

# Introduction to Vega-Lite



Gustavo Moreira, University of Illinois Chicago

**Marcos Lage, Universidade Federal Fluminense**

Nivan Ferreira, Universidade Federal de Pernambuco

**Fabio Miranda, University of Illinois Chicago**

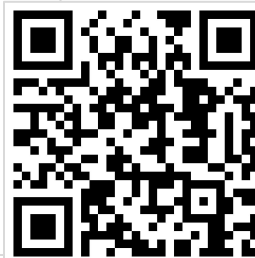


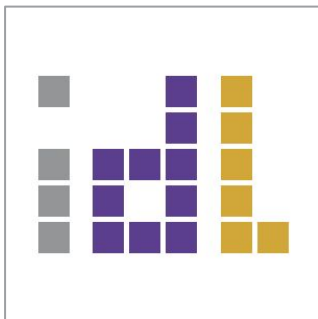
Vega-Lite specifications describe visualizations as encoding mappings from data to **properties of graphical marks** (e.g., points or bars). The Vega-Lite compiler **automatically produces visualization components** including axes, legends, and scales. It determines default properties of these components based on a set of **carefully designed rules**. This approach allows Vega-Lite specifications to be concise for quick visualization authoring, while giving user control to override defaults and customize various parts of a visualization. As we also designed Vega-Lite to support data analysis, Vega-Lite supports both **data transformations** (e.g., aggregation, binning, filtering, sorting) and **visual transformations** (e.g., stacking and faceting). Moreover, Vega-Lite specifications can be **composed** into layered and multi-view displays, and made **interactive with selections**.

For more information, read our [introduction article to Vega-Lite v2 on Medium](#), watch our [OpenVis Conf talk about the new features in Vega-Lite v2](#), see the [documentation](#) and take a look at our [example gallery](#). Follow us on [Twitter](#) at [@vega\\_vis](#) to stay informed about updates.

Latest Version: 5.16.0

Try online

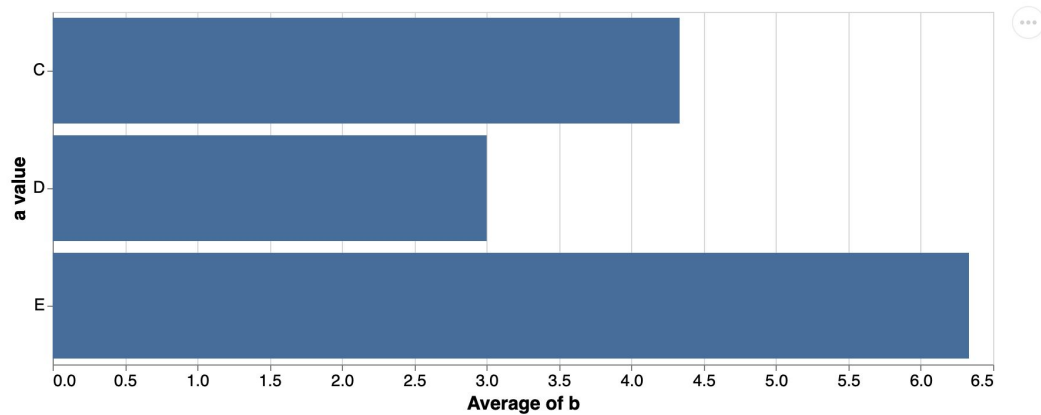




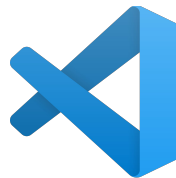
The development of Vega-Lite is led by the alumni and members of the [University of Washington Interactive Data Lab](#) (UW IDL), including [Kanit “Ham” Wongsuphasawat](#) (now at Databricks), [Dominik Moritz](#) (now at CMU / Apple), [Arvind Satyanarayan](#) (now at MIT), and [Jeffrey Heer](#) (UW IDL).



## Grammars for Vis Tutorial: SIBGRAPI 2023



**Hello World**



## Mark

[Edit this page](#)

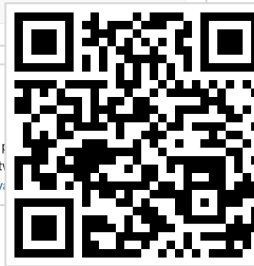
Marks are the basic visual building block of a visualization. They provide basic shapes whose properties (such as position, size, and color) can be used to visually encode data, either from a data field, or a constant value.

The `mark` property of a single view specification can either be (1) a string describing a mark type or (2) a mark definition object.

```
// Single View Specification
{
  "data": ... ,
  "mark": ... ,    // mark
  "encoding": ... ,
  ...
}
```

## Encoding

An integral part of the data visualization process is encoding data with visual properties. The `encoding` property of a single view specification represents the mapping between visual properties (such as `x`, `y`, or `color`) and data fields, constant visual values, or constant data values.



# 1. Marks and Encodings



## Composing Layered & Multi-view Plots

[Edit this page](#)

With Vega-Lite, you can not only create single view visualizations, but also [facet](#), [layer](#), [concatenate](#), and [repeat](#) these views into layered or multiview displays.

A layered or multi-view display can also be composed with other views. Through this hierarchical **composition**, you can create a whole dashboard as a single specification.

Vega-Lite's compiler infers how input data should be reused across constituent views, and domains should be unioned or remain independent.



## 2. Layers & Multi-view

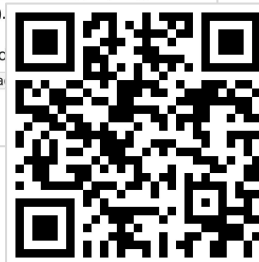


## Transformation

[Edit this page](#)

Data transformations in Vega-Lite are described via either **view-level** transforms (the `transform` property) or **field transforms inside** `encoding` (`bin`, `timeUnit`, `aggregate`, `sort`, and `stack`).

When both types of transforms are specified, the view-level `transform`s are executed in the array. Then the inline transforms are executed in this order: `bin`, `timeUnit`, `a`, `stack`.



## 3. Layers & Multi-view

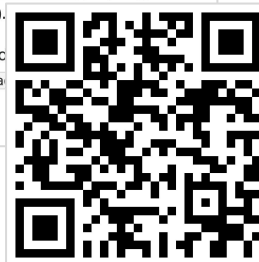


## Transformation

[Edit this page](#)

Data transformations in Vega-Lite are described via either **view-level** transforms (the `transform` property) or **field transforms inside** `encoding` (`bin`, `timeUnit`, `aggregate`, `sort`, and `stack`).

When both types of transforms are specified, the view-level `transform`s are executed in the array. Then the inline transforms are executed in this order: `bin`, `timeUnit`, `a`, `stack`.



## 4. Transformations



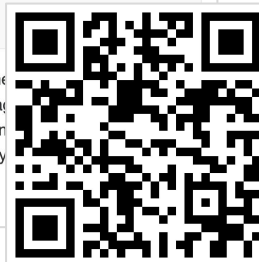


## Dynamic Behaviors with Parameters

[Edit this page](#)

```
// A Single View Specification
{
  ...,
  "params": [ // An array of named parameters.
    { "name": ..., ... }
  ],
  "mark": ...,
  "encoding": ...,
  ...
}
```

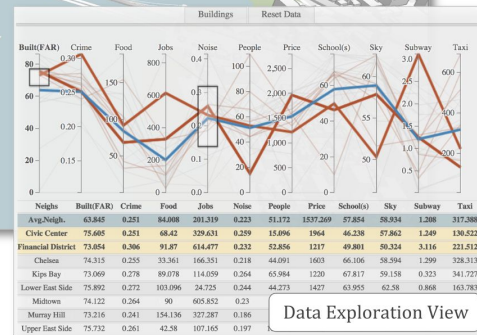
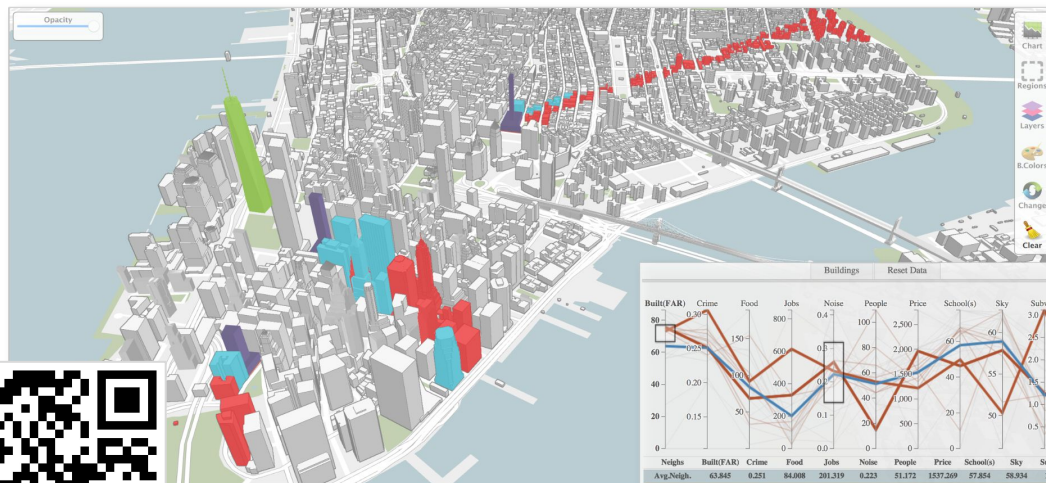
Parameters are the basic building block in Vega-Lite's *grammar of interaction*. Parameters can be simple *variables* or more complex *selections* that map user input (e.g., mouse clicks and drags) to data. Parameters can be used throughout the remainder of the chart specification to determine data points, determine data extents, or in expression strings. They can also optionally be used to create interactive widgets (e.g., sliders or drop down menus).



## 5. Interactions



## **6. The best stats you've ever seen, Hans Rosling**



## 7. Urbane, IEEE VIS 2015

# Introduction to Vega-Lite



Gustavo Moreira, University of Illinois Chicago

**Marcos Lage, Universidade Federal Fluminense**

Nivan Ferreira, Universidade Federal de Pernambuco

**Fabio Miranda, University of Illinois Chicago**

