Altair

Declarative Statistical Visualization in Python

SciPy 2016 Keynote
Brian E. Granger

Altair is developed and designed by:
Brian Granger (@ellisonbg)
Jake Vanderplas (@jakevdp)

https://github.com/ellisonbg/altair

Visualization in Python: Power

- Many powerful libraries for visualization in Python:
  - Matplotlib, Seaborn, Plotly, Bokeh, Lightning, Pandas,…
- Each of these libraries does certain things really well
- Ultimately, just about any visualization is possible
- Many of the APIs are inspired by Wilkinson's Grammar of Graphics

Visualization in Python: Pain

- Visualization is a persistent and significant source of user pain
  - “I have been using Matplotlib for a decade now, and I still have to look most things up”
  - “I love Python, but do my visualizations using R/ggplot2”

- Sparse handling of statistical visualization, categorical data, date/time data
- Users have to write code, even for incidental aspects of visualizations
- New (and experienced) users have to track and learn all of these libraries and make decisions about what library to use for a given task
- Many of the most basic things are still horribly complicated :(

Don’t Try This at Home

- Fire up the Jupyter/IPython notebook and import matplotlib
- Grab a DataFrame of your choice
- Make a 2d scatter plot of two quantitative columns
- Facet your scatter plot by a categorical column
- Now try faceted histograms or bar charts
- Color points or bars by the hour of a date/time column
- Now try teaching this to someone who is just learning Data Science with Python
- Now try convincing an R user that they should use Python

Statistical Visualization

- The data to be visualized is a DataFrame
- The DataFrame is in a tidy format
  - Columns are observed variables of a given data type (quantitative, categorical, date/time)
  - Rows are samples
- The data is mapped to visual properties (x, y, color, shape) using group-by operations
- The groups correspond to conditional probability distributions

Visualization Grammar

Statistical visualizations can be expressed with a small number of abstractions that form a visualization grammar

Data	Input data to visualize
Transforms	Grouping, agg, stats, projection, layout
Scales	Map data values to visual properties
Guides	Axes and legends visualize scales
Marks	Visual elements to represent data

Declarative API

DECLARATIVE

- Specify *what* should be done
- *How* something is done becomes an internal detail
- Separates specification from execution

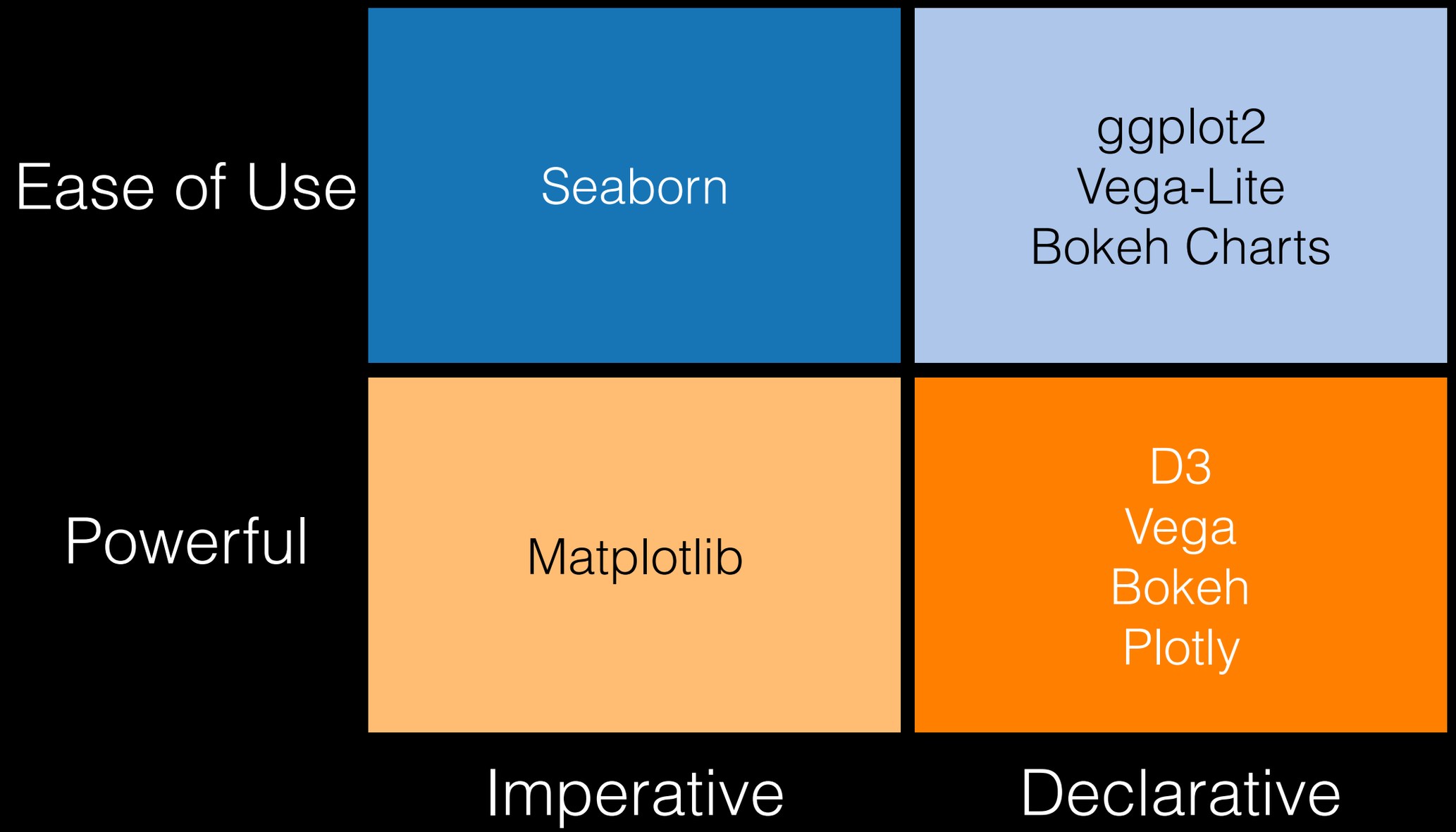
IMPERATIVE

- Specify *how* it should be done
- Must provide detailed logic and steps
- If the *what* changes even a small amount, the *how* can change significantly

Why a Declarative API?

- Usability
  - The *what* question is usually closer to the user's end goal than the *how* question
  - Most often users don't care *how* something gets done
    - Users have to write less code
    - Fewer abstractions, lower cognitive load
- Natural serialization format (JSON) for sharing, portability
- Possible to auto-generate APIs, code, examples, tests
- Much easier to support multiple languages (JS, Python, R, Scala, Julia, etc.)
- Performance: responsibility of implementation, not user

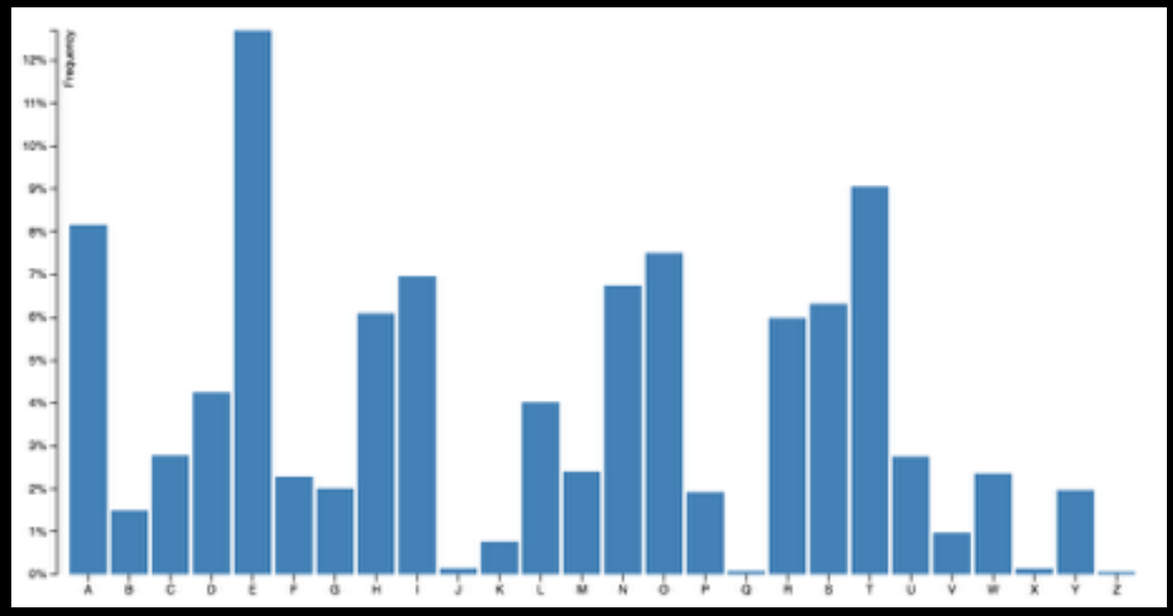
Mapping Visualization Libraries



```
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;
}
```

D3.js

- https://d3js.org/
- JavaScript library
- Declarative visualization grammar
- Powerful but verbose



Vega-Lite

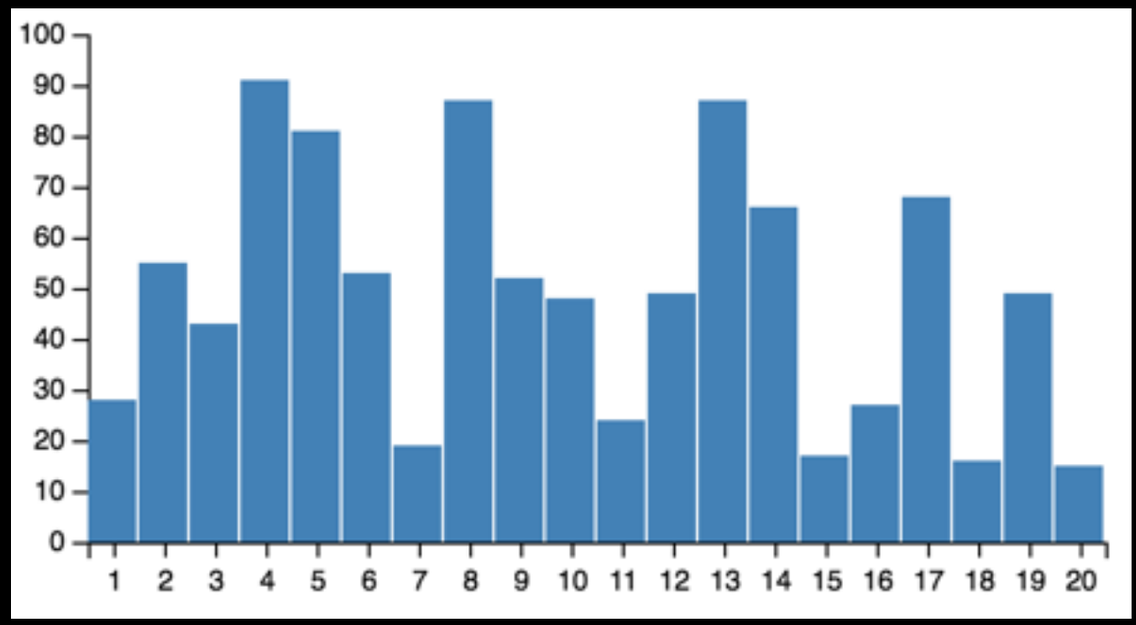
Vega

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"}
        },
```

Vega

- https://vega.github.io/vega/
- JSON specification for visualization grammar
- JS library compiles JSON down to D3
- Powerful but verbose
- Comparable to Bokeh and Plotly



Vega-Lite

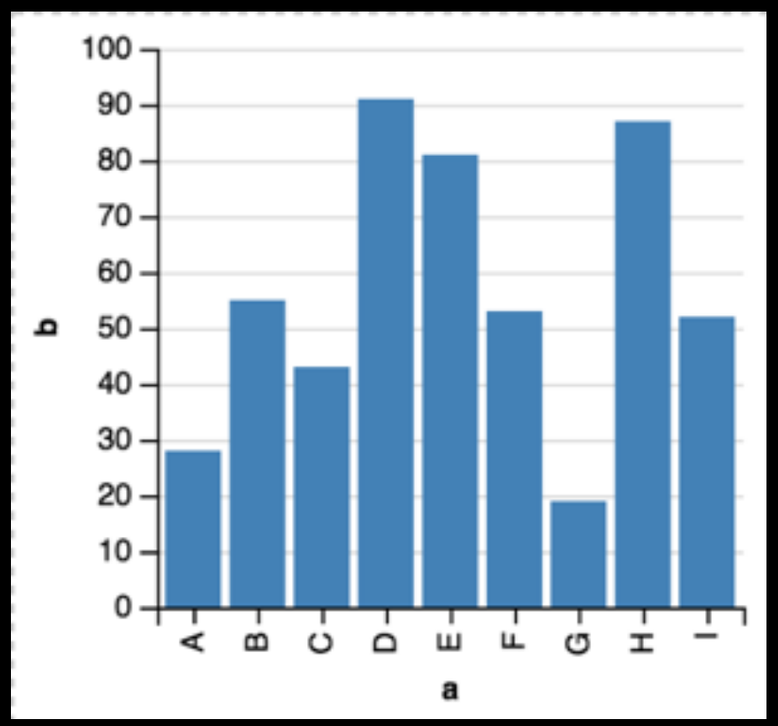
Vega

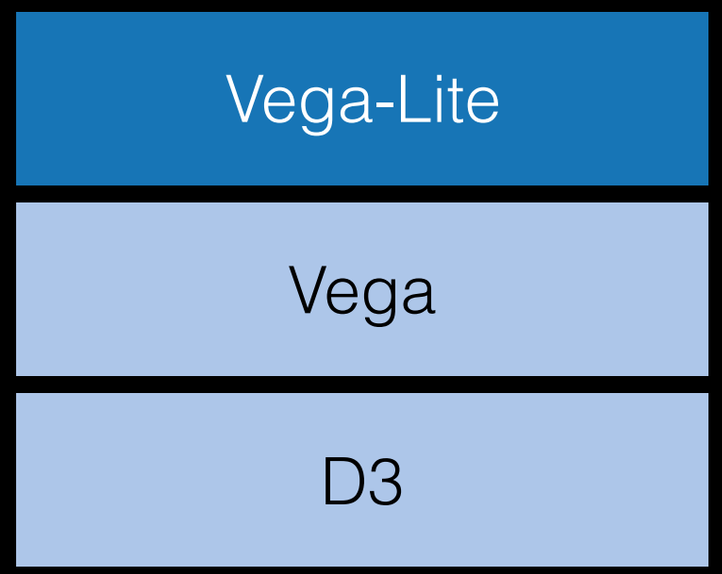
D3

Vega-Lite

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

- https://vega.github.io/vega-lite
- JSON specification for statistical visualization
- JavaScript library compiles JSON down to Vega/D3
- Constrained and concise
- Comparable to ggplot2
- Vega-Lite Online Editor
- Polestar UI on top of Vega-Lite





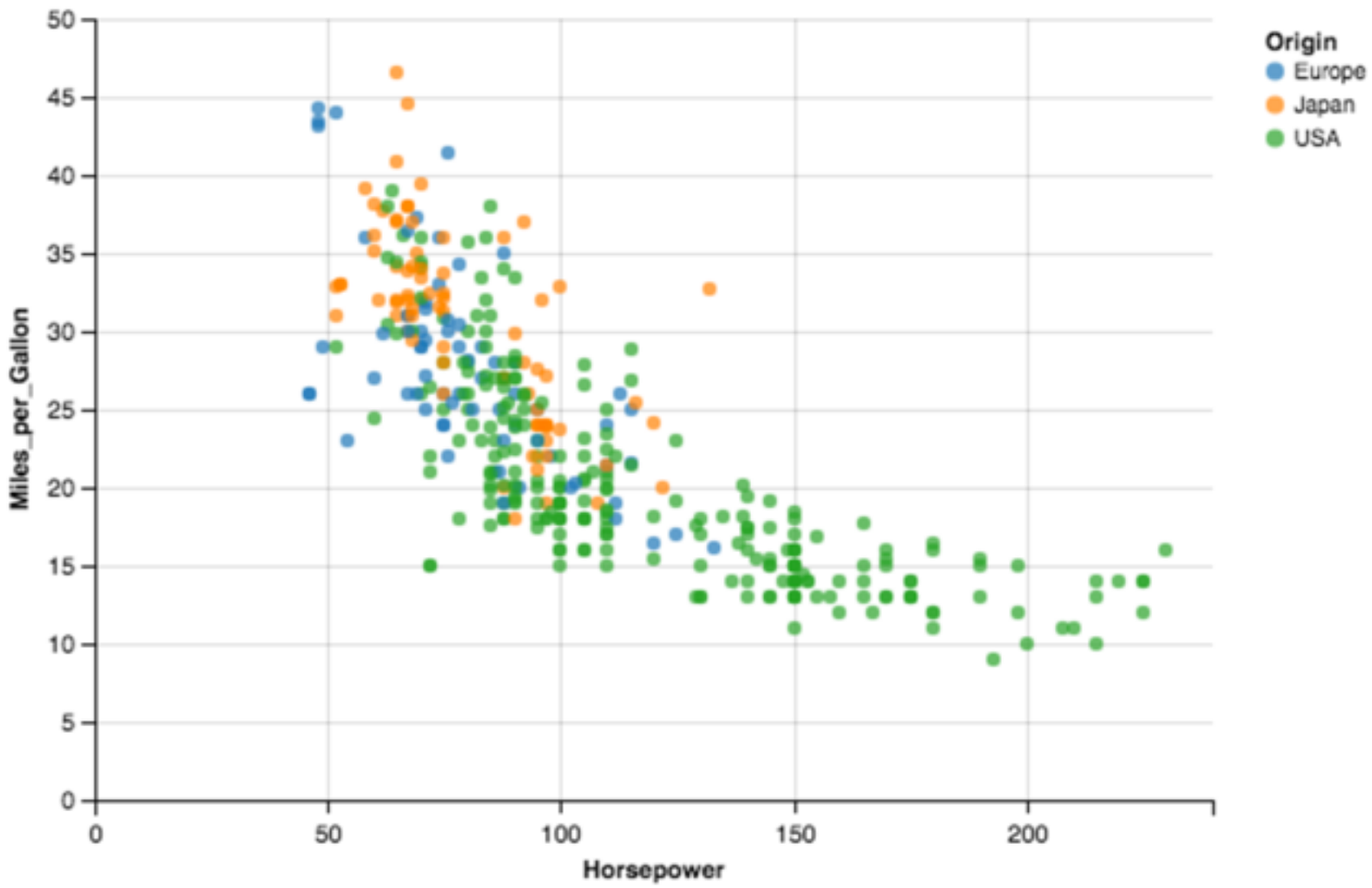
Introducing Altair

Altair

- Is a Python API for declarative statistical visualizations
- Performs no rendering of visualizations
- Emits type-checked and validated Vega-Lite JSON
- That is rendered by the Vega-Lite JS library in the Jupyter Notebook
- There is interest from Matplotlib, Bokeh and Plotly developers for adding their own Vega-Lite renderers
    - Users need the power of these libraries, but don't need multiple APIs for statistical visualization
- We are advocating that Vega/Vega-Lite become the lingua-franca of data visualization

Altair Example

In [1]:
```
from altair import datasets, Chart

data = datasets.load_dataset('cars')

Chart(data).mark_circle().encode(
    x='Horsepower',
    y='Miles_per_Gallon',
    color='Origin',
)
```

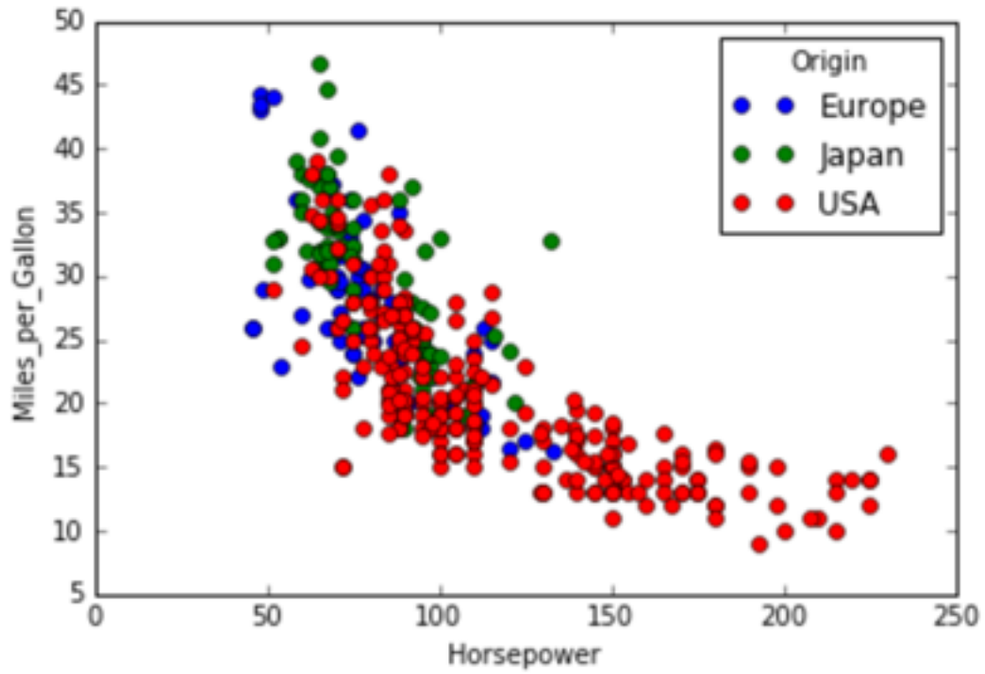


Comparison to Matplotlib

In [4]:
```
%matplotlib inline
import matplotlib.pyplot as plt
```

In [5]:
```
def scatter(group):
    plt.plot(group['Horsepower'], group['Miles_per_Gallon'],
             'o', label=group.name)

data.groupby('Origin').apply(scatter)
plt.legend(title='Origin')
plt.xlabel('Horsepower')
plt.ylabel('Miles_per_Gallon');
```

Explicit groupby and labeling required



Altair is Robust

- Most of the Altair code base is auto-generated from the Vega-Lite JSON specification
    - API, tests, example notebooks
- Type checking and validation performed by traitlets
- The Vega-Lite JSON produced by Altair is essentially guaranteed to be valid
    - At worst, it may not produce the visualization you expect

Altair is Usable

- Carefully designed API focused on usability
  - Type-checked attributes
  - Thin layer of convenience methods
  - Method chaining and operators (+, +=)
- Intelligent heuristics, such as type inference, to reduce the amount of code needed
- Shorthand notation for specifying columns, data types and aggregation
- Flat is better than nested: subtle API changes to flatten the highly nested Vega-Lite API
- Extensive documentation and over 70 examples

Altair is Powerful

- Canvas/SVG/PNG in the Jupyter Notebook
- All visualizations display on GitHub and nbviewer
- Export to PNG, Online Vega-Lite Editor
- Serialize visualizations as JSON files
- Auto-generate readable Altair Python code from Vega-Lite JSON

Explore Altair!

Install with conda:

```
conda install altair --channel conda-forge
```

Or pip:

```
pip install altair
jupyter nbextension install --sys-prefix --py vega
```

Explore Altair with live Jupyter Notebooks:

```
from altair import *
tutorial()
```

Read the documentation on GitHub:

https://github.com/ellisonbg/altair

Thanks!

https://github.com/ellisonbg/altair

Special thanks to Jeff Heer and members of his Interactive Data Lab at UW: Dominik Moritz, Arvind Satyanarayan

If you are an R/Julia/Scala/etc. developer interested in creating an Altair implementation for your language, please get in touch. Most of the hard work is done!