Building Blocks for composing an expressive range of graphics
Grammar of Graphics

Data
Input data source to visualize.

Transform
Filter, aggregation, binning, etc.

Mark
Data-representative graphics.

Encoding
Mapping between data and mark properties.

Scale
Functions that map data values to visual values.

Guides
Axes & legends that visualize scales.
Grammar of Graphics for Customized Designs

Offer fine-grained control for composing interactive graphics.

But require verbose specifications and technical expertise.
Grammar of Graphics for Exploration

Facilitate rapid exploration with concise specifications by omitting low-level details.

Infer sensible defaults and allow customization by overriding defaults.

But limited support for interactions.
Vega-Lite's Mission

Facilitate exploratory data analysis with an expressive yet concise language to specify interactive multi-view graphics.
Vega-Lite: a Grammar of Graphics

- **Histogram**
- **Multi-series Line Chart**
- **Stripplot**
- **Slope Graph**
- **Binned Scatterplot**
- **Area Chart**
Vega-Lite: a Grammar of Multi-View Graphics
Vega-Lite: a Grammar of Interactive Multi-View Graphics

Indexed Chart

Focus+Context

Cross-filtering
Vega-Lite: a Grammar of Interactive Graphics

The Design of Vega-Lite

- **Single View** Specification
- **Layered** and **Multi-view** Composition
- **Interactions** with Selections

Using Vega-Lite

- **Programming** with Vega-Lite
- Higher-level **Tools** and **Recommendations**
Specifying Single Views

Abstract Data  Visual Representation
### Specifying Single Views

#### Weather Data for Seattle

<table>
<thead>
<tr>
<th>date</th>
<th>temperature</th>
<th>precipitation</th>
<th>weather</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/1</td>
<td>10.6</td>
<td>10.9</td>
<td>&quot;rain&quot;</td>
</tr>
<tr>
<td>1/2</td>
<td>11.7</td>
<td>0.8</td>
<td>&quot;drizzle&quot;</td>
</tr>
<tr>
<td>1/3</td>
<td>12.2</td>
<td>10.2</td>
<td>&quot;rain&quot;</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

#### Visual Representation

Strip Plot of Temperature

![Strip Plot of Temperature](image)
Strip Plot = (Tick with x=field)

```json
{
  data: {url: "weather-seattle.json"},
  mark: "tick",
  encoding: {
    x: {
      field: "temperature",
      type: "quantitative"
    }
  }
}
```

Vega-Lite is portable JSON specification
Strip Plot: Default Scales and Axes

```json
{
  data: {url: "weather-seattle.json"},
  mark: "tick",
  encoding: {
    x: {
      field: "temperature",
      type: "quantitative",
      scale: {type: "linear", domain: [0, 8], ...}
    }
    axis: {title: "temp", grid: true, ...}
  }
}
```
Strip Plot

```json
{
  data: {url: "weather-seattle.json"},
  mark: "tick",
  encoding: {
    x: {
      field: "temperature",
      type: "quantitative"
    }
  }
}
```

How many days?

---

`temperature`
Histogram = (Bar with \( x = \text{binned field} \), \( y = \text{count} \))
Histogram = (Bar with \( x = \text{binned field} \), \( y = \text{count} \))

```json
{
  data: {url: "weather-seattle.json"},
  mark: "tick",
  encoding: {
    x: {
      field: "temperature",
      type: "quantitative"
    }
  }
}
```
Histogram = (Bar with $x=binned\ field$, $y=count$)

```
{
  data: {url: "weather-seattle.json"},
  mark: "tick",
  encoding: {
    x: {
      bin: true,
      field: "temperature",
      type: "quantitative"
    }
  }
}
```
Histogram = \textbf{(Bar with } \textit{x}=\text{binned field, y}=\text{count)}

```json
{
  data: {url: "weather-seattle.json"},
  mark: "tick",
  encoding: {
    x: {
      bin: true,
      field: "temperature",
      type: "quantitative"
    },
    y: {
      aggregate: "count",
      type: "quantitative"
    }
  }
}
```
Histogram = (Bar with $x=binned$ field, $y=count$)

```javascript
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {
      bin: true,
      field: "temperature",
      type: "quantitative"
    },
    y: {
      aggregate: "count",
      type: "quantitative"
    }
  }
}
```
Histogram = (Bar with $x=$ binned field, $y=$ count)

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

Channel determines guide and bin parameters

<table>
<thead>
<tr>
<th>Guide</th>
<th>Color/Opacity/Shape</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td># of Bins</td>
<td>Legend with range labels</td>
<td>Quantitative axis</td>
</tr>
<tr>
<td></td>
<td>Fewer bins</td>
<td>More bins</td>
</tr>
</tbody>
</table>

**Hottest Temperature**

- -10–40
- 0–10
- 10–20
- 20–30
- 30–40

![Bar chart showing binning for temperature](image)
Histogram

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

```json
{
  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"
    }
}
```
Histogram + Color

```json
{
  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",
      scale: {
        "domain": ["sun", "fog", "drizzle", "rain", "snow"],
        "range": ["#e7ba52", "#c7c7c7", "#aec7e8", "#1f77b4", "#9467bd"]
      }
    }
  }
}
```
Histogram + Color = Stacked Histogram

```json
{
  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",
      ...
    }
  }
}
```
Stacked Histogram: **Sensible Defaults**

Channel (color) + Mark (bar) automatically enables stacking: a layout transform.

![Stacked Histogram Diagrams](image)
Stacked Histogram: **Sensible Defaults**

Channel (color) + Mark (bar) automatically enables stacking: a layout transform.

no stack $\rightarrow$ overlap

stack (default)
Histogram + Color = Stacked Histogram

```json
{
  "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"
    }
  }
}
```

hard to compare without common baseline
Histogram + Color = Stacked Histogram

```json
{
  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"}
  }
}
```
Histogram + Column = Trellis Histogram

```json
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {bin: true, field: "temperature", type: "quantitative"},
    y: {aggregate: "count", type: "quantitative"},
    column: {field: "weather", type: "nominal"}
  }
}
```
Vega-Lite: a Grammar of Interactive Graphics

The Design of Vega-Lite

Single View Specification

Layered and Multi-view Composition

Interactions with Selections

Using Vega-Lite

Programming with Vega-Lite

Higher-level Tools and Recommendations
View Composition Operators

facet row: C

\[ \begin{align*}
&= \begin{cases}
  c_1 \\
  c_2
\end{cases}
\end{align*}\]

vconcat: [ \begin{cases}
  \end{cases}, \begin{cases}
  \end{cases} \] =

layer: [ \begin{cases}
  \end{cases}, \begin{cases}
  \end{cases} \] =

repeat row: [A,B]

\[ \begin{align*}
&= \begin{cases}
  \end{cases}
\end{align*}\]
Monthly Precipitation

```json
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {
      timeUnit: "month",
      field: "date",
      type: "ordinal"
    },
    y: {
      aggregate: "mean",
      field: "precipitation",
      type: "quantitative"
    }
  }
}
```
Layering

```json
{
  data: {url: "weather-seattle.json"},
  layer: [{
    mark: "bar",
    encoding: {
      x: {
        timeUnit: "month",
        field: "date",
        type: "ordinal"
      },
      y: {
        aggregate: "mean",
        field: "precipitation",
        type: "quantitative"
      }
    }
  }]
}
```
Layering

```json
{
  data: {url: "weather-seattle.json"},
  layer: [{
    mark: "bar",
    encoding: {
      x: {
        timeUnit: "month",
        field: "date",
        type: "quantitative"
      },
      y: {
        aggregate: "mean",
        field: "precipitation",
        type: "quantitative"
      }
    }
  }, {
    mark: "rule",
    encoding: {
      y: {
        aggregate: "mean",
        field: "precipitation",
        type: "quantitative"
      }
    }
  }]
}
```
Concat

```json
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {
      timeUnit: "month",
      field: "date",
      type: "ordinal"
    },
    y: {
      aggregate: "mean",
      field: "precipitation",
      type: "quantitative"
    }
  }
}
```
Concat

```json
{
  data: {
    url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {
      timeUnit: "month",
      field: "date",
      type: "ordinal"
    },
    y: {
      aggregate: "mean",
      field: "precipitation",
      type: "quantitative"
    }
  }
}
```
```json
{
    vconcat: [ {
        data: {url: "weather-seattle.json"},
        mark: "bar",
        encoding: {
            x: {
                timeUnit: "month",
                field: "date",
                type: "quantitative"
            },
            y: {
                aggregate: "mean",
                field: "precipitation",
                type: "ordinal"
            }
        }
    },
    {
        data: {url: "weather-seattle.json"},
        mark: "bar",
        encoding: {
            x: {
                timeUnit: "month",
                field: "date",
                type: "ordinal"
            },
            y: {
                aggregate: "mean",
                field: "temperature",
                type: "quantitative"
            }
        }
    }
]}
```
Repeat

```json
{
  data: {url: "weather-seattle.json"},
  mark: "bar",
  encoding: {
    x: {
      timeUnit: "month",
      field: "date",
      type: "ordinal"
    },
    y: {
      aggregate: "mean",
      field: "precipitation",
      type: "quantitative"
    }
  }
}
```
```
Repeat

```
```python
{
    repeat: {
        column: [
            "precipitation",
            "temperature",
            "wind"
        ],

    },

    spec: {
        data: {
            url: "weather-seattle.json"
        },

        mark: "bar",

        encoding: {
            x: {
                timeUnit: "month",
                field: "date",
                type: "ordinal"
            },

            y: {
                aggregate: "mean",
                field: {repeat: "column"},
                type: "quantitative"
            }
        }
    }
}
```
Repeat

```json
{
  repeat: {
    column: ["precipitation", "temperature", "wind"]
  },
  spec: {
    data: {url: "weather-seattle.json"},
    mark: "bar",
    encoding: {
      x: {
        timeUnit: "month",
        field: "date",
        type: "ordinal"
      },
      y: {
        aggregate: "mean",
        field: {repeat: "column"},
        type: "quantitative"
      }
    }
  }
}
```
Repeat: SPLOM

```
{ repeat: {
    column: [
        "temperature",
        "precipitation",
        "wind"],
    row: [
        "wind",
        "precipitation",
        "temperature",
    ]
},
spec: {
}
}```
Dashboards
Many views, some of them composed
Manual data, and view management
Composition
Hierarchical View Composition

Facet

Weather

Temp, Prec, Wind

Prec, Temp, Wind

Layer

Repeat

Layer
Hierarchical View Composition

Facet

Repeat

Layer

Facet Weather

Repeat Temp, Prec, Wind

Layer Prec, Temp, Wind

HConcat
Hierarchical View Composition

- **VConcat**
- **HConcat**
- **Repeat** Prec, Temp, Wind
- **Facet**
  - **Weather**
- **Repeat** Temp, Prec, Wind
- **Layer**
Hierarchical View Composition

VConcat

HConcat

Repeat
Prec, Temp, Wind

Facet
Weather

Repeat
Temp, Prec, Wind

Layer
Layer + Repeat

```json
{  
  Repeat
  Prec, Temp, Wind 'precipitation", "temperature", "wind"
},
spec: {
  Layer
}
}
Vega-Lite: a Grammar of Interactive Graphics

The Design of Vega-Lite

- Single View Specification
- Layered and Multi-view Composition
- Interactions with Selections

Using Vega-Lite

- Programming with Vega-Lite
- Higher-level Tools and Recommendations
Selections
Selections

The core interactive building blocks. Define three components:

1. Event processing – how does the interaction occur?
2. Points of interest – which marks/data points were interacted with?
3. Predicate function – what is the full set of selected marks/data points?
Vega-Lite Selections

```json
{
    data: {url: "data/cars.json"},
    mark: "circle",
    encoding: {
        x: {field: "Horsepower", type: "Q"},
        y: {field: "Miles_per_Gallon", type: "Q"},
        color: {field: "Origin", type: "N"}
    }
}
```

Selections define event processing, points of interest, and a predicate function.
Vega-Lite Selections: A Single Point

```
{
  data: {url: "data/cars.json"},
  mark: "circle",
  selection: {
    picked: {type: "single"}
  },
  encoding: {
    x: {field: "Horsepower", type: "Q"},
    y: {field: "Miles_per_Gallon", type: "Q"},
    color: {field: "Origin", type: "N"}
  }
}
```

**Selections** define event processing, points of interest, and a predicate function.
Vega-Lite Selections: A Single Point

```json
{
  data: {url: "data/cars.json"},
  mark: "circle",
  selection: {
    picked: {type: "single"}
  },
  encoding: {
    x: {field: "Horsepower", type: "Q"},
    y: {field: "Miles_per_Gallon", type: "Q"},
    color: {
      condition: {
        selection: "!picked", value: "grey"
      },
      field: "Origin", type: "N"
    }
  }
}
```

Selections define event processing, points of interest, and a predicate function.

Selection types provide defaults values for these three components.
Selections define event processing, points of interest, and a predicate function.

Selection types provide defaults values for these three components.
Vega-Lite Selections: Multiple Points on hover

```
{
  data: {url: "data/cars.json"},
  mark: "circle",
  selection: {
    picked: {type: "multi", on: "mouseover"}
  },
  encoding: {
    x: {field: "Horsepower", type: "Q"},
    y: {field: "Miles_per_Gallon", type: "Q"},
    color: {
      condition: {
        selection: "!picked", value: "grey"
      },
      field: "Origin", type: "N"
    }
  }
}
```

**Selections** define event processing, points of interest, and a predicate function.

Selection **types** provide defaults values for these three components.
Selections

The core interactive building blocks. Define three components:

1. Event processing – how does the interaction occur?
2. Points of interest – which marks/data points were interacted with?
3. Predicate function – what is the full set of selected marks/data points?
The core interactive building blocks. Define three components:

1. **Event processing** – how does the interaction occur?
2. **Points of interest** – which marks/data points were interacted with?
3. **Predicate function** – what is the full set of selected marks/data points?

**Selection Transforms**

Composable operators that modify a selection's components.
Vega-Lite Selections: A Single Cylinder

```
{
  data: {url: "data/cars.json"},
  mark: "circle",
  selection: {
    picked: {
      type: "single",
      fields: ["Cylinders"]
    }
  },
  encoding: {
    x: {field: "Horsepower", type: "Q"},
    y: {field: "Miles_per_Gallon", type: "Q"},
    color: {
      condition: {
        selection: "!picked",
        value: "grey"
      },
      field: "Origin", type: "N"
    }
  }
}
```

The **project transform** rewrites the predicate to match on **fields** or encodings.
Vega-Lite Selections: **A Single Cylinder & Year**

```json
{
  data: {url: "data/cars.json"},
  mark: "circle",
  selection: {
    picked: {
      type: "single",
      fields: ["Cylinders"],
      bind: {input: "range", min: 3, ...}
    }
  },
  encoding: {
    x: {field: "Horsepower", type: "Q"},
    y: {field: "Miles_per_Gallon", type: "Q"},
    color: {
      condition: {
        selection: "!picked",
        value: "grey"
      },
      field: "Origin", type: "N"
    }
  }
}
```

The **bind transform** drives selections via **query widgets** and scale functions.
Vega-Lite Selections: A Single Cylinder & Year

```json
{
  data: {url: "data/cars.json"},
  mark: "circle",
  selection: {
    picked: {
      type: "single",
      fields: ["Cylinders", "Year"],
      bind: {
        Cylinders: {input: "range", min: 3},
        Year: {input: "range", min: 1967}
      }
    }
  },
  encoding: {
    x: {field: "Horsepower", type: "Q"},
    y: {field: "Miles_per_Gallon", type: "Q"},
    color: {
      condition: {
        selection: "!picked", value: "grey"
      },
      field: "Origin", type: "N"
    }
  }
}
```
Vega-Lite Selections: Continuous Region

```json
{
  data: {url: "data/cars.json"},
  mark: "circle",
  selection: {
    grid: {type: "interval"}
  },
  encoding: {
    x: {field: "Horsepower", type: "Q"},
    y: {field: "Miles_per_Gallon", type: "Q"},
    color: {
      condition: {
        selection: "!grid", value: "grey"
      },
      field: "Origin", type: "N"
    }
  }
}
```
Vega-Lite Selections: Continuous Region

```json
{
  data: {url: "data/cars.json"},
  mark: "circle",
  selection: {
    grid: {
      type: "interval",
      encodings: ["x"]
    }
  },
  encoding: {
    x: {field: "Horsepower", type: "Q"},
    y: {field: "Miles_per_Gallon", type: "Q"},
    color: {
      condition: {
        selection: "!grid", value: "grey"
      },
      field: "Origin", type: "N"
    }
  }
}
```
Vega-Lite Selections: Continuous Region

```
{
  data: {url: "data/cars.json"},
  mark: "circle",
  selection: {
    grid: {
      type: "interval",
      encodings: ["x"],
      bind: "scales"
    }
  },
  encoding: {
    x: {field: "Horsepower", type: "Q"},
    y: {field: "Miles_per_Gallon", type: "Q"},
    color: {
      condition: {
        selection: "!grid", value: "grey"
      },
      field: "Origin", type: "N"
    }
  }
}
```
Vega-Lite Selections: Continuous Region

```
{
  data: {url: "data/cars.json"},
  mark: "circle",
  selection: {
    grid: {
      type: "interval",
      encodings: ["x"],
      bind: "scales"
    },
    encoding: {
      x: {
        field: "Horsepower", type: "Q",
        scale: {domain: {selection: "grid"}}
      },
      y: {field: "Miles_per_Gallon", type: "Q"},
      "color": {
        condition: {
          selection: "$!picked$", value: "grey"
        },
        "field": "Origin", "type": "N"
      }
    }
  }
}
```

**Two-way binding** between a selection and scales:
- Selection is populated with scale domains.
- Selection now drives scale domains.
Vega-Lite Selections: Continuous Region

```json
{
  data: {url: "data/cars.json"},
  mark: "circle",
  selection: {
    grid: {
      type: "interval",
      encodings: ["x"],
      bind: "scales"
    }
  },
  encoding: {
    x: {field: "Horsepower", type: "Q"},
    y: {field: "Miles_per_Gallon", type: "Q"},
    color: {
      condition: {
        selection: "!grid", value: "grey"
      },
      field: "Origin", type: "N"
    }
  }
}
```
```json
{
  data: {url: "data/cars.json"},
  repeat: {
    row: ["Displacement", "Miles_per_Gallon"],
    column: ["Horsepower", "Miles_per_Gallon"]
  },
  spec: {
    mark: "circle",
    selection: {
      grid: {
        type: "interval",
        encodings: ["x"],
        bind: "scales"
      }
    },
    encoding: {
      x: {field: "Horsepower", type: "Q"},
      y: {field: "Miles_per_Gallon", type: "Q"},
      color: {
        condition: {
          selection: "!grid", value: "grey"
        },
        field: "Origin", type: "N"
      }
    }
  }
}
```
Vega-Lite Overview+Detail

```json
{
  data: {url: "data/sp500.csv", ...},
  vconcat: [
    { mark: "area",
      encoding: {
        x: {field: "date", type: "T", ...},
        y: {field: "price", type: "Q", ...}
      }
    },
    { mark: "area",
      encoding: {
        x: {field: "date", type: "T", ...},
        y: {field: "price", type: "Q", ...}
      }
    }
  ]
}
```
Vega-Lite Overview+Detail

```json
{
  data: {url: "data/sp500.csv", ...},
  vconcat: [
    {
      mark: "area",
      selection: {
        region: {
          type: "interval", encodings: ["x"]
        }
      }
    },
    ...
  ],
  mark: "area",
  encoding: {
    x: {field: "date", type: "T", ...},
    y: {field: "price", type: "Q", ...}
  }
}
```
Vega-Lite Overview+Detail

```
{
  data: {url: "data/sp500.csv", ...},
  vconcat: [{
    mark: "area",
    selection: {
      region: {
        type: "interval", encodings: ["x"]
      }
    },
  }, ...],
  mark: "area",
  encoding: {
    x: {
      field: "date", type: "T",
      scale: {domain: {selection: "region"}}
    },
    y: {field: "price", type: "Q", ...}
  }
}
```
Vega-Lite Layered Cross Filtering

```
{
  data: {url: "data/flights.json"},
  mark: "bar",
  encoding: {
    x: {field: "hour", type: "Q", bin: true},
    y: {aggregate: "count", type: "Q"}
  }
}
```
Vega-Lite Layered Cross Filtering

```json
{
  repeat: {column: ["hour", "delay", "distance"]},
  spec: {
    data: {url: "data/flights.json"},
    mark: "bar",
    encoding: {
      x: {field: {repeat: "column"}, type: "Q", bin: true},
      y: {aggregate: "count", field: "*", type: "Q"}
    }
  }
}
```
Vega-Lite Layered Cross Filtering

```json
{
  repeat: {column: ["hour", "delay", "distance"]},
  spec: {
    layer: [{
      data: {url: "data/flights.json"},
      mark: "bar",
      encoding: {
        x: {field: {repeat: "column"}, type: "Q", bin: true},
        y: {aggregate: "count", type: "Q"}
      }
    }, {
      ...,
      color: {value: "goldenrod"}
    }]
  }
}
```
Vega-Lite Layered Cross Filtering

```json
{
  repeat: {column: ["hour", "delay", "distance"]},
  spec: {
    layer: [{
      ...,
      selection: {
        region: {type: "interval", encodings: ["x"]}
      }
    }, {
      ...,
    }]
  }
}
```
Vega-Lite Layered Cross Filtering

```json

{  repeat: {column: ["hour", "delay", "distance"]},  spec: {    layer: [{      ...      selection: {        region: {type: "interval", encodings: ["x"]}      },      ...      transform: [{filter: {selection: "region"}}]    }]  }}

```

35 Lines of JSON!
Vega-Lite: a Grammar of Interactive Graphics

The Design of Vega-Lite
   Single View Specification
   Layered and Multi-view Composition
   Interactions with Selections

Using Vega-Lite
   Programming with Vega-Lite
   Higher-level Tools and Recommendations
Vega-Lite: a Grammar of Interactive Graphics

The Design of Vega-Lite
- Single View Specification
- Layered and Multi-view Composition
- Interactions with Selections

Using Vega-Lite
- Programming with Vega-Lite
- Higher-level Tools and Recommendations
Using Vega-Lite

Compile to Vega and use Vega's runtime

- Retarget to different renderers (Web-based/Server, Canvas/SVG)
- Support streaming data

Declarative JSON Syntax

- Serve as file format
- Bindings for different languages
Altair is a declarative statistical visualization library for Python, based on Vega-Lite.

With Altair, you can spend more time understanding your data and its meaning. Altair’s API is simple, friendly and consistent and built on top of the powerful Vega-Lite visualization grammar. This elegant simplicity produces beautiful and effective visualizations with a minimal amount of code.
Histogram in Altair

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

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

from altair import Chart, expr
data_weather = expr.DataFrame('data/weather-seattle.json')Chart(data_weather).mark_bar() .encode(
  x=X(bin=True, field='Temperature'),
  y=Y(aggregate='count')
)

Altair's API is automatically generated from the Vega-Lite JSON schema.
“It is this type of 1:1:1 mapping between thinking, code, and visualization that is my favorite thing about [Altair]”

– Dan Saber.

https://dansaber.wordpress.com/2016/10/02/a-dramatic-tour-through-pythons-data-visualization-landscape-including-ggplot-and-altair/
“We see this portion of the effort as much bigger than Altair itself: the Vega and Vega-Lite specifications are perhaps the best existing candidates for a principled lingua franca of data visualization” – Altair Team.
Vega-Lite: a Grammar of Interactive Graphics

The Design of Vega-Lite

Single View Specification
Layered and Multi-view Composition
Interactions with Selections

Using Vega-Lite

Programming with Vega-Lite
Higher-level Tools and Recommendations
Voyager

Augment manual specification with recommendation to **promote breadth & reduce tedium** in exploration.

Use Vega-Lite to **recommend** data and visual encodings.

https://github.com/vega/voyager
1. Browsing Univariate Summaries

```json
{
  mark: "tick",
  encoding: {
    x: {
      field: "Horsepower",
      type: "quantitative"
    }
  }
}
```
3. Related Views
3. Related Views
3. Related Views

```json
{
    mark: "tick",
    encoding: {
        x: {
            field: "Horsepower",
            type: "quantitative"
        }
    }
}
```
mark: "tick",
encoding: {
  x: {
    field: "Horsepower",
    type: "quantitative"
  }
}
3. Related Views

```json
{
  "mark": "tick",
  "encoding": {
    "x": {
      "fn": "bin/mean?",
      "field": "Horsepower",
      "type": "quantitative"
    }
  }
}
```
3. Related Views

```json
{
  mark: "tick",
  encoding: {
    x: {
      fn: "bin",
      field: "Horsepower",
      type: "quantitative"
    }
  }
}
```
3. Related Views

```json
{
  mark: "tick",
  encoding: {
    x: {
      fn: "bin",
      field: "Horsepower",
      type: "quantitative"
    }
  }
}
```
3. Related Views

```json
{
  mark: "tick",
  encoding: {
    x: {
      fn: "bin",
      field: "Horsepower",
      type: "quantitative"
    },
    y: {
      aggregate: "count",
      type: "quantitative"
    }
  }
}
```
3. Related Views

Enumerate & Rank

Encodings

```json
{
  mark: ?,
  encoding: {
    x: {
      fn: "bin",
      field: "Horsepower",
      type: "quantitative"
    },
    y: {
      aggregate: "count",
      type: "quantitative"
    }
  }
}
```
3. Related Views

Enumerate & Rank Encodings

```json
{
  mark: ?,
  encoding: {
    x: {
      fn: "bin",
      field: "Horsepower",
      type: "quantitative"
    },
    y: {
      aggregate: "count",
      type: "quantitative"
    }
  }
}
```
3. Related Views
4. Wildcard Specification
4. Wildcard Specification

```json
{  
  mark: "tick",
  encoding: {
    x: {
      field: "Horsepower",
      type: "quantitative"
    },
    y: {
      field: "?",
      type: "quantitative"
    }
  }
}

Enumerate Variable
```
Voyager

Augment manual specification with recommendation to promote breadth & reduce tedium in exploration

Use Vega-Lite to recommend data and visual encodings.

interactions?

https://github.com/vega/voyager
Interval

Repeat

Project

Bind

Multi

Repeat

Project

Bind

Hover

Nearest

Systematically enumerate the interaction design space
Vega-Lite: a Grammar of Interactive Graphics

The Design of Vega-Lite

Single View Specification
Layered and Multi-view Composition
Interactions with Selections

Using Vega-Lite

Programming with Vega-Lite
Higher-level Tools and Recommendations
Vega-Lite 2 beta
(including multi-view composition and interaction support)
vega.github.io/vega-lite/
Exploring Data

In this tutorial, you'll learn a few more techniques for creating visualizations in Vega-Lite. If you are not familiar with Vega-Lite, please read the getting started tutorial first.

For this tutorial, we will create visualizations to explore weather data for Seattle, taken from NOAA. The data file with columns for the temperature (in Celsius), precipitation (in centimeter), wind (in meter/second), and date has one row for each day from January 1st, 2012 to December 31st, 2015.

To load the CSV file with Vega-Lite, we need to provide a URL and set the format type in the data schema specification.

```
"data": {"url": "data/seattle-weather.csv"
```

Let's start by looking at the precipitation. Precipitation is a quantitative variable. Let's use a tick mark to represent the distribution of precipitation.

```

```

It looks as though precipitation is skewed towards lower values; that is, when it rains, it usually doesn't rain very much. To better see this, we can create a histogram of the precipitation data. For this, we have to add an encoding that uses a Vega field that is an aggregated count. It is difficult to see patterns across continuous time, so here are some visualizations that show the data over one year.

Table of Contents

Below is an overview of the documentation for Vega-Lite properties. See below for an overview of Vega-Lite specifications.

- Overview
- Spec
- Data
- Transform
- Mark
- Encoding
- Aggregate
- Bin
- Sort
- Time Unit
- Scale
- Axis
- Legend
- Config
- Customizing Size
- Faceting
- Layering
- Interactions

Vega-Lite is a high-level visualization grammar. It provides a concise JSON syntax for supporting rapid generation of visualizations to support analysis. Vega-Lite can serve as a declarative format for describing and creating data visualizations. Vega-Lite specifications can be compiled to a lower-level, more detailed Vega specifications and rendered using Vega’s compiler.

This documentation describes the JSON specification language and how to use Vega-Lite visualizations in a web application.
Vega-lite

```
{
  "data": {
    "url": "data/seattle-weather.csv",
    "format": {
      "type": "csv"
    }
  },
  "mark": "bar",
  "encoding": {
    "x": {
      "field": "date",
      "type": "temporal",
      "timeunit": "month",
      "axis": {
        "title": "Month of the year"
      }
    },
    "y": {
      "field": "*",
      "type": "quantitative",
      "aggregate": "count"
    },
    "color": {
      "field": "weather",
      "type": "nominal",
      "scale": {
        "domain": ["sun", "fog", "drizzle", "rain", "snow"],
        "range": ["#f7bd4c", "#7c7c7c", "#c7c7c7", "#e6d8d8", "#0f77b4", "#9467bd"]
      }
    },
    "legend": {
      "title": "Weather Type"
    }
  }
}
```
Contributors: Vega-Lite, Altair & Voyager

and many more...
Vega-Lite – A High-Level Visualization Grammar

Vega-Lite is a high-level visualization grammar. It provides a concise JSON syntax for supporting and generating visualizations to support analysts. Vega-Lite specifications can be compiled to Vega specifications.

Vega-Lite specifications describe visualizations as mappings from data to properties of graphical marks (e.g., points or bars). It automatically produces visualization components including axes, legends, and scales. It then determines properties of these components based on a set of carefully designed rules. This approach allows Vega-Lite specifications to be simple and expressive, but also promote user control. As Vega-Lite is designed for analysts, it supports data transformations such as aggregation, binning, filtering, sorting, and visual transformations including stacking and linking.

Read our introduction article on Medium, check out the documentation and take a look at our example gallery.

Example

This is a Vega-Lite specification to create a bar chart that shows the average temperature in Seattle for each month.

vega.github.io/vega-lite