Python Plotting for Exploratory Data Analysis

pythonplot.com

> The simple graph has brought more information to the data analyst's mind than any other device.

Contents

Introduction

Plotting is an essential component of data analysis. As a data scientist, I spend a significant amount of my time making simple plots to understand complex data sets (exploratory data analysis) and help others understand them (presentations).

In particular, I make a lot of bar charts (including histograms), line plots (including time series), scatter plots, and density plots from data in Pandas data frames. I often want to facet these on various categorical variables and layer them on a common grid.

Python Plotting Options

Python plotting libraries are manifold. Most well known is Matplotlib.

"Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms." Native Matplotlib is the cause of frustration to many data analysts due to the complex syntax. Much of that frustration would be alleviated if it were recognized as a library of lower level plotting primitives on which other tools can be built. (If you are frustrated by Matplotlib and haven't read Effectively Using Matplotlib by Chris Moffitt, go read it.)

Matplotlib-Based Libraries

Many excellent plotting tools are built on top of Matplotlib.

Pandas plots provides the "basics to easily create decent looking plots" from data frames. It provides about 70% of what I want to do day-to-day. Importantly, it lacks robust faceting capabilities.

"plotnine is an implementation of a grammar of graphics in Python, it is based on ggplot2." plotnine is a attempt to directly translate ggplot2 to Python; despite some quirks and bugs, it works very well for a young product.

"Seaborn is a Python visualization library based on matplotlib. It provides a high-level interface for drawing attractive statistical graphics." Seaborn makes beautiful plots but is geared toward specific statistical plots, not general purpose plotting. It does have a powerful faceting utility function that I use regularly.

Interactive Plotting Libraries

There are several tools that can make the kinds of plots described here. At present, I have little experience with them. If anyone would like to help add examples, please get in touch.

"Altair is a declarative statistical visualization library for Python, based on Vega-Lite." According to Jake Vanderplas, "Declarative visualization lets you think about data and relationships, rather than incidental details." I provide Altair examples rendered as static images.

"plotly's Python graphing library makes interactive, publication-quality graphs online. Examples of how to make line plots, scatter plots, area charts, bar charts, error bars, box plots, histograms, heatmaps, subplots, multiple-axes, polar charts, and bubble charts." I provide plotly examples rendered as static images.

"Bokeh is a Python interactive visualization library that targets modern web browsers for presentation."

"bqplot is a Grammar of Graphics-based interactive plotting framework for the Jupyter notebook."

The Python Plotting Landscape

If you're interested in the breadth of plotting tools available for Python, I commend Jake Vanderplas's Pycon 2017 talk called the The Python Visualization Landscape. Similarly, the blogpost A Dramatic Tour through Python's Data Visualization Landscape (including ggplot and Altair) by Dan Saber is worth your time.

Hearty Thank You

Much Python plotting development is done by open source developers who have an (almost) thankless task. I am extremely grateful for the countless hours of many who have helped me do my job. Please keep it up!

Why all the talk about ggplot?

The word "ggplot" comes up a lot in discussions of plotting. Before I started using Python, I did most of my data analysis work in R. I, with many Pythonistas, remain a big fan of Hadley Wickham's ggplot2, a "grammar of graphics" implementation in R, for exploratory data analysis.

Like scikit-learn for machine learning in Python, ggplot2 provides a consistent API with sane defaults. The consistent interface makes it easier to iterate rapidly with low cognitive overhead. The sane defaults makes it easy to drop plots right into an email or presentation.

Particularly, ggplot2 allows the user to make basic plots (bar, histogram, line, scatter, density, violin) from data frames *with* faceting and layering by discrete values.

An excellent introduction to the power of ggplot2 is in Hadley Wickham and Garrett Grolemund's book R for Data Science.

Humble Rosetta Stone for Visualization in Exploratory Data Analysis

Below I have begun compiling a list of basic plots for exploratory data analysis. I have generated the plots with as many different libraries as time (and library) permits.

My hope is that this will (1) help you in your daily practice to work with what is available and (2) help inspire future development of Python plotting libraries.

Some rudimentary instructions on how you can contribute plots are here. General feedback or other plot suggestions are welcome.

Data

The datasets used below are included with ggplot2. One is the Prices of 50,000 round cut diamonds and the other is Fuel economy data from 1999 and 2008 for 38 popular models of car.

The time series example is a random walk I generate with a quick Python script.

Here's what a few rows of the datasets looks like:

ts

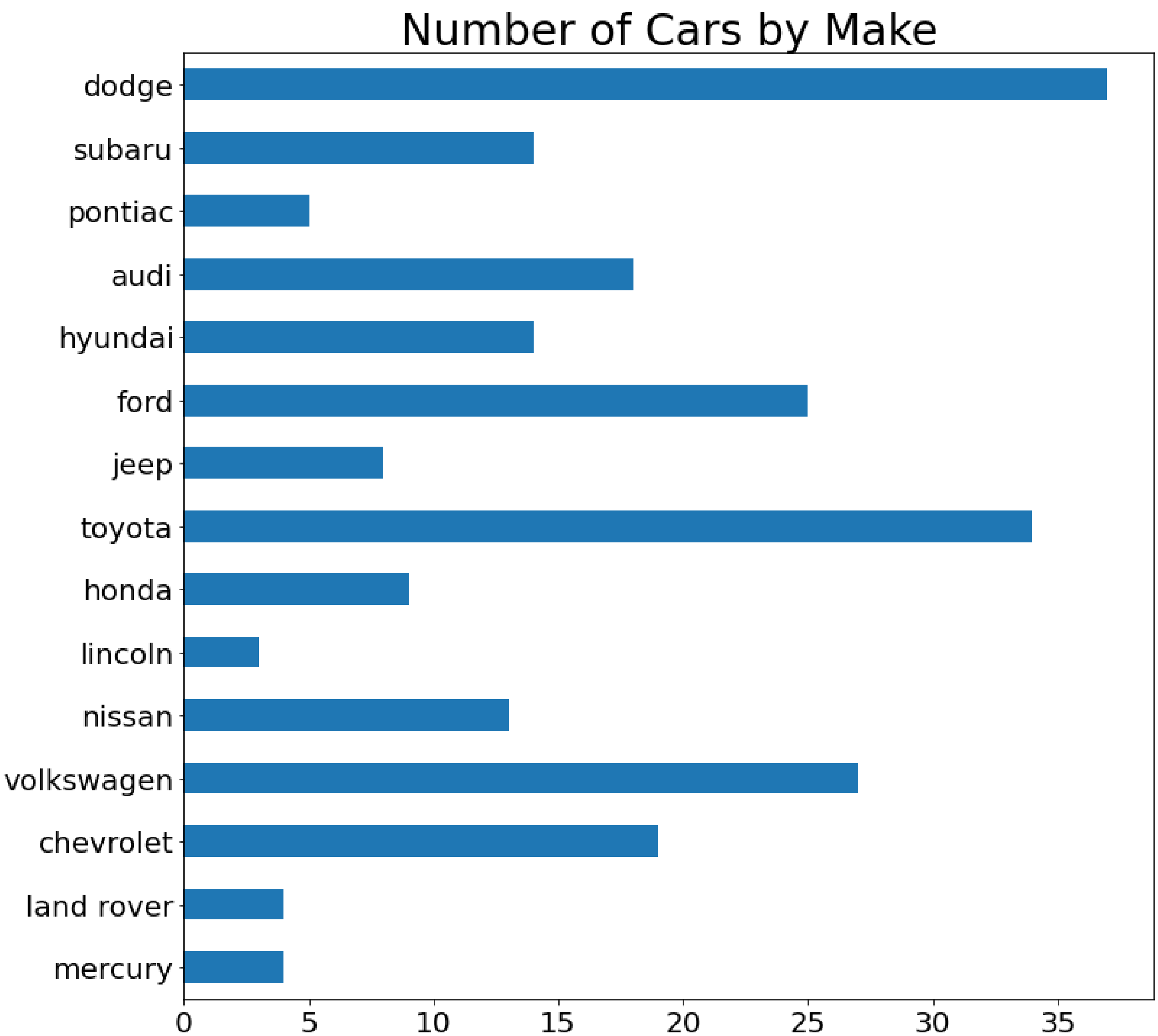
date	value
2000-01-01	0.218938
2000-01-02	0.195322
2000-01-03	-0.397765
2000-01-04	0.359213
2000-01-05	1.760460

mpg

manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact
audi	a4	2.0	2008	4	manual(m6)	f	20	31	p	compact
audi	a4	2.0	2008	4	auto(av)	f	21	30	p	compact
audi	a4	2.8	1999	6	auto(l5)	f	16	26	p	compact

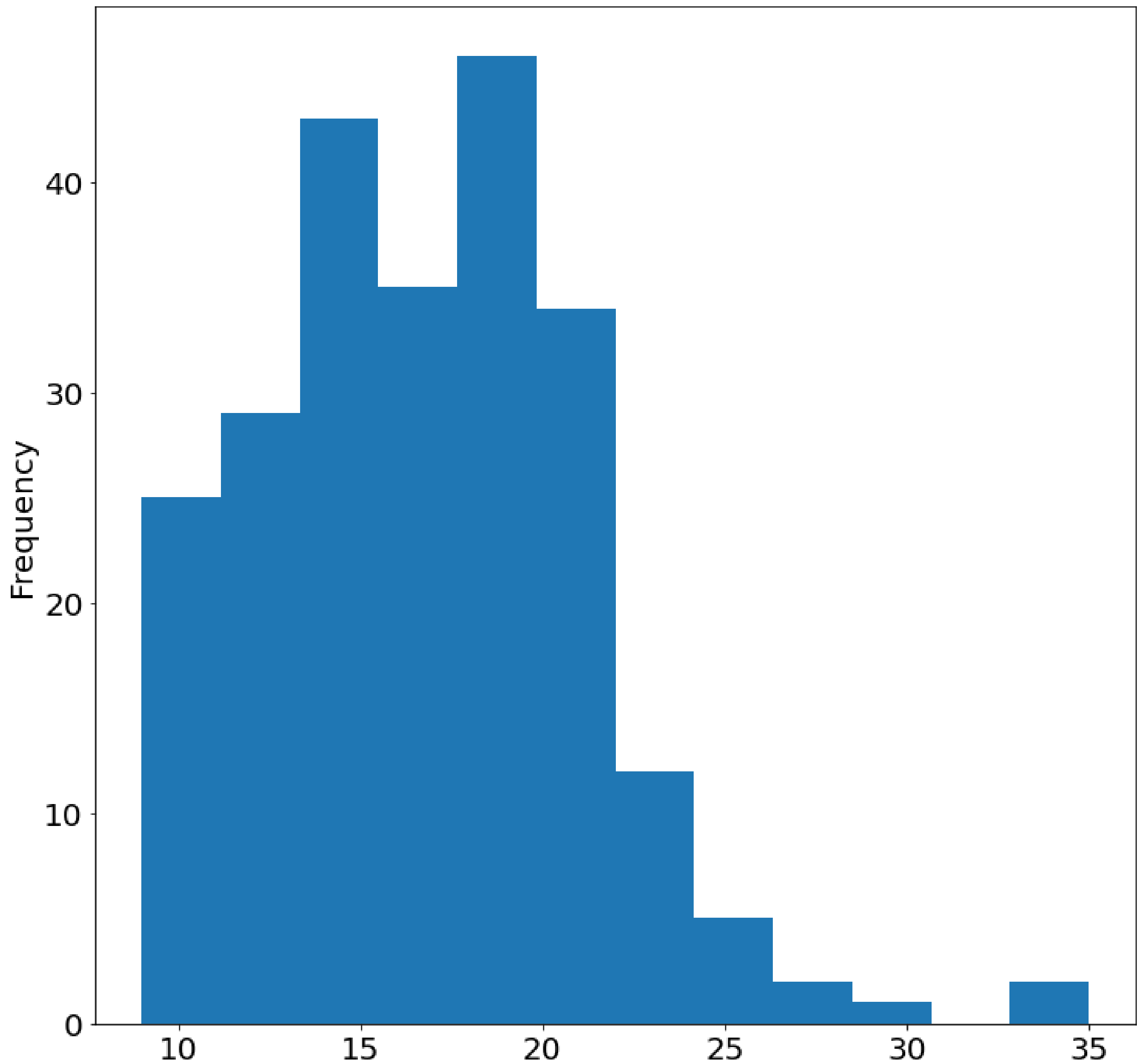
diamonds

carat	cut	color	clarity	depth	table	price	x	y	z
0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75



Code:

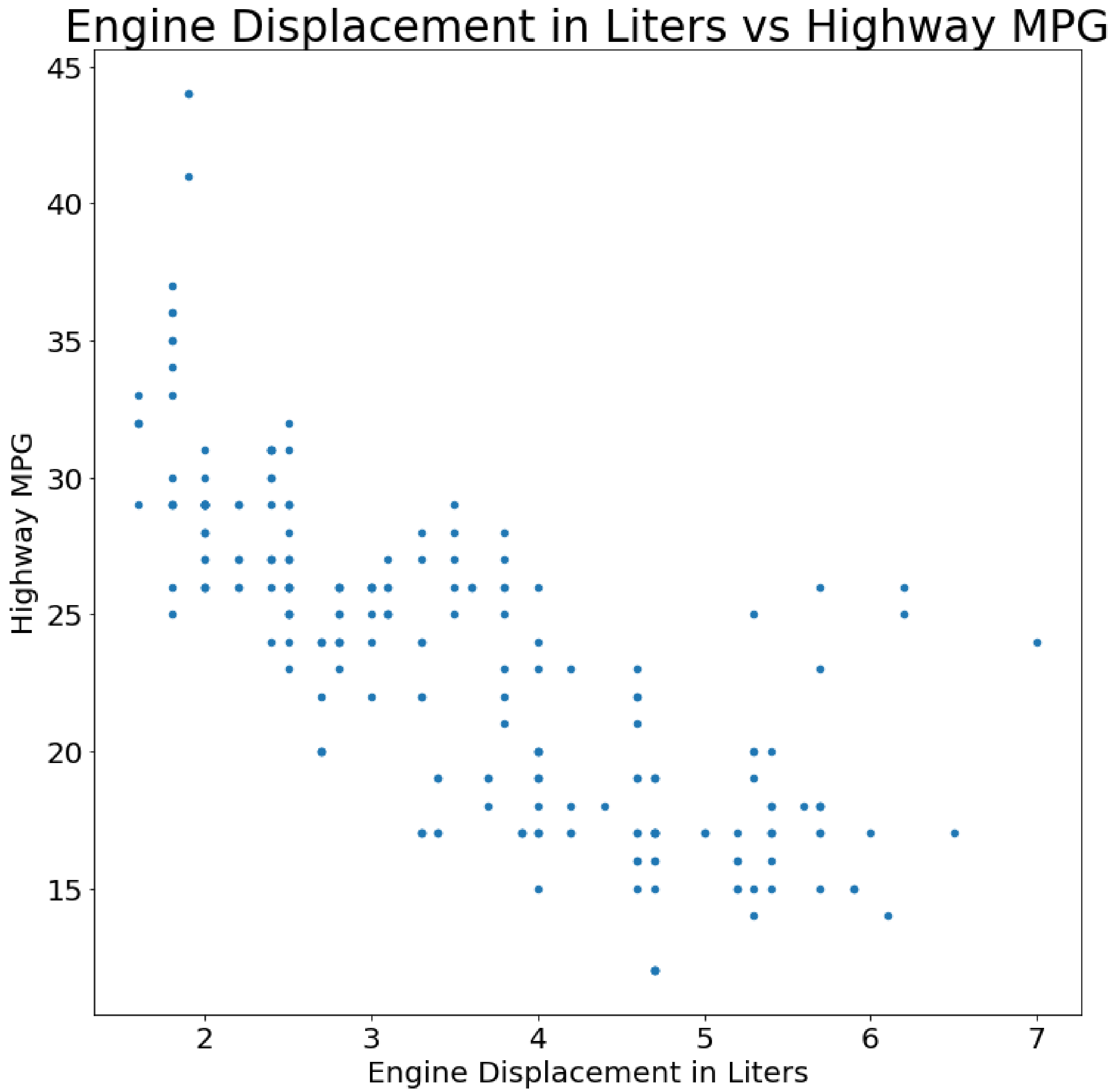
```
(mpg['manufacturer']
 .value_counts(sort=False)
 .plot.barh()
 .set_title('Number of Cars by
Make')
)
```



Code:

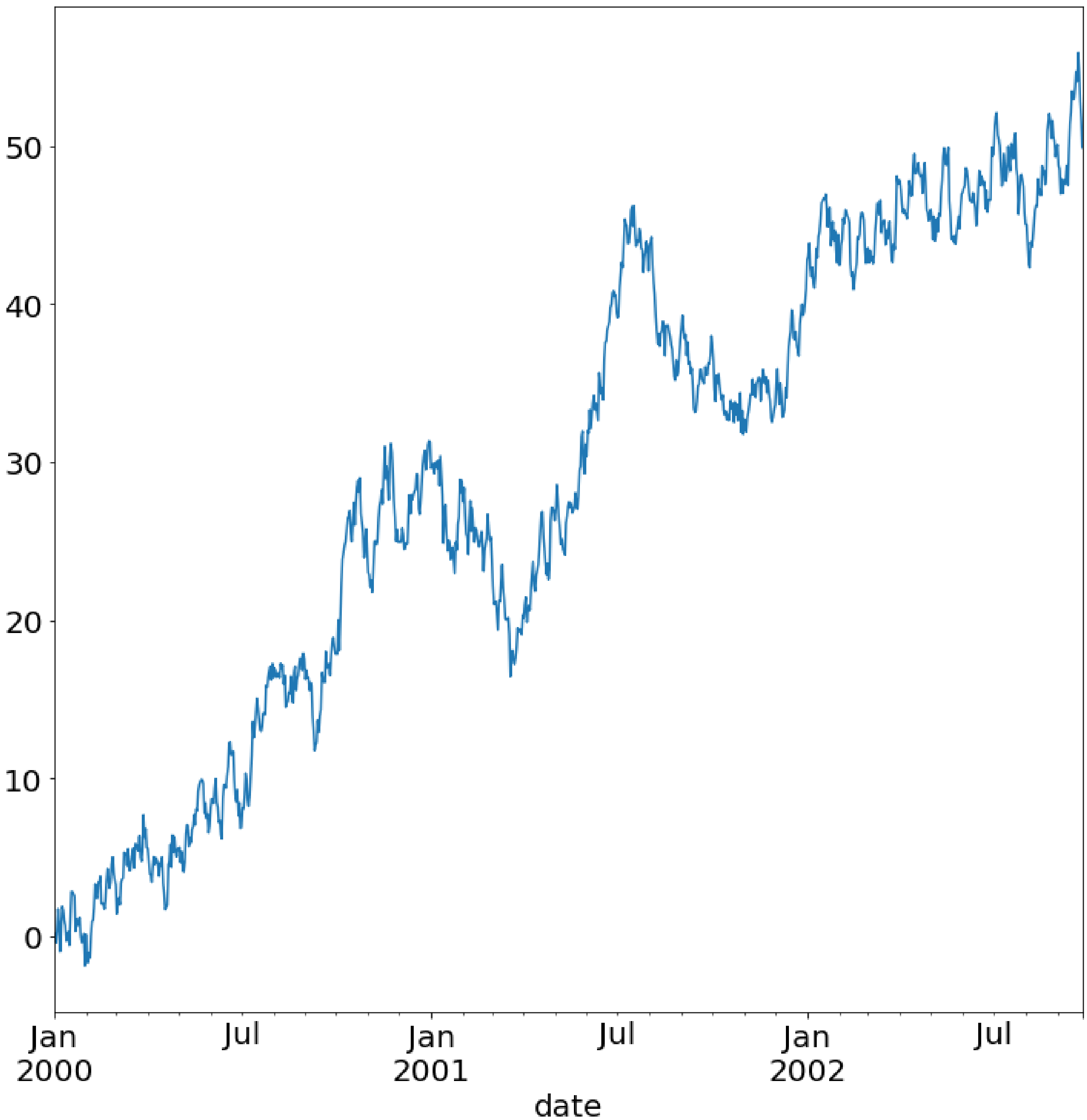
```
(mpg['cty']
 .plot

.hist(bins=1
```



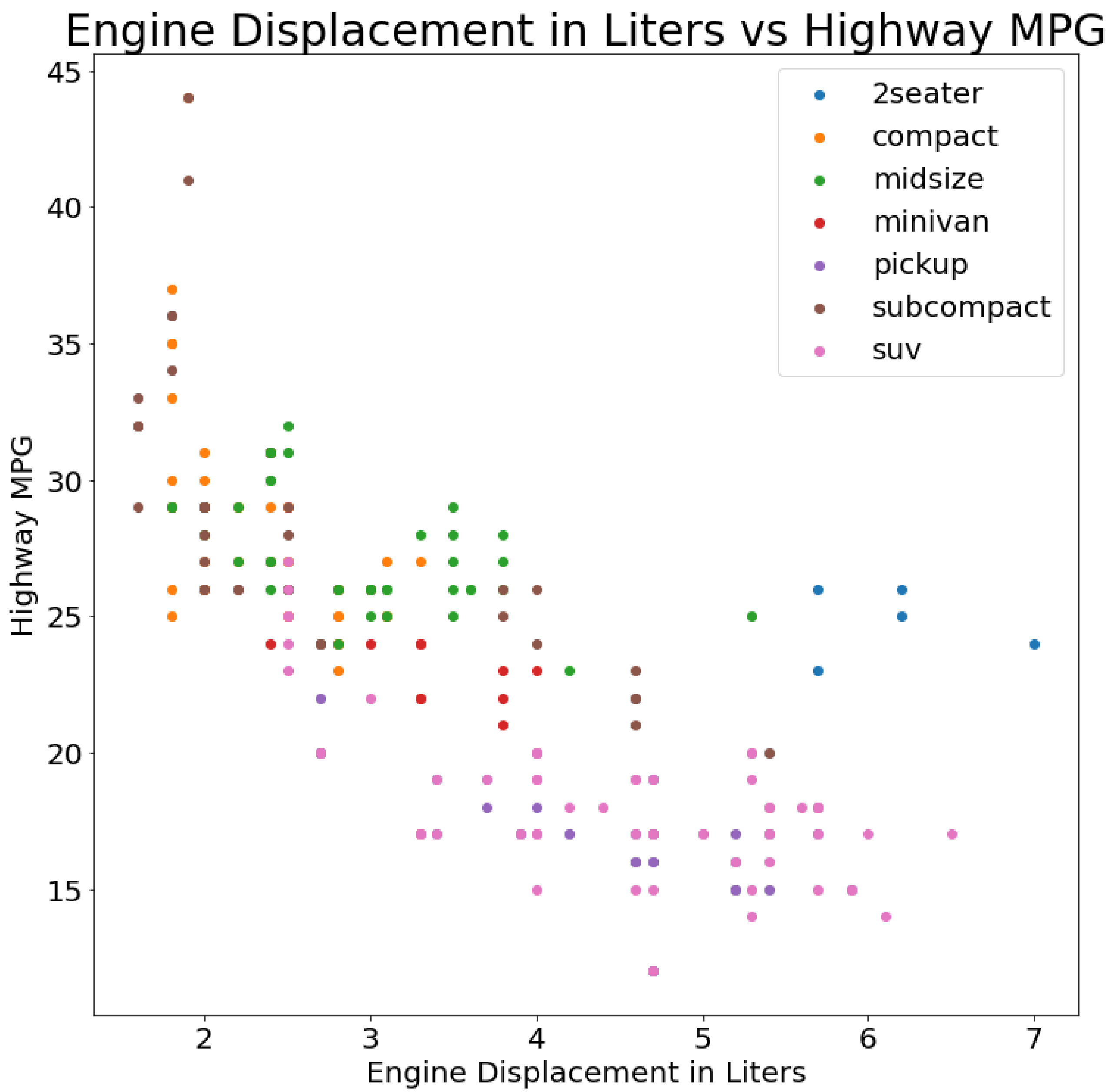
Code:

```
(mpg
 .plot
 .scatter(x='displ', y='hwy')
 .set(title='Engine Displacement in Liters vs Highway
MPG',
      xlabel='Engine Displacement in Liters',
      ylabel='Highway MPG'))
```



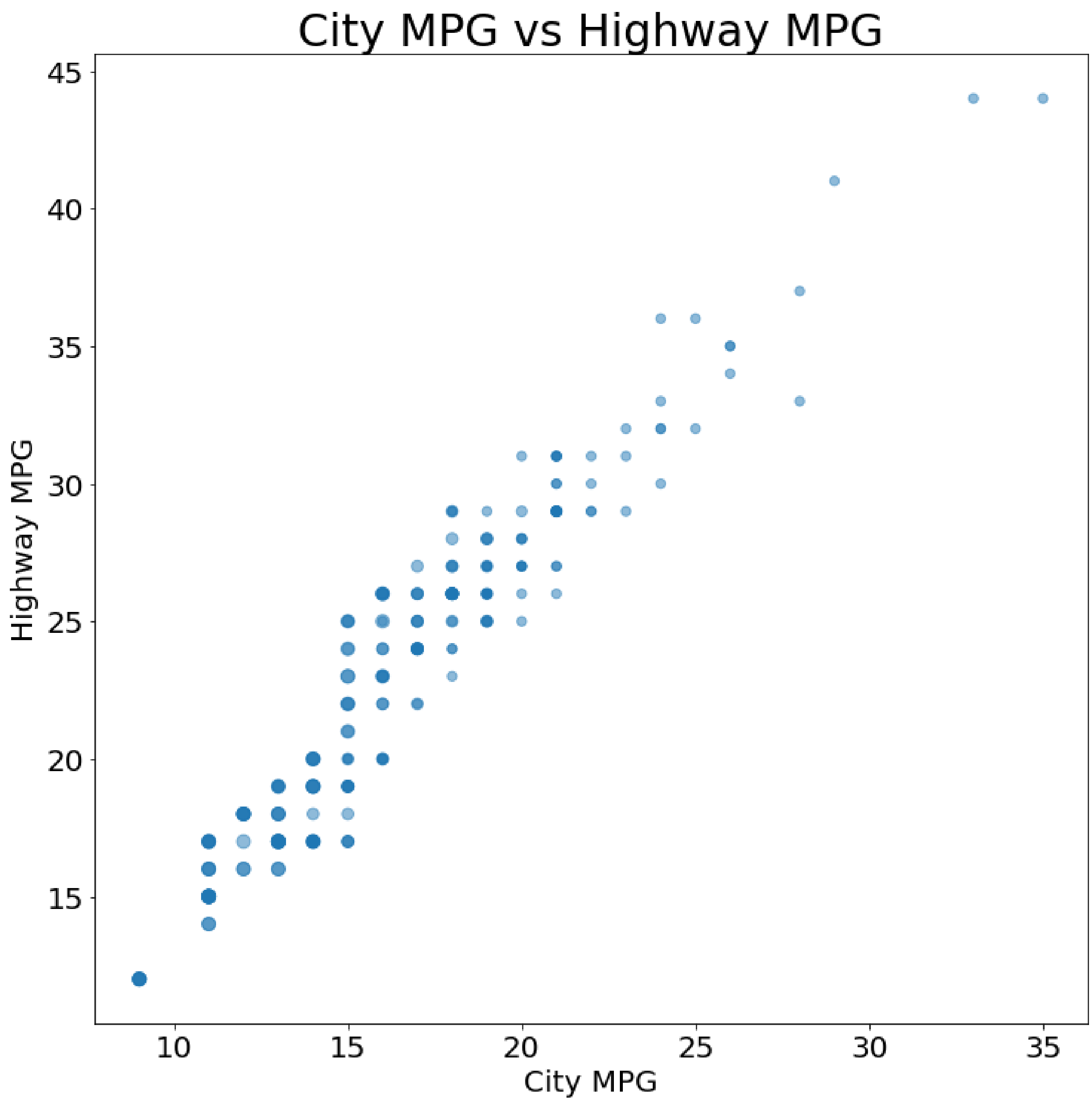
Code:

```
ts.set_index('date')
['value'].plot()
```



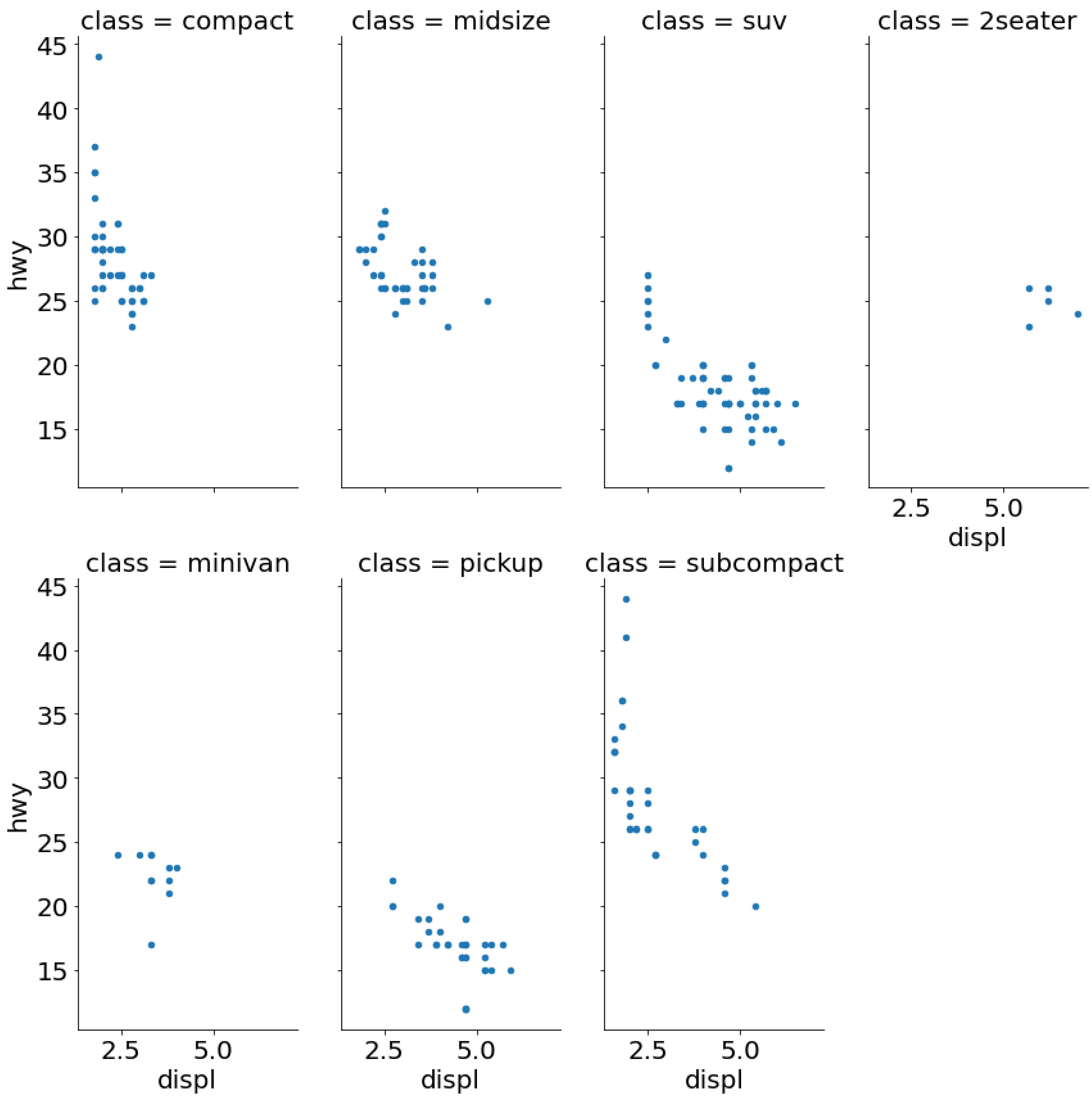
Code:

```
fig, ax = pyplot.subplots()
for c, df in mpg.groupby('class'):
    ax.scatter(df['displ'], df['hwy'], label=c)
ax.legend()
ax.set_title('Engine Displacement in Liters vs Highway
MPG')
ax.set_xlabel('Engine Displacement in Liters')
ax.set_ylabel('Highway MPG')
```



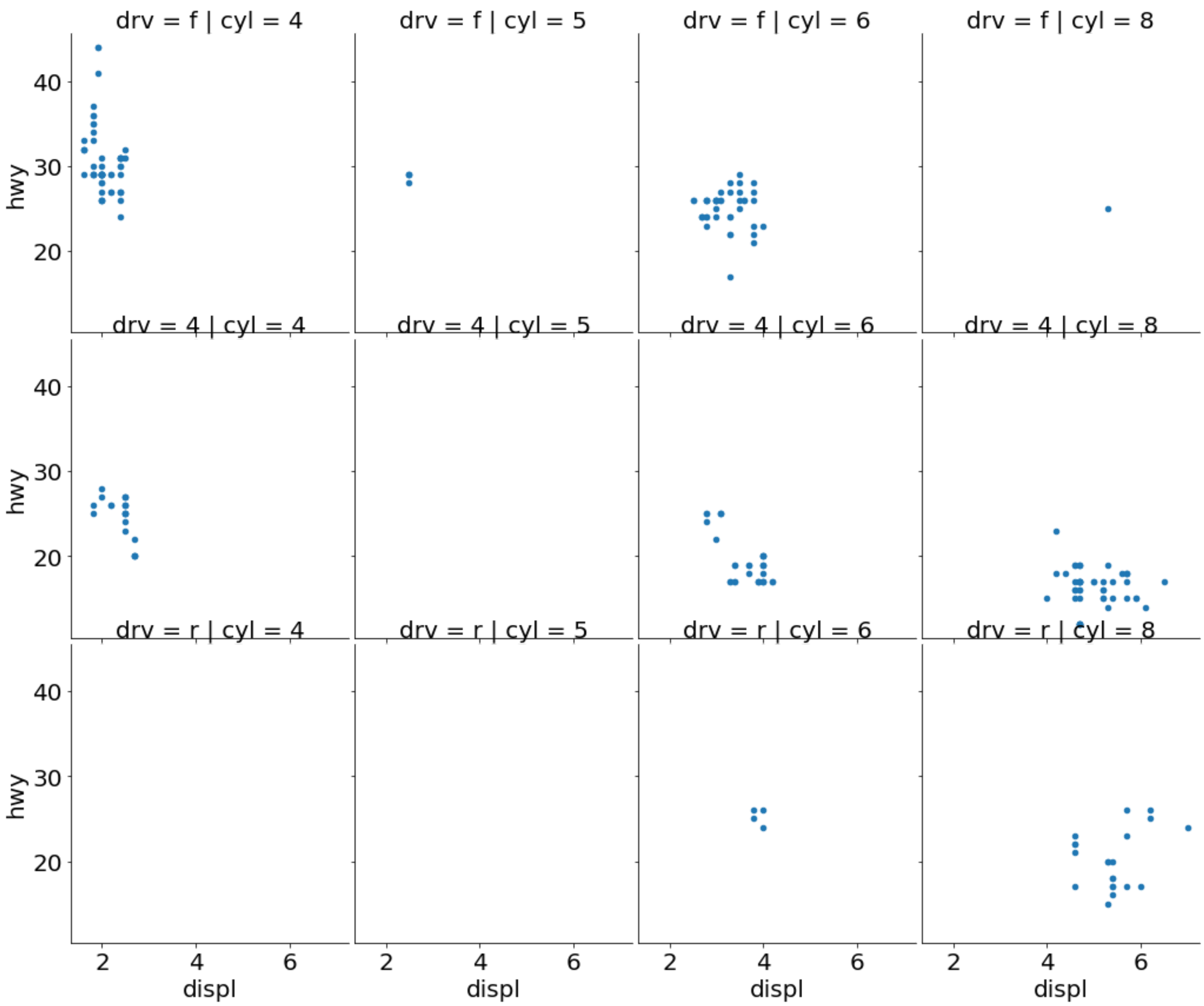
Code:

```
ax = (mpg
    .plot
    .scatter(x='cty',
             y='hwy',
             s=10*mpg['cyl'],
             alpha=.5))
ax.set_title('City MPG vs Highway
MPG')
ax.set_xlabel('City MPG')
ax.set_ylabel('Highway MPG')
```



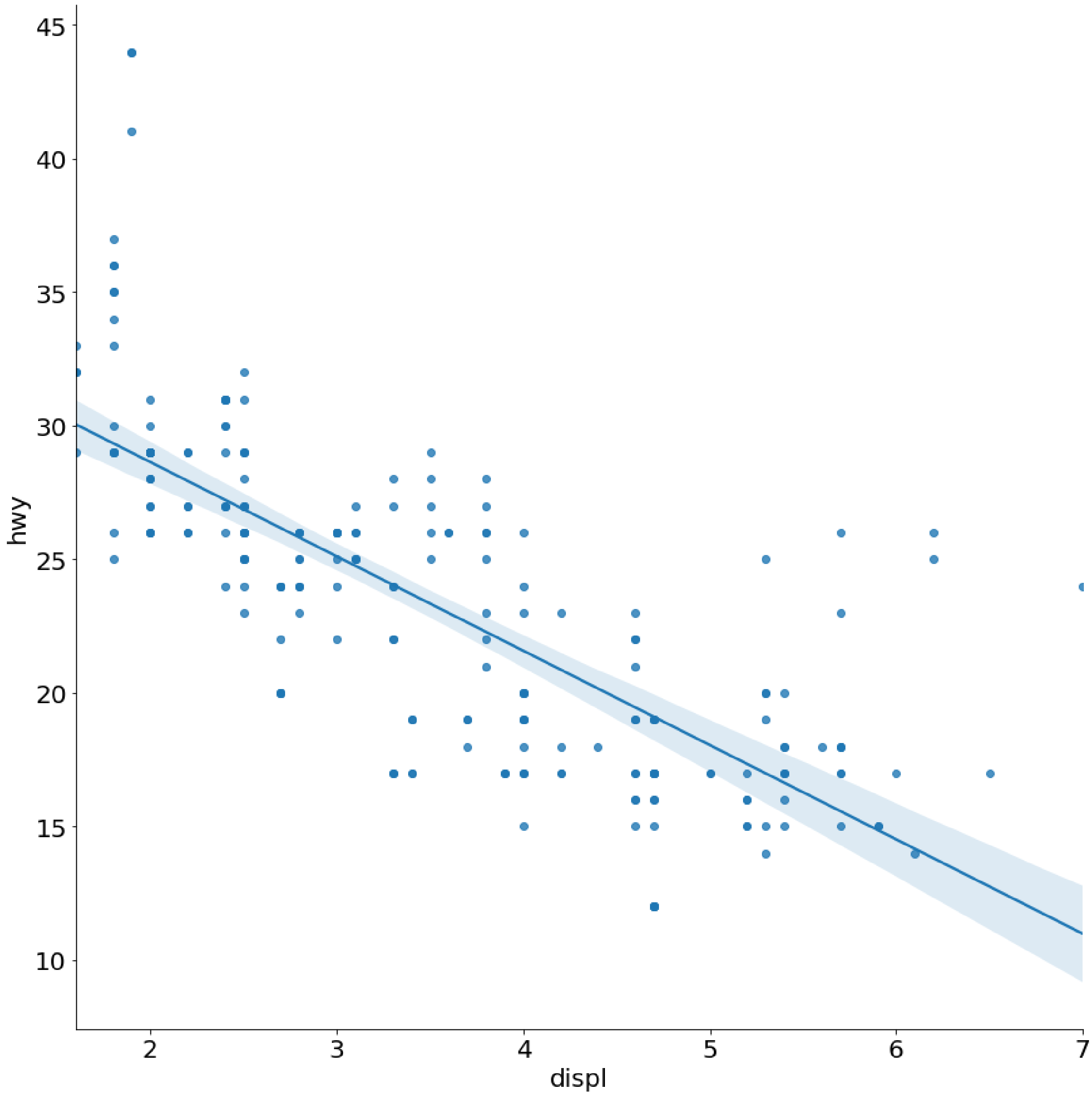
Code:

```
(mpg
 .pipe(sns.FacetGrid,
       col='class',
       col_wrap=4,
       aspect=.5,
       size=6)
 .map(pyplot.scatter, 'displ', 'hwy',
s=20)
 .fig.subplots_adjust(wspace=.2,
hspace=.2)
)
```



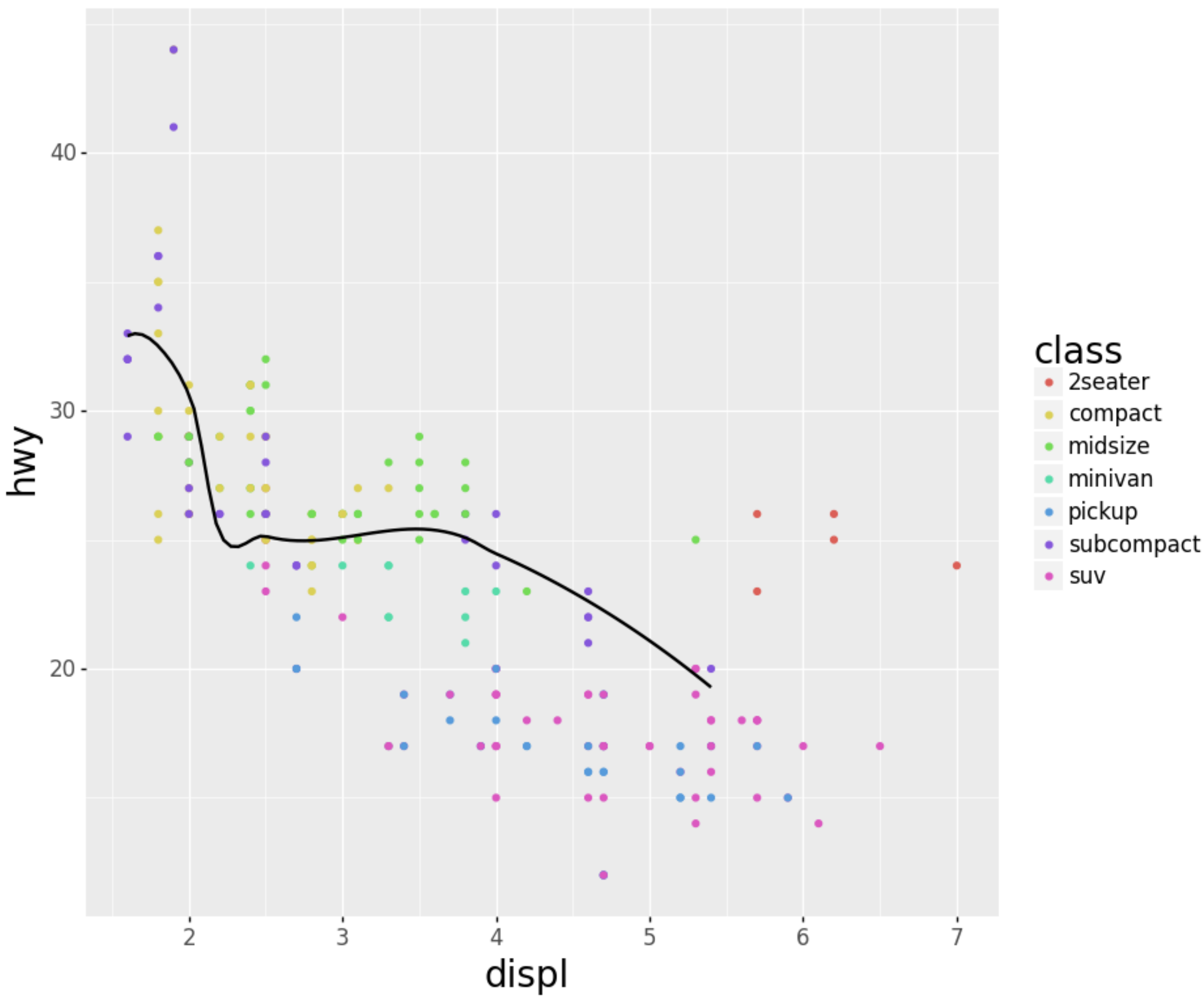
Code:

```
(mpg
 .pipe(sns.FacetGrid,
       col='cyl',
       row='drv',
       aspect=.9,
       size=4)
 .map(pyplot.scatter, 'displ', 'hwy',
s=20)
 .fig.subplots_adjust(wspace=.02,
hspace=.02)
)
```



Code:

```
sns.lmplot(x='displ',
y='hwy',
           data=mpg,
size=12)
```

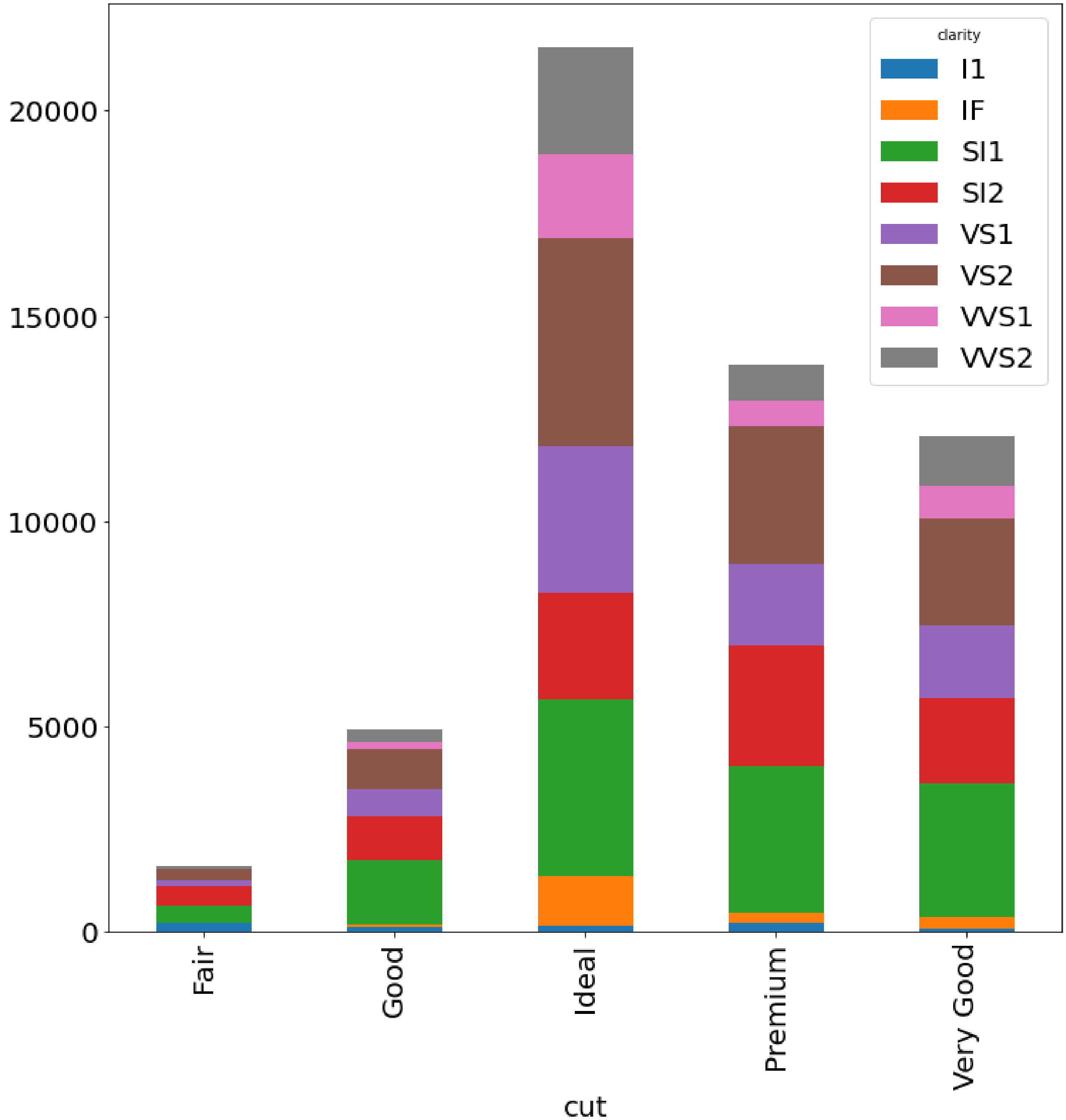


Code:

```
(ggplot(data=mpg,
        mapping=aes(x='displ', y='hwy')) +
  geom_point(mapping=aes(color = 'class')) +
  geom_smooth(data=mpg[mpg['class'] == 'subcompact'],
              se=False,
              method = 'loess'
             ))
```

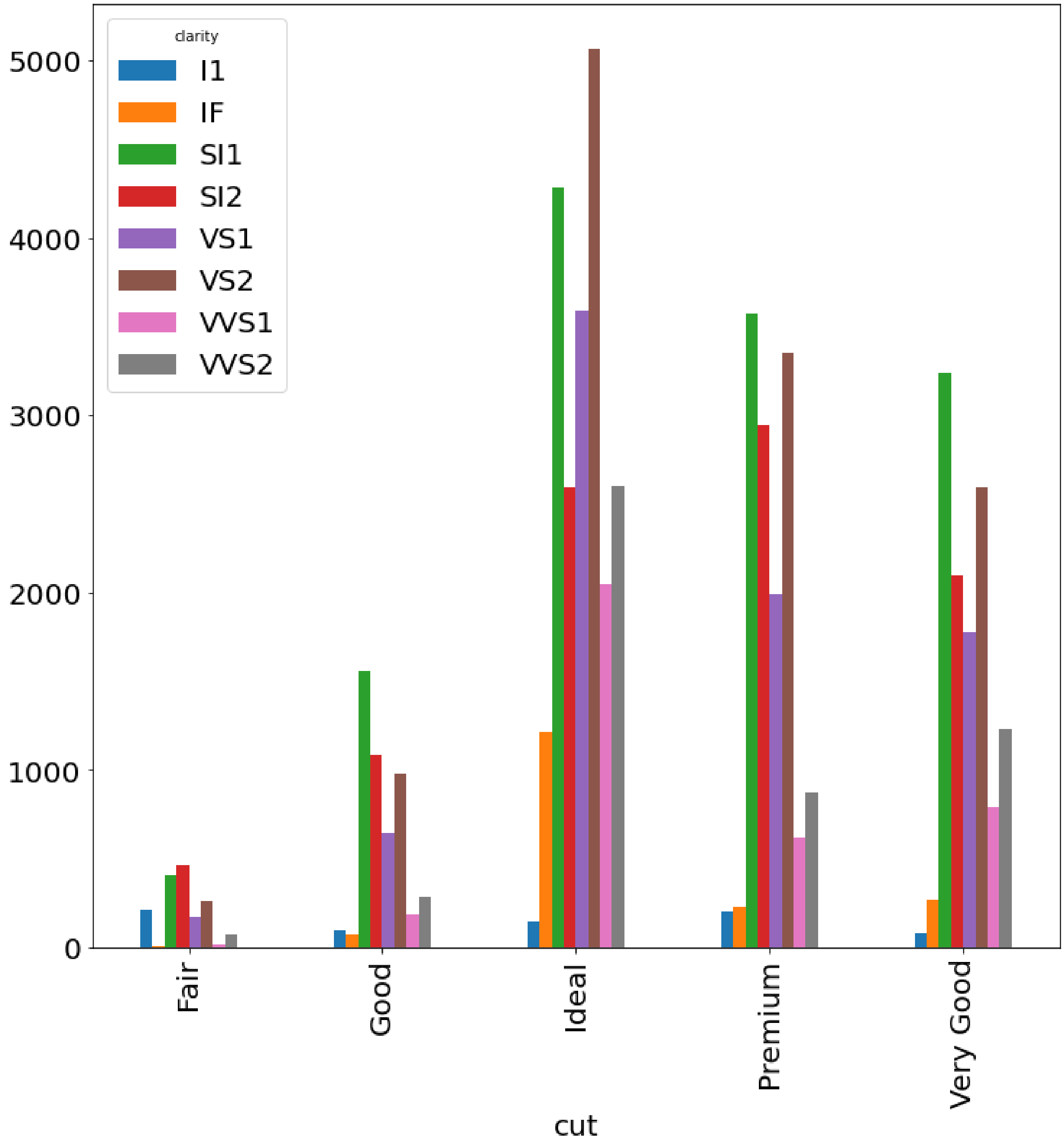
Note:

Notice the smoothed line isn't as smooth as it is in ggplot2.



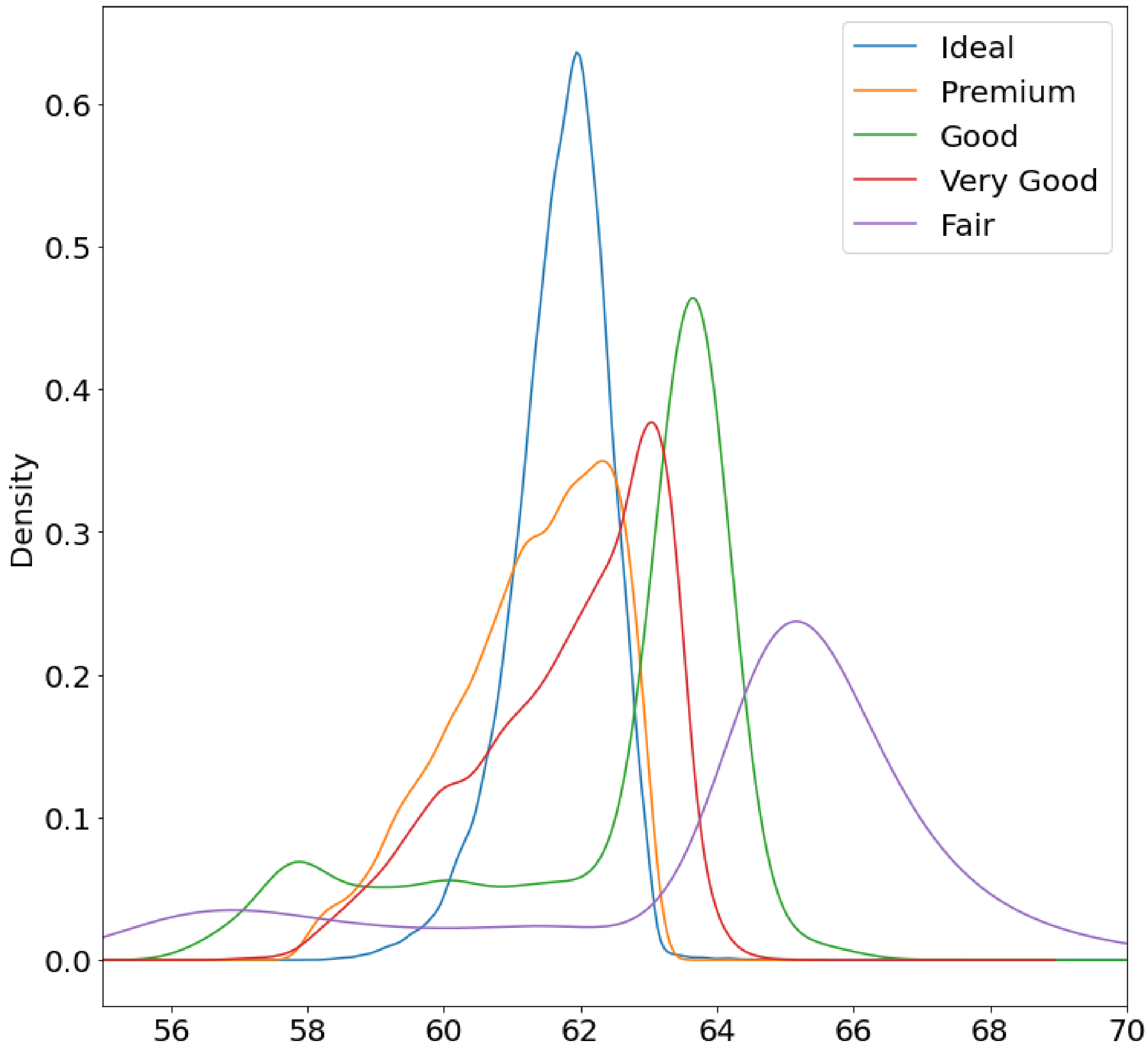
Code:

```
(diamonds
 .groupby(['cut',
'clarity'])
 .size()
 .unstack()
 .plot.bar(stacked=True)
)
```



Code:

```
(diamonds
 .groupby(['cut',
'clarity'])
 .size()
 .unstack()
 .plot.bar()
)
```



Code:

```
fig, ax = pyplot.subplots()
ax.set_xlim(55, 70)
for cut in diamonds['cut'].unique():
    s = diamonds[diamonds['cut'] == cut]['depth']
    s.plot.kde(ax=ax, label=cut)
ax.legend()
```

Note:

I don't know whether Pandas can fill a KDE curve.

This requires using some Matplotlib to get them to stack and to have a legend.