Bespoke Visualizations with a Declarative Twist

Jake VanderPlas @jakevdp
Convoy Tech
Feb 15, 2018
Python Viz is a bit Painful...

"I have been using Matplotlib for a decade now, and I still have to look most things up"

“I love Python but I switch to R for making plots”

“I do viz in Python, but switch from matplotlib to seaborn to bokeh depending on what I need to do”
Python's Visualization Landscape

- matplotlib
- seaborn
- pandas
- ggpy
- scikit-plot
- Yellowbrick
- networkx
- basemap/cartopy
- ipyleaflet
- bqplot
- bokeh
- cufflinks
- plotly
- toyplot
- pythreejs
- Vaex
- matplotlib
- holoviews
- d3js
- mpld3
- Altair
- Vega
- Vega-Lite
- d3po
- Vincent
- GlueViz
- Lightning
- PyQTgraph
- MayaVi
- Chaco
- pygal
- GR framework
- YT

Frameworks:
- ipyvolume
- ipyleaflet
- Vispy
- Glumpy
- OpenGL

JavaScript Libraries:
- plotly
- bqplot
- bokeh
- cufflinks
- toyplot
- pythreejs
- d3js
- mpld3
- Vega
- Vega-Lite

WebGL Libraries:
- PyQTgraph
- MayaVi
- GlueViz
- Lightning

Visualization Tools:
- ipyleaflet
- Vispy
- Lightning
- GlueViz
- PyQTgraph
- MayaVi
- Lightning

Graph Visualization Tools:
- networkx
- basemap/cartopy
- Yellowbrick
- graph-tool
- graphviz
Problem: where would you tell beginners to start?

- Matplotlib
- Bokeh
- Plotly
- Seaborn
- Holoviews
- VisPy
- ggplot
- pandas plot
- Lightning

Each library has strengths, but arguably none is yet the “killer viz app” for Data Science.
Some examples . . .
http://matplotlib.org/
Plotting with Matplotlib

import matplotlib.pyplot as plt
from numpy.random import import rand

for color in ['red', 'green', 'blue']:
    x, y = rand(2, 100)
    size = 200.0 * rand(100)
    plt.scatter(x, y, c=color, s=size, label=color, alpha=0.3, edgecolor='none')

plt.legend(frameon=True)
plt.show()
Plotting with Matplotlib

Strengths:
- Designed like MatLab: switching was easy
Plotting with Matplotlib

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

```python
In [26]: from matplotlib import rcsetup
crcsetup.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 over a decade

May 11, 2003 – Feb 15, 2018

Contributions to master, excluding merge commits
Matplotlib Gallery
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 over a decade

**Weaknesses:**
- API is imperative & often overly verbose
- Sometimes poor stylistic defaults
- Poor support for web/interactive graphs
- Often slow for large & complicated data
http://bokeh.pydata.org/
from bokeh.plotting import figure, show
from bokeh.models import LinearAxis, Range1d

p = figure()

for color in ['red', 'green', 'blue']:
    x, y = rand(2, 100)
    size = 0.03 * rand(100)
    p.circle(x, y, fill_color=color, radius=size,
             legend=color, fill_alpha=0.3,
             line_color=None)

show(p)
Plotting with Bokeh

In [10]:
   p = figure()
   p.circle(iris.petalLength, iris.sepalLength)
   show(p)
Plotting with Bokeh

Advantages:
- Web view/interactivity
- Imperative and Declarative layer
- Handles large and/or streaming datasets
- Geographical visualization
- Fully open source

Disadvantages:
- No vector output (need PDF/EPS? Sorry)
- Newer tool with a smaller user-base than matplotlib
Basic Plotting with Plotly

```python
from plotly.graph_objs import Scatter
from plotly.offline import iplot

p = Scatter(x=iris.petalLength,
            y=iris.sepalLength,
            mode='markers')

iplot([p])
```
Plotly Gallery
Plotting with Plotly

Advantages:
- Web view/interactivity
- Multi-language support
- 3D plotting capability
- Animation capability
- Geographical visualization

Disadvantages:
- Some features require a paid plan
Moving to *Statistical Visualization*
from altair import load_dataset
iris = load_dataset('iris')
iris.head()

<table>
<thead>
<tr>
<th>petalLength</th>
<th>petalWidth</th>
<th>sepalLength</th>
<th>sepalWidth</th>
<th>species</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1.4</td>
<td>0.2</td>
<td>5.1</td>
<td>3.5</td>
<td>setosa</td>
</tr>
<tr>
<td>1 1.4</td>
<td>0.2</td>
<td>4.9</td>
<td>3.0</td>
<td>setosa</td>
</tr>
<tr>
<td>2 1.3</td>
<td>0.2</td>
<td>4.7</td>
<td>3.2</td>
<td>setosa</td>
</tr>
<tr>
<td>3 1.5</td>
<td>0.2</td>
<td>4.6</td>
<td>3.1</td>
<td>setosa</td>
</tr>
<tr>
<td>4 1.4</td>
<td>0.2</td>
<td>5.0</td>
<td>3.6</td>
<td>setosa</td>
</tr>
</tbody>
</table>

Data in **Tidy 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: 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
Most Useful for Data Science is **Declarative Visualization**

**Imperative**
- Specify *How* something should be done.
- Must manually specify plotting steps
- Specification & Execution intertwined.

**Declarative**
- Specify *What* should be done
- Details determined automatically
- Separates Specification from Execution

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

g = sns.FacetGrid(iris, col="species", hue="species")
g.map(plt.scatter, "petalLength", "sepalWidth", alpha=0.3)
g.add_legend();
http://altair-viz.github.io/
Altair for Declarative Visualization

```python
from altair import Chart
from vega_datasets import data

iris = data.iris()

Chart(iris).mark_circle().encode(
    x='petalLength',
    y='sepalWidth',
    color='species'
)
```
Altair for Declarative Visualization

```python
from altair import Chart
from vega_datasets import data

iris = data.iris()

Chart(iris).mark_circle().encode(
    x='petalLength',
    y='sepalWidth',
    color='species'
).interactive()
```
Encodings are Flexible:

```python
from altair import Chart
from vega_datasets import data

iris = data.iris()

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

Declarative statistical visualization library for Python, driven by Vega-Lite

http://github.com/altair-viz/altair

Collaboration with Brian Granger (Jupyter team), myself, and UW’s Interactive Data Lab
So What Is Altair?
Drought and Deluge in the Lower 48

Last summer’s drought, one of the worst in a century, has continued through the winter. This chart shows the proportion of what is now the contiguous U.S. in various stages of drought over 118 years of record-keeping. Roll mouse over individual months to see what percentage of the lower 48 was in drought.

During January, 56 percent of the contiguous U.S. was in moderate to extreme drought, the highest January level since 1955.

2010  2011  2012  2013

Dryness based on the Palmer Drought Index

Wetter  Avg.  Drier

Roll mouse over to isolate categories

2000
Large, intense regional droughts that migrated across the country. The West saw persistent drought in the first half of the decade.


The drought of 1988 destroyed at least half the crops on the Great Plains and was one of the costliest U.S. natural disasters.


(live version at NYT)
But working in D3 can be challenging . . .
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.
Bar Chart: Vega-Lite

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

```
{
  "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: Altair

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

```python
import pandas as pd
from altair import Chart

                     'b': [28, 55, 43, 91, 81, 53, 19, 87, 52]})

Chart(data).mark_bar().encode(
    x='a',
    y='b',
)
```
url = load_dataset('iris', url_only=True)

chart = Chart(url).mark_circle(
    opacity=0.3
).encode(
    x='petalLength:Q',
    y='sepalWidth:Q',
    color='species:N',
)
chart.display()
From Declarative API to declarative Grammar

```python
>>> chart.to_dict()

{'config': {'mark': {'opacity': 0.3}},
 'data':
    {'url': 'https://vega.github.io/vega-datasets/data/iris.json'},
    'encoding': {'color': {'field': 'species', 'type': 'nominal'},
                   'x': {'field': 'petalLength', 'type': 'quantitative'},
                   'y': {'field': 'sepalWidth', 'type': 'quantitative'}},
    'mark': 'circle'}
```
Key Features of Altair:

- Designed with Statistical Visualizations in mind
- Data specified in Tidy Format & linked to a declared type: Quantitative, Nominal, Ordinal, Temporal
- Well-defined set of marks to represent data
- Encoding Channels map data features (i.e. columns) to visual encodings (e.g. x, y, color, size, etc.)
- Simple data transformations supported natively
But why another plotting library?

**Teaching:** students can learn visualization concepts with minimal syntactic distraction.

**Publishing:** Instead of publishing pixels, can publish data + plot specification for greater flexibility & reproducibility.

**Cross-Pollination:** Vega-Lite has the potential to provide a cross-platform lingua franca of statistical visualization.

- Matplotlib
- Bokeh
- Plotly
- Seaborn
- Holoviews
- VisPy
- ggplot
- pandas plot
- Lightning
Altair/Vega-Lite supports many plot types:

In [4]: Chart(cars).mark_point().encode(
    x='Weight_in_lbs',
    y='Acceleration',
    color='Cylinders:N'
)
Altair/Vega-Lite supports many plot types:

```python
In [6]: Chart(cars).mark_point().encode(
    x='Displacement',
    y='Weight_in_lbs',
    color='Horsepower',
    row='Origin'
).configure_cell(width=300, height=150)
```
Altair/Vega-Lite supports many plot types:

```
In [7]: Chart(cars).mark_bar(stacked='normalize').encode(
    Y('Origin'),
    X('*:Q', aggregate='count', sort='descending'),
    Color('Cylinders:N')
)
```
Altair/Vega-Lite supports many plot types:

```python
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)
```
Altair/Vega-Lite supports many plot types:
Altair/Vega-Lite supports many plot types:

```python
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)
```
Altair 2.0: a Grammar of Interaction
Some Live Examples . . .

See the notebook at
Try Altair:

$ conda install altair --channel conda-forge

or

$ pip install altair
$ jupyter nbextention install --sys-prefix --py vega

For a Jupyter notebook tutorial, type

import altair
altair.tutorial()

http://github.com/ellisonbg/altair/
Altair’s Development is Active!

- More plot types
- Higher-level Statistical routines
- Improve layering API
- Vega-Tooltip interaction
- Vega-Lite's Grammar of Interaction
Thank You!

Email:   jakevdp@uw.edu
Twitter: @jakevdp
Github:  jakevdp
Web:     http://vanderplas.com
Blog:    http://jakevdp.github.io