The Yhat Blog

machine learning, data science, engineering

ggplot for python

by Yhat

October 13, 2013

Analytical projects often begin w/ exploration--namely, plotting distributions to find patterns of interest and importance. And while there are dozens of reasons to add R and Python to your toolbox, it was the superior visualization faculties to add R and Python to your toolbox, it was the superior visualization facilities that spurred my own investment in these tools.

Excel makes some great looking plots, but I wouldn't be the first to say that creating charts in Excel involves a lot of manual work. Data is messy, and exploring it requires considerable effort to clean it up, transform it, and rearrange it from one format to another. R and Python make these tasks easier, allowing you to visually inspect data in several ways quickly and without tons of effort.

The preeminent graphics packages for R and Python are `ggplot2` and `matplotlib` respectively. Both are feature-rich, well maintained, and highly capable. Now, I've always been a `ggplot2` guy for graphics, but I'm a Python guy for everything else. As a result, I'm constantly toggling between the two languages which can become rather tedious.

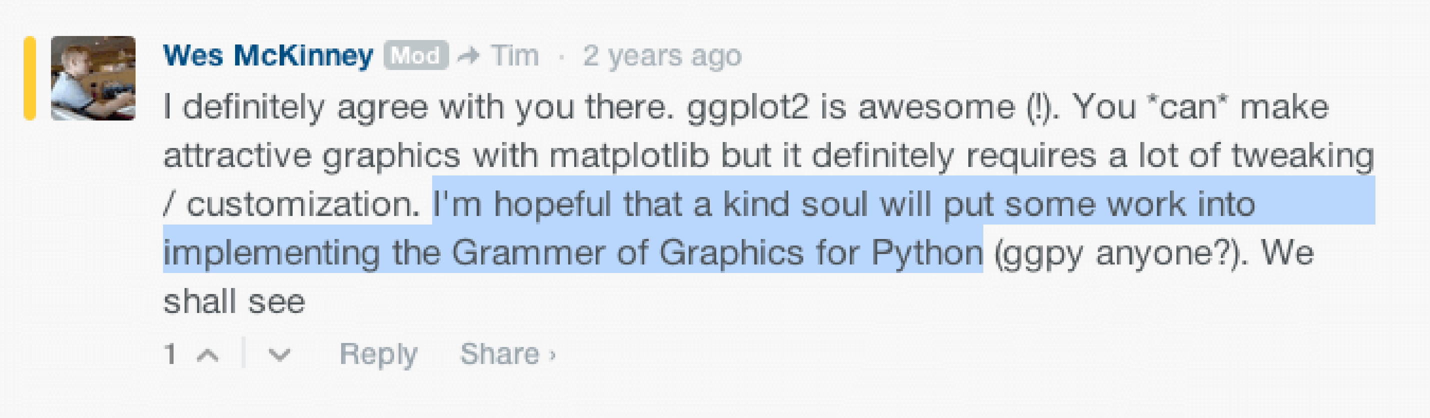
This is a post about `ggplot2` and an attempt to bring it to Python.

Give me some `ggplot`

There's no shortage of talk around improving the plotting capabilities of Python. Libraries like `Bokeh`, `d3py` are exciting or at least intruiging, but they aren't as accessible as either `ggplot2` or `matplotlib` (yet, at least).

I don't know all that much about these two projects, but are they solving for interactivity and presentation or for every day data exploration? Most of the time, I just want to `plot(X,y)`, see the results, and move on. `matplotlib` works, but it's not exactly the belle of the ball among contemporary graphics libraries. And let's get real for a second, `matplotlib` just stinks the big one from a usability perspective.

We've been hearing whispers of a Grammer of Graphics Python implementation for a while...



I definitely agree with you there. ggplot2 is awesome (!). You *can* make attractive graphics with matplotlib but it definitely requires a lot of tweaking / customization. I'm hopeful that a kind soul will put some work into implementing the Grammer of Graphics for Python (ggpy anyone?). We shall see

> matplotlib is powerful...but its plotting commands remain rather verbose, and its no-frills, default output looks much more like Excel circa 1993 than ggplot circa 2013. ~ Jake Vanderplas, Matplotlib & the Future of Visualization in Python (@jakevdp)

But you either get busy livin', or you get busy dyin', so we thought we'd give it a shot.

`from ggplot import *`

`ggplot` is a graphics package for Python that aims to approximate R's `ggplot2` package in both usage and aesthetics. What we're trying to do w/ this library is keep the API as close to the `R` version as possible and make the plots look as great as the Big Guy's.

Now, there are some things in here that'll make some of you Pythonistas just cringe. But it's a fun little library, so hold on to your hats!

Usage and Examples

Install with `pip`:

```
  pip install -U ggplot
```

Let's look at an example to compare usage in R vs. Python:

Here's R:

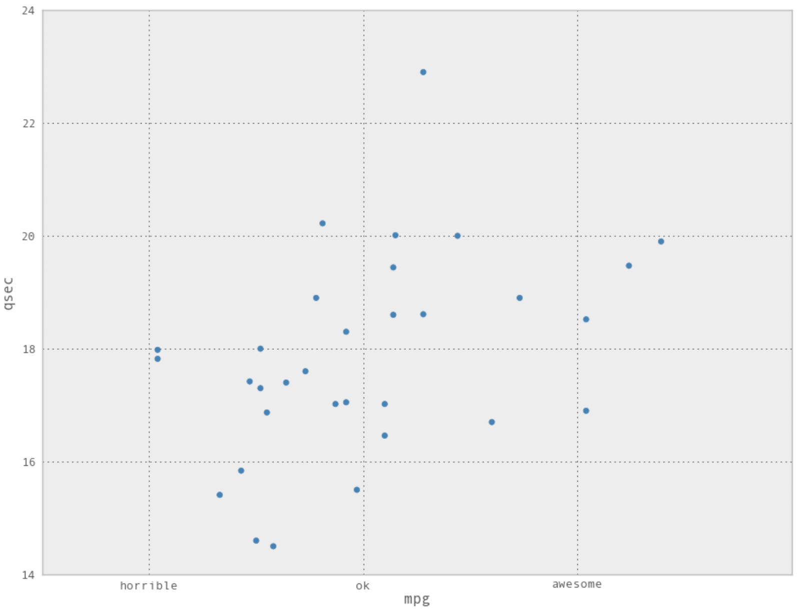
```
library(ggplot2)


ggplot(mtcars, aes(mpg, qsec)) +
  geom_point(colour='steelblue') +
  scale_x_continuous(breaks=c(10,20,30),
                     labels=c("horrible", "ok", "awesome"))
```

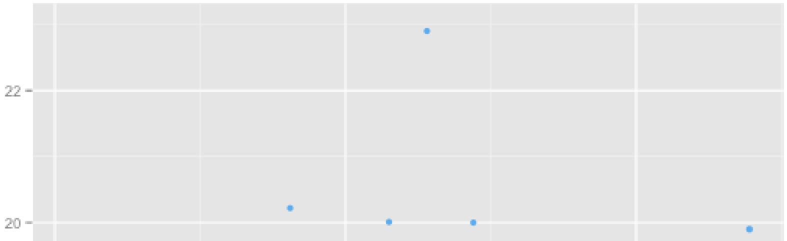
And here's Python:

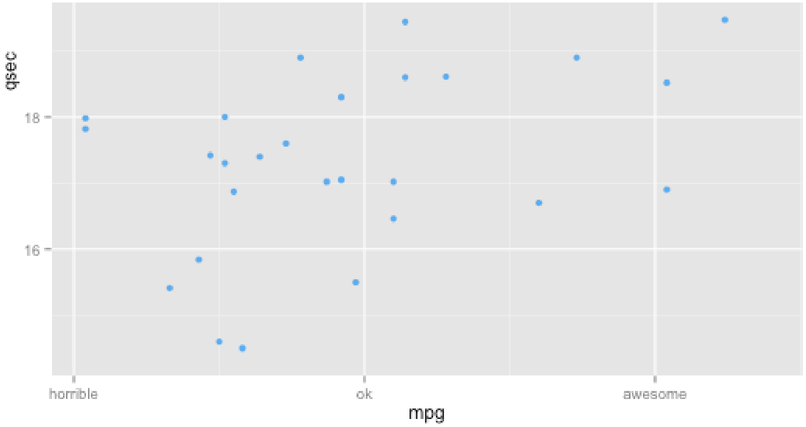
```
from ggplot import *
```

```
print ggplot(mtcars, aes('mpg', 'qsec')) + \
  geom_point(colour='steelblue') + \
  scale_x_continuous(breaks=[10,20,30],  \
                     labels=["horrible", "ok", "awesome"])
```

Plots below. Python output is on the left. R's output is on the right.



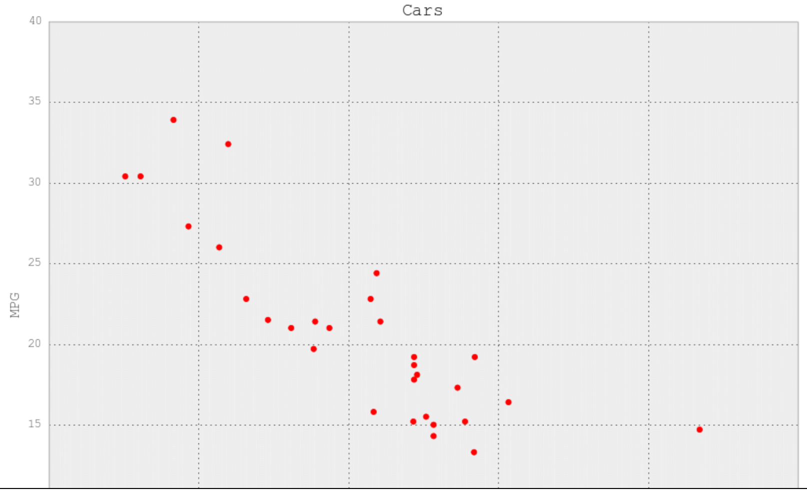


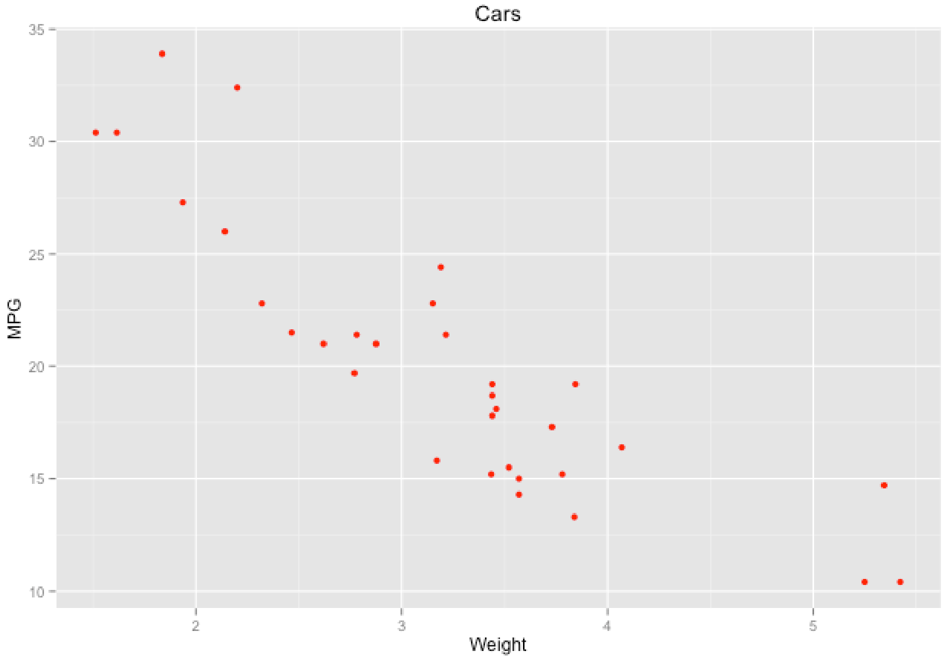


Pretty similar, right?

Let's do one that's a bit more complicated. I only have one code snippet because it's exactly the same!

```
p + geom_point(color = "red") + ggtitle("Cars") + xlab("Weight") + ylab
("MPG")
```





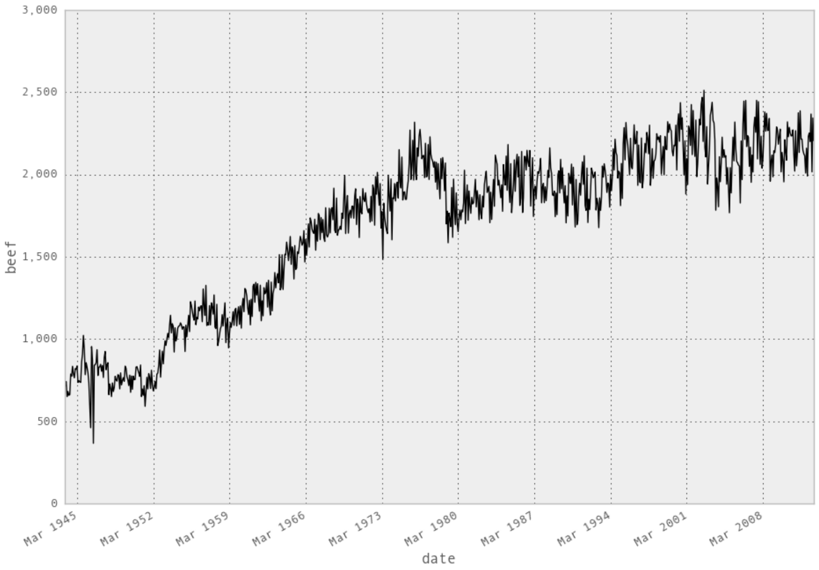
How about one with dates?

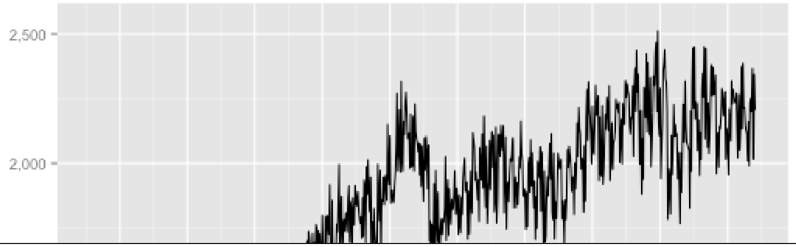
R

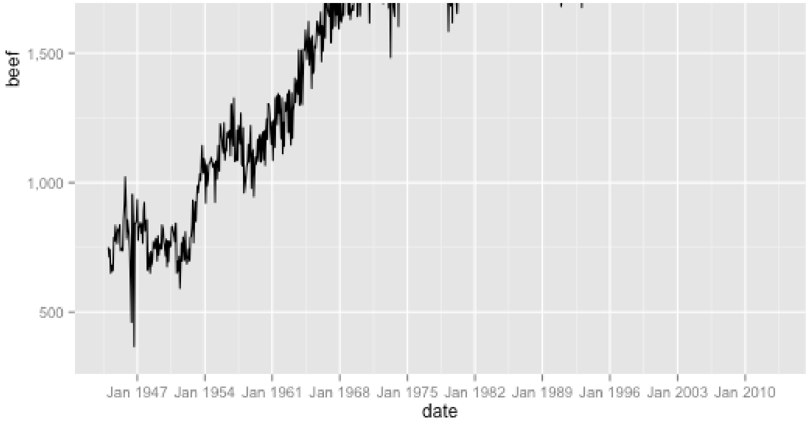
```
ggplot(meat, aes(date,beef)) +
  geom_line(colour='black') +
  scale_x_date(breaks=date_breaks('7 years'),labels = date_format("%b %Y")) +
  scale_y_continuous(labels=comma)
```

Python

```
print ggplot(meat, aes('date','beef')) + \
    geom_line(color='black') + \
    scale_x_date(breaks=date_breaks('7 years'), labels='%b %Y') + \
    scale_y_continuous(labels='comma')
```







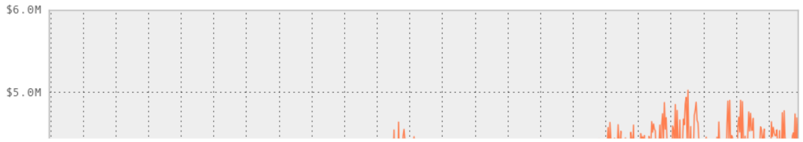
Reminder of how this would be done in pure `matplotlib`:

```
import matplotlib.pyplot as plt
from matplotlib.dates import YearLocator, DateFormatter
from ggplot import meat
tick_every_n = YearLocator(7)
date_formatter = DateFormatter('%b %Y')
x = meat.date
y = meat.beef
fig, ax = plt.subplots()
ax.plot(x, y, 'black')
ax.xaxis.set_major_locator(tick_every_n)
ax.xaxis.set_major_formatter(date_formatter)
fig.autofmt_xdate()
plt.show()
```

Not terrible, but if you wanted to change the x-ticks from a yearly interval to a different unit of time, say months? Or if you wanted your yaxis format to be a currency or in millions instead of comma format?

You'd need to modify the code more than you'd prefer. `ggplot` style makes that super easy to do.

```
from ggplot import *

print ggplot(meat, aes('date','beef * 2000')) + \
    geom_line(color='coral') + \
    scale_x_date(breaks=date_breaks('36 months'), labels='%Y') + \
    scale_y_continuous(labels='millions')
```



Faceting

No grammar of graphics would be complete with out faceting/trellis plots. Faceting is still somewhat of a work in progress but the core functionality is there. The `facet_wrap` and `facet_grid` functions take x and y parameters as strings (compared to x ~ y in R). There are still a few bugs, but for the most part stuff is working as expected.

About a year ago Carl from Slender Means put together a Python port of Drew Conway and John Myles White's *Machine Learning for Hackers*. One of the biggest pains he experienced was doing trellis plots in `matplotlib`. Take a look at his code.

```
nrow = 13; ncol = 4; hangover = len(us_states) % ncol
fig, axes = plt.subplots(nrow, ncol, sharey = True,
figsize = (9, 11))

fig.suptitle('Monthly UFO Sightings by U.S. State\nJanuary 1990 through August 2010',
             size = 12)
plt.subplots_adjust(wspace = .05, hspace = .05)
num_state = 0
for i in range(nrow):
    for j in range(ncol):
        xs = axes[i, j]
        xs.grid(linestyle = '-', linewidth = .25, color = 'gray')

        if num_state < 51:
            st = us_states[num_state]
            sightings_counts.ix[st, ].plot(ax = xs, linewidth = .75)
            xs.text(0.05, .95, st.upper(), transform = axes[i, j].transAx
es,
                    verticalalignment = 'top')
            num_state += 1
        else:
            # Make extra subplots invisible
            plt.setp(xs, visible = False)

        xtl = xs.get_xticklabels()
        ytl = xs.get_yticklabels()

        # X-axis tick labels:
        # Turn off tick labels for all the the bottom-most
        # subplots. This includes the plots on the last row, and
        # if the last row doesn't have a subplot in every column
        # put tick labels on the next row up for those last
        # columns.
        #
        # Y-axis tick labels:
```

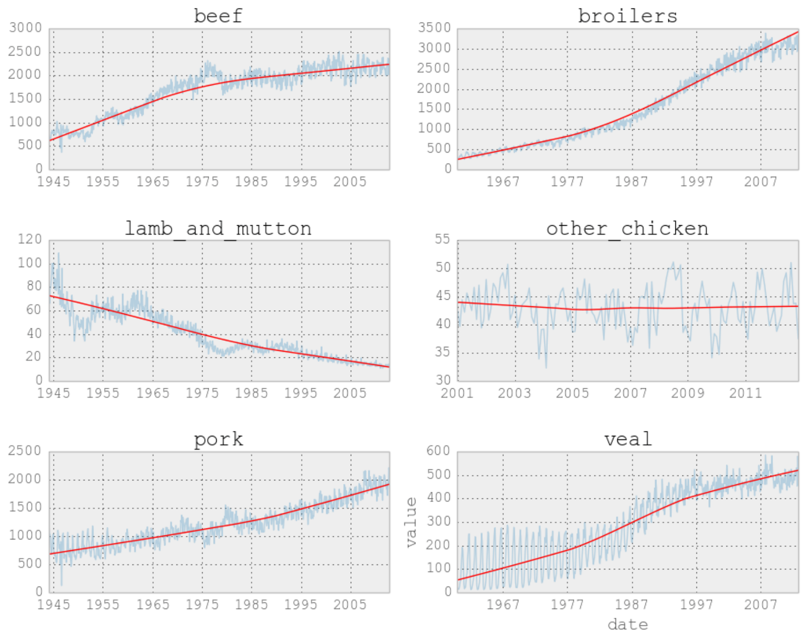
```
        # Put left-axis labels on the first column of subplots,
        # odd rows. Put right-axis labels on the last column
        # of subplots, even rows.
        if i < nrow - 2 or (i < nrow - 1 and (hangover == 0 or
                            j <= hangover - 1)):
            plt.setp(xtl, visible = False)
        if j > 0 or i % 2 == 1:
            plt.setp(ytl, visible = False)
        if j == ncol - 1 and i % 2 == 1:
            xs.yaxis.tick_right()

        plt.setp(xtl, rotation=90.)
```

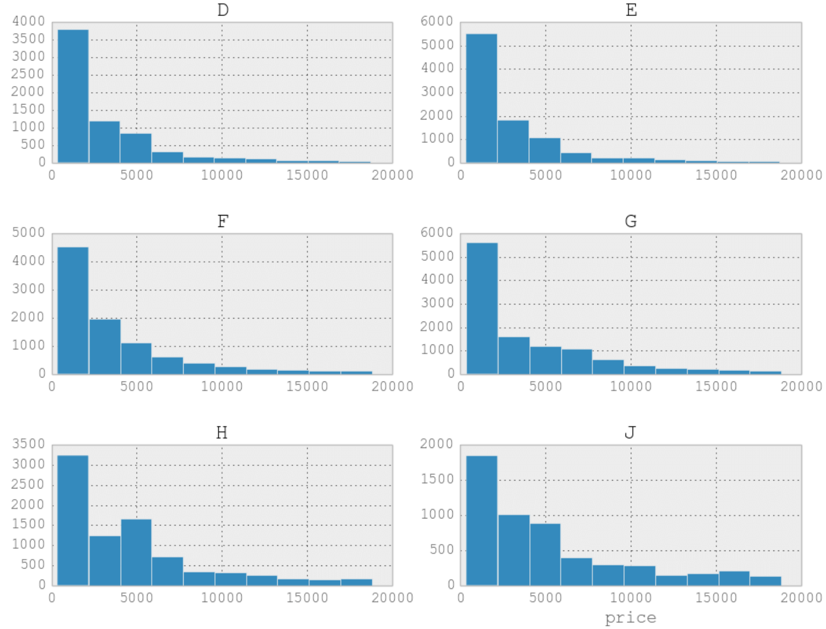
No chance I'd have the patience to go through this every time I wanted to facet something.

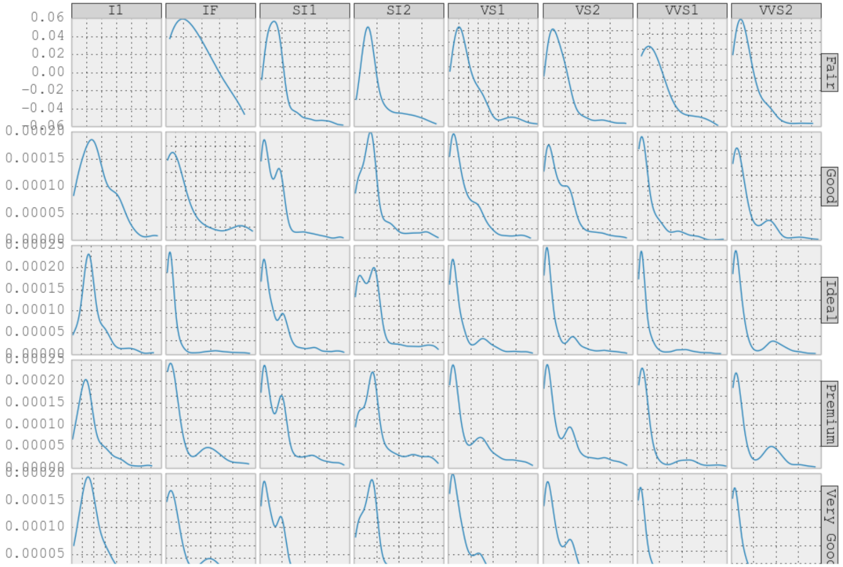
Take a look at how easy it is to do in `ggplot`.

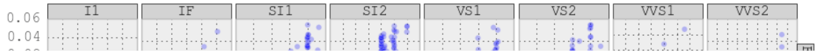
```
import pandas as pd

meat_lng = pd.melt(meat, id_vars=['date'])

p = ggplot(aes(x='date', y='value'), data=meat_lng)
p + geom_point() + \
    stat_smooth(colour="red") + \
    facet_wrap("variable")
    facet_wrap("variable")

p + geom_hist() + facet_wrap("color")

p = ggplot(diamonds, aes(x='price'))
p + geom_density() + \
    facet_grid("cut", "clarity")

p = ggplot(diamonds, aes(x='carat', y='price'))
p + geom_point(alpha=0.25) + \
    facet_grid("cut", "clarity")
```

***Note**: I'm running in `IPython` so I don't need to use print.*




D
E
F
G
H
J
0
5000
10000
15000
20000
0
500
1000
1500
2000
2500
3000
3500
4000
5000
6000
price


I1
IF
SI1
SI2
VS1
VS2
VVS1
VVS2
Fair
Good
Ideal
Premium
Very Good
0.06
0.04
0.02
0.00
−0.02
−0.04
0.00020
0.00015
0.00010
0.00005
0.00025



Way less code. Way more plots.

Final Thoughts

There's really too much to cover in just one post, but we're pretty excited about this project. We'll do a follow-up post soon to show more features!

Other resources

- Matplotlib and the Future of Visualization in Python by Jake Vanderplas (@jakevdp)

(@jakevdp)

- A Superficial Comparison of matplotlib vs ggplot2
- making matplotlib graphs look like R by default? - popular stackoverflow question
- rplot.py - a fork of pandas on github which appears to have some ggplot2 type features.
- Plotting for Pandas GSoC2012 (see rploy.py link above)
- Trellis graphs (Chapter 1, Part 5 from Carl Vogel's "Will it Python?" series)
- why there is no 'ggplot2' like graphics lib for python? (thread on reddit)
- Ggplot2 graph style with matplotlib
- Making matplotlib look like ggplot
- When to use Excel? And when to use R? by Michael Milton
- Consultants? Chart in ggplot2 via Learn R Blog

Our Products

A Python IDE built for doing data science directly on your desktop.

DOWNLOAD IT NOW!

A platform for productionizing, scaling, and monitoring predictive models in production applications.

LEARN MORE

Yhat (pronounced Y-hat) provides data science and decision management solutions that let data scientists create, deploy and integrate insights into any business application without IT or custom coding.

With Yhat, data scientists can use their preferred scientific tools (e.g. R and Python) to develop analytical projects in the cloud collaboratively and then deploy them as highly scalable real-time decision making APIs for use in customer- or employee-facing apps.