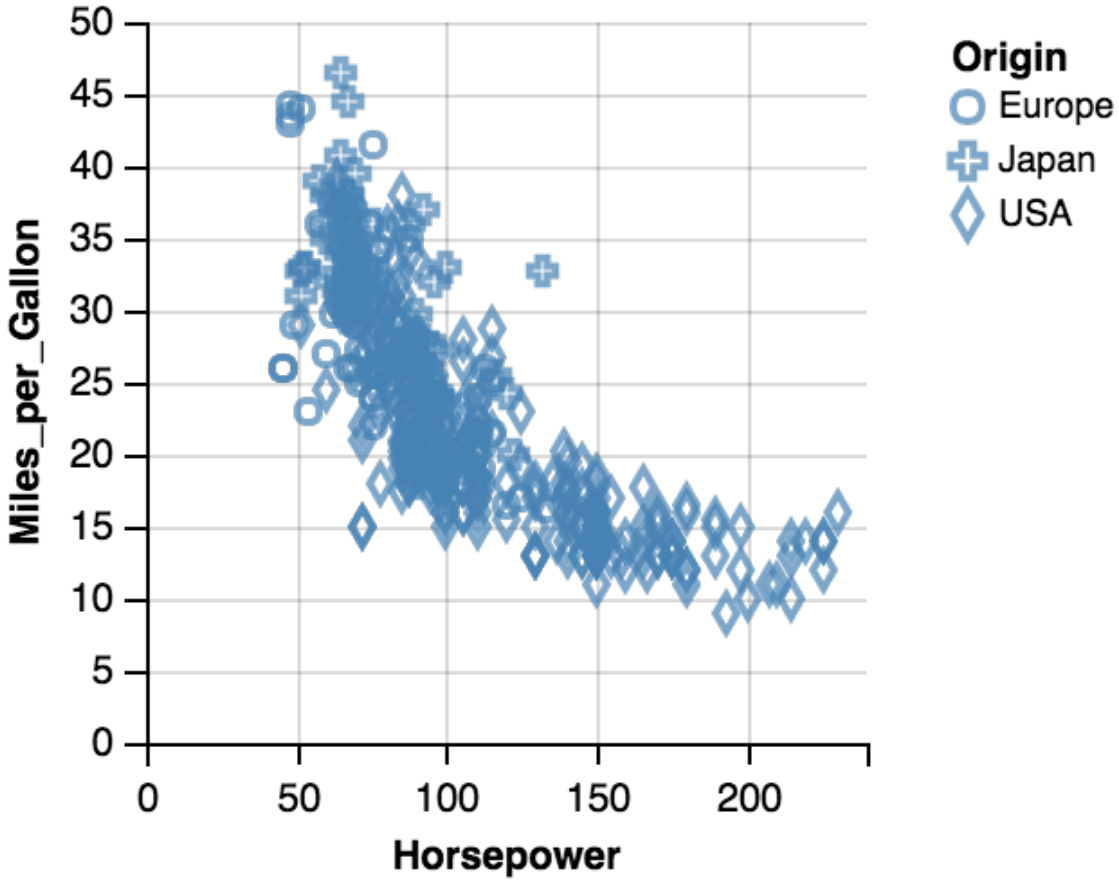
Covering the design space





Covering the design space





Vega-Lite

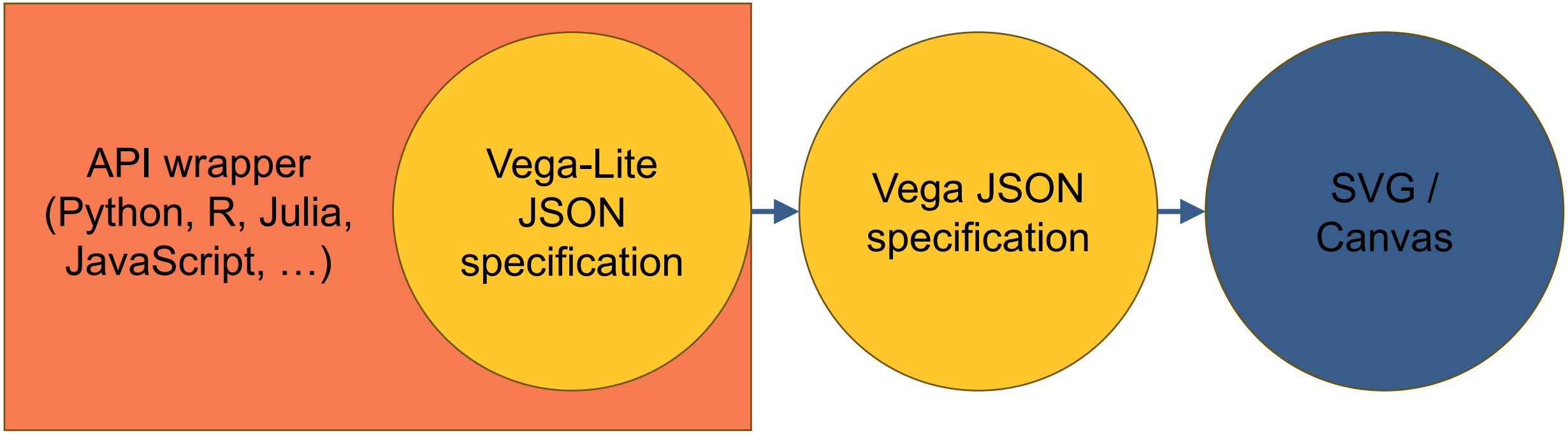
- JSON-based language to specify visualizations.
- Concise syntax for rapidly creating visualizations.
- JSON files can serve as a file format for creating and sharing visualizations.

Vega-Lite specification

- **Data** source
- Data **transformation** (optional)
- Graphical **mark** type
- **Encoding** / mapping between data and encoding channels (x, y, color, etc.)
- Faceting

```
{
    "data": {
        ...
    },
    "mark": ...,
    "encoding": {
        ...
    }
}
```

Vega-Lite / Vega / D3



Vega-Lite specification

Data

a	b
C	2
C	7
C	4
D	1
D	2
D	6
E	8
E	4
E	7

```
[
  {"a": "C", "b": 2},
  {"a": "C", "b": 7},
  {"a": "C", "b": 4},
  {"a": "D", "b": 1},
  {"a": "D", "b": 2},
  {"a": "D", "b": 6},
  {"a": "E", "b": 8},
  {"a": "E", "b": 4},
  {"a": "E", "b": 7}
]
```

Vega-Lite specification

Data

a	b
C	2
C	7
C	4
D	1
D	2
D	6
E	8
E	4
E	7

```
[
  {"a": "C", "b": 2},
  {"a": "C", "b": 7},
  {"a": "C", "b": 4},
  {"a": "D", "b": 1},
  {"a": "D", "b": 2},
  {"a": "D", "b": 6},
  {"a": "E", "b": 8},
  {"a": "E", "b": 4},
  {"a": "E", "b": 7}
]
```

JSON

Vega-Lite specification

Data

a	b
C	2
C	7
C	4
D	1
D	2
D	6
E	8
E	4
E	7

```
{
    "data": {
      "values": [
        {"a": "C", "b": 2},
        {"a": "C", "b": 7},
        {"a": "C", "b": 4},
        {"a": "D", "b": 1},
        {"a": "D", "b": 2},
        {"a": "D", "b": 6},
        {"a": "E", "b": 8},
        {"a": "E", "b": 4},
        {"a": "E", "b": 7}
      ]
    }
  }
```

JSON within Vega-Lite

Vega-Lite specification

```
{
    "data": {
      "values": [
        {"a": "C", "b": 2}, {"a": "C", "b": 7}, {"a": "C", "b": 4},
        {"a": "D", "b": 1}, {"a": "D", "b": 2}, {"a": "D", "b": 6},
        {"a": "E", "b": 8}, {"a": "E", "b": 4}, {"a": "E", "b": 7}
      ]
    },
    "mark": "point",
    "encoding": {}
  }
```

Vega-Lite specification

```
{
    "data": {
      "values": [
        {"a": "C", "b": 2}, {"a": "C", "b": 7}, {"a": "C", "b": 4},
        {"a": "D", "b": 1}, {"a": "D", "b": 2}, {"a": "D", "b": 6},
        {"a": "E", "b": 8}, {"a": "E", "b": 4}, {"a": "E", "b": 7}
      ]
    },
    "mark": "point",
    "encoding": {}
  }
```

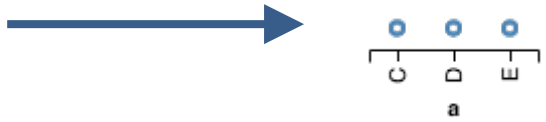
Vega-Lite specification

```
{
    "data": {
      "values": [
        {"a": "C", "b": 2}, {"a": "C", "b": 7}, {"a": "C", "b": 4},
        {"a": "D", "b": 1}, {"a": "D", "b": 2}, {"a": "D", "b": 6},
        {"a": "E", "b": 8}, {"a": "E", "b": 4}, {"a": "E", "b": 7}
      ]
    },
    "mark": "point",
    "encoding": {
      "x": {"field": "a", "type": "nominal"}
    }
  }
```

Mapping nominal field “a” to channel “x” (x-position of the point mark).

Vega-Lite specification

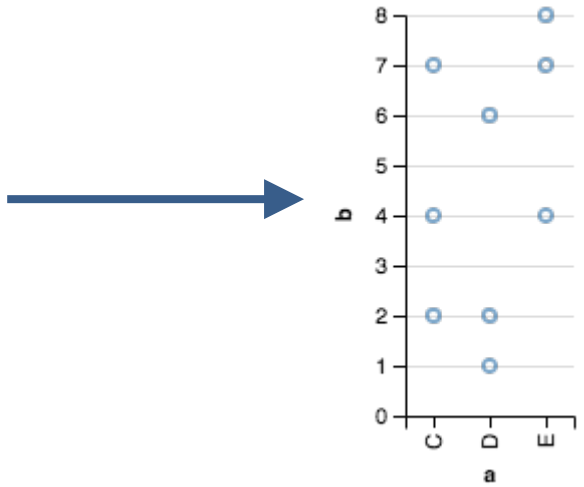
```
{
    "data": {
      "values": [
        {"a": "C", "b": 2}, {"a": "C", "b": 7}, {"a": "C", "b": 4},
        {"a": "D", "b": 1}, {"a": "D", "b": 2}, {"a": "D", "b": 6},
        {"a": "E", "b": 8}, {"a": "E", "b": 4}, {"a": "E", "b": 7}
      ]
    },
    "mark": "point",
    "encoding": {
      "x": {"field": "a", "type": "nominal"}
    }
  }
```



Mapping nominal field “a” to channel “x” (x-position of the point mark).

Vega-Lite specification

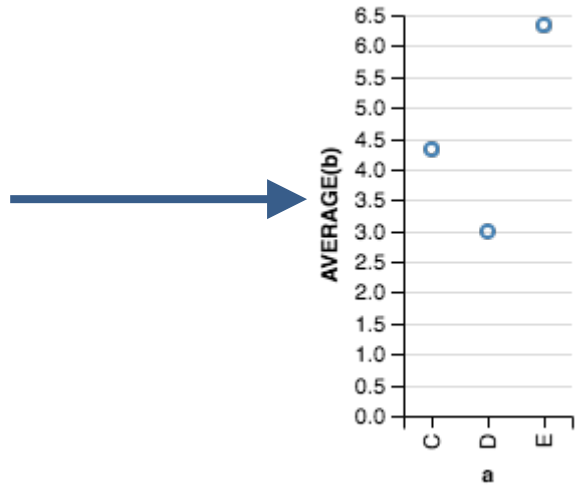
```
{
    "data": {
      "values": [
        {"a": "C", "b": 2}, {"a": "C", "b": 7}, {"a": "C", "b": 4},
        {"a": "D", "b": 1}, {"a": "D", "b": 2}, {"a": "D", "b": 6},
        {"a": "E", "b": 8}, {"a": "E", "b": 4}, {"a": "E", "b": 7}
      ]
    },
    "mark": "point",
    "encoding": {
      "x": {"field": "a", "type": "nominal"},
      "y": {"field": "b", "type": "quantitative"}
    }
  }
```



Mapping nominal field “b” to channel “y” (y-position of the point mark).

Vega-Lite specification

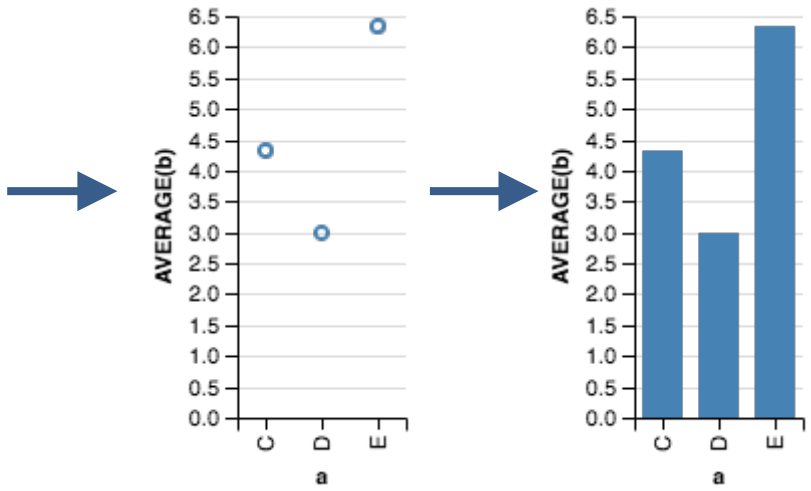
```
{
    "data": {
      "values": [
        {"a": "C", "b": 2}, {"a": "C", "b": 7}, {"a": "C", "b": 4},
        {"a": "D", "b": 1}, {"a": "D", "b": 2}, {"a": "D", "b": 6},
        {"a": "E", "b": 8}, {"a": "E", "b": 4}, {"a": "E", "b": 7}
      ]
    },
    "mark": "point",
    "encoding": {
      "x": {"field": "a", "type": "nominal"},
      "y": {"aggregate": "mean", "field": "b", "type": "quantitative"}
    }
  }
```



Mapping the mean of “b” to channel “y” (y-position of the point mark).

Vega-Lite specification

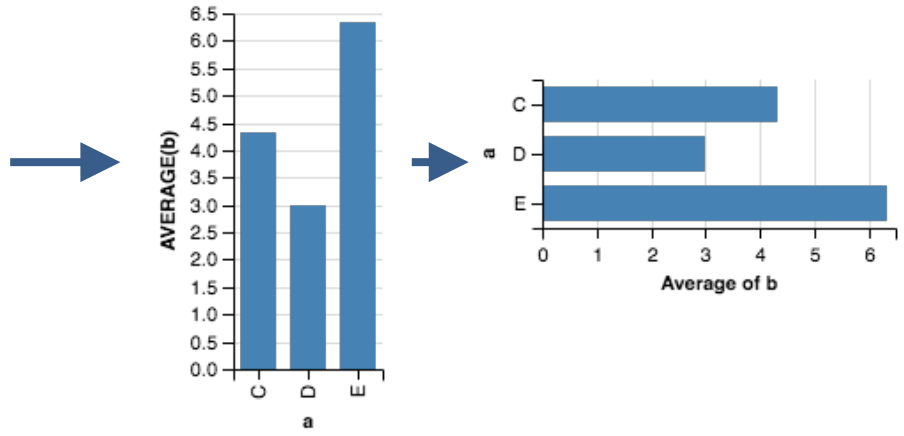
```
{
    "data": {
      "values": [
        {"a": "C", "b": 2}, {"a": "C", "b": 7}, {"a": "C", "b": 4},
        {"a": "D", "b": 1}, {"a": "D", "b": 2}, {"a": "D", "b": 6},
        {"a": "E", "b": 8}, {"a": "E", "b": 4}, {"a": "E", "b": 7}
      ]
    },
    "mark": “bar",
    "encoding": {
      "x": {"field": "a", "type": "nominal"},
      "y": {"aggregate": "mean", "field": "b", "type": "quantitative"}
    }
  }
```



Making a bar chart.

Vega-Lite specification

```
{
    "data": {
      "values": [
        {"a": "C", "b": 2}, {"a": "C", "b": 7}, {"a": "C", "b": 4},
        {"a": "D", "b": 1}, {"a": "D", "b": 2}, {"a": "D", "b": 6},
        {"a": "E", "b": 8}, {"a": "E", "b": 4}, {"a": "E", "b": 7}
      ]
    },
    "mark": "point",
    "encoding": {
      "y": {"field": "a", "type": "nominal"},
      "x": {"aggregate": "mean", "field": "b", "type": "quantitative"}
    }
  }
```



Transposing the bar chart.

Embedding Vega-Lite

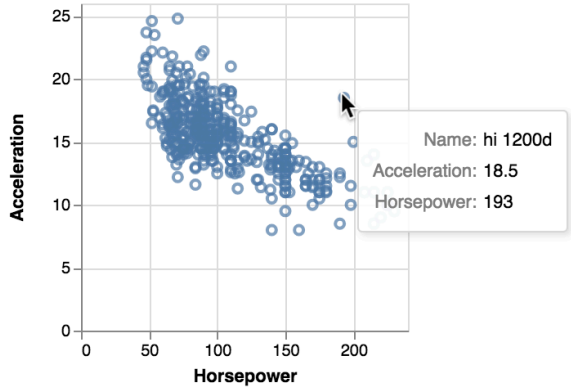
Vega-Embed allows you to embed Vega-Lite specifications on a web page.

```
<!doctype html>
<html>
  <head>
    <title>Embedding Vega-Lite</title>
    <script src="https://cdn.jsdelivr.net/npm/vega@5.25.0"></script>
    <script src="https://cdn.jsdelivr.net/npm/vega-lite@5.15.1"></script>
    <script src="https://cdn.jsdelivr.net/npm/vega-embed@6.22.2"></script>
  </head>
  <body>
    <div id="vis"></div>

    <script type="text/javascript">
      var yourVlSpec = {
        $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: 'ordinal'},
          y: {field: 'b', type: 'quantitative'}
        }
      };
      vegaEmbed('#vis', yourVlSpec);
    </script>
  </body>
</html>
```

Embedding Vega-Lite

IPython Vega allows you to embed Vega-Lite on a Jupyter Notebook.

```
In [3]: from vega import VegaLite

In [7]: VegaLite({
          "mark": "point",
          "encoding": {
            "y": {"type": "quantitative", "field": "Acceleration"},
            "x": {"type": "quantitative", "field": "Horsepower"},
            "tooltip": [
                {"type": "nominal", "field": "Name"},
                {"type": "quantitative", "field": "Acceleration"},
                {"type": "quantitative", "field": "Horsepower"}
            ]
          }
        }, df)
```



Export as PNG View Source Open in Vega Editor