python_vis

May 13, 2020

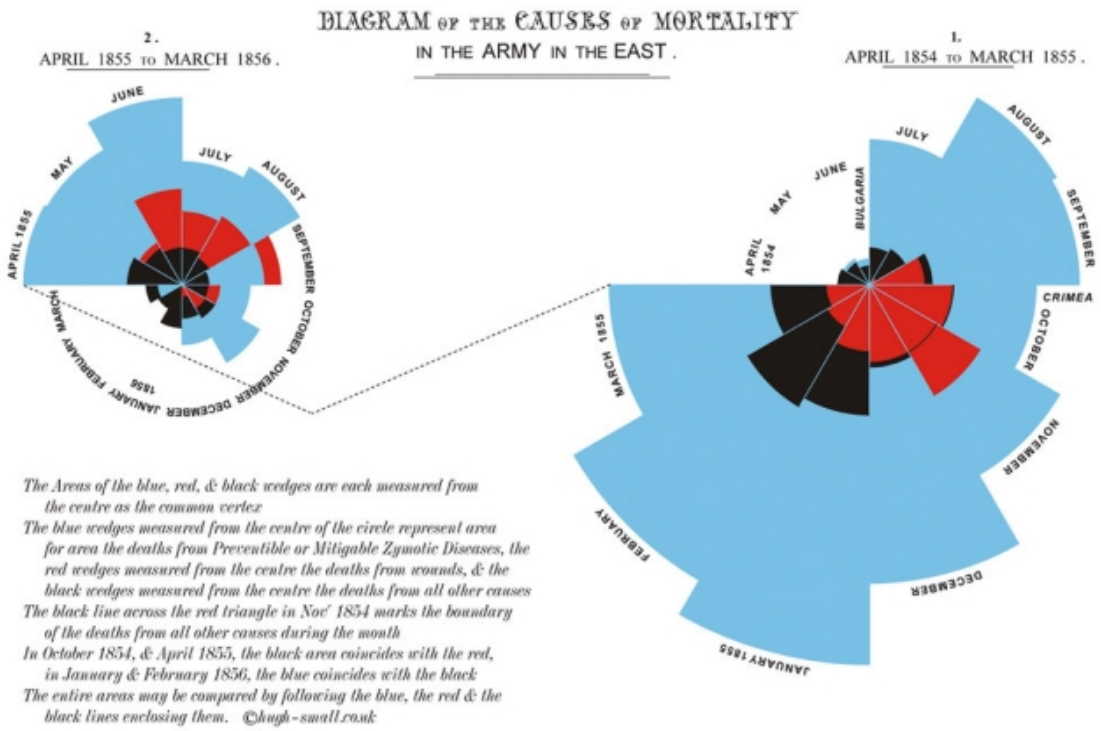
1 Data visualization using Python

1.1 Introduction

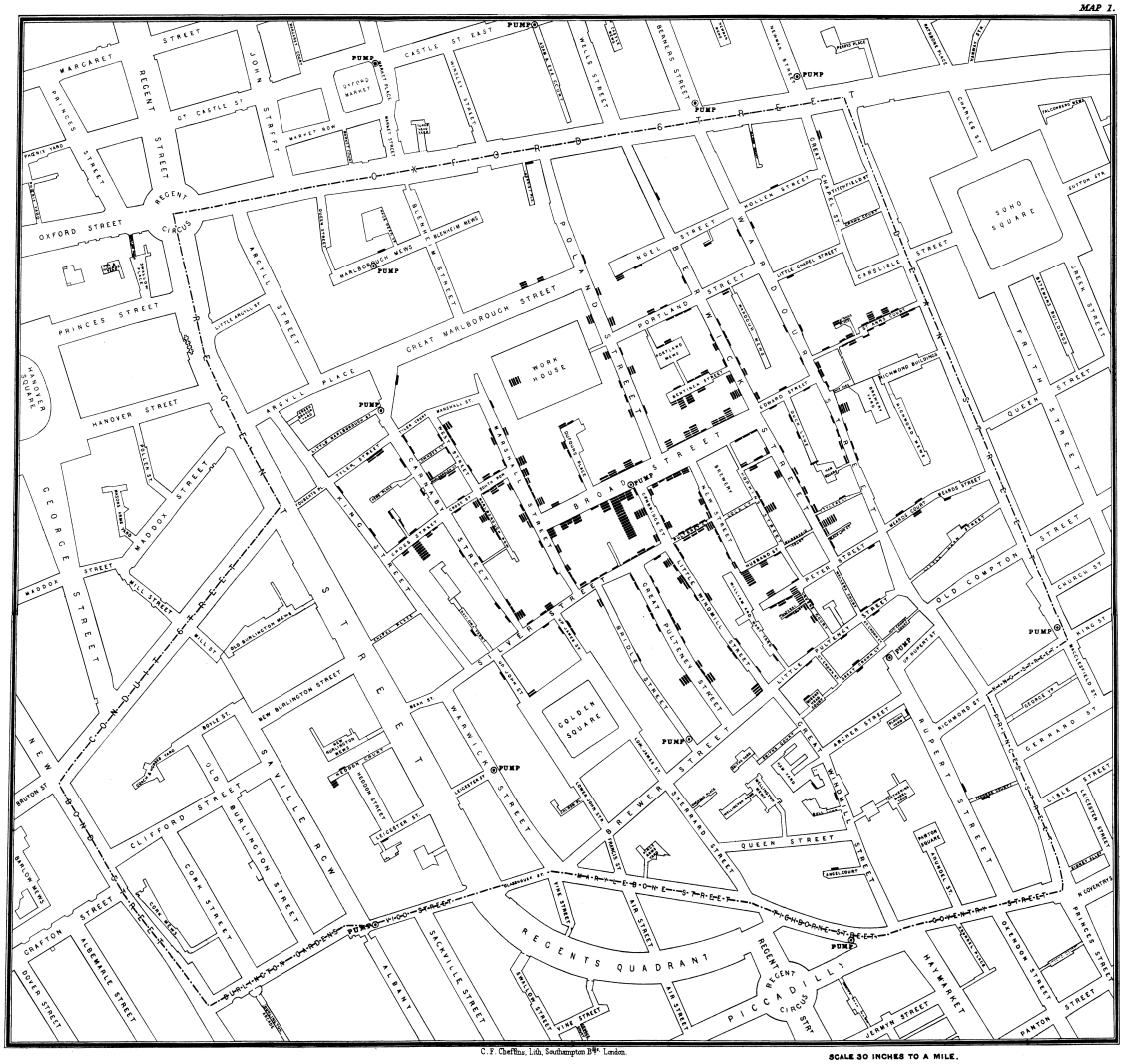
Data visualization is a basic task in data exploration and understanding. Humans are mainly visual creatures, and data visualization provides an opportunity to enhance communication of the story within the data. Often we find that data and the data-generating process is complex, and a visual representation of the data and our innate ability at pattern recognition can help reveal the complexities in a cognitively accessible way.

1.1.1 An example gallery

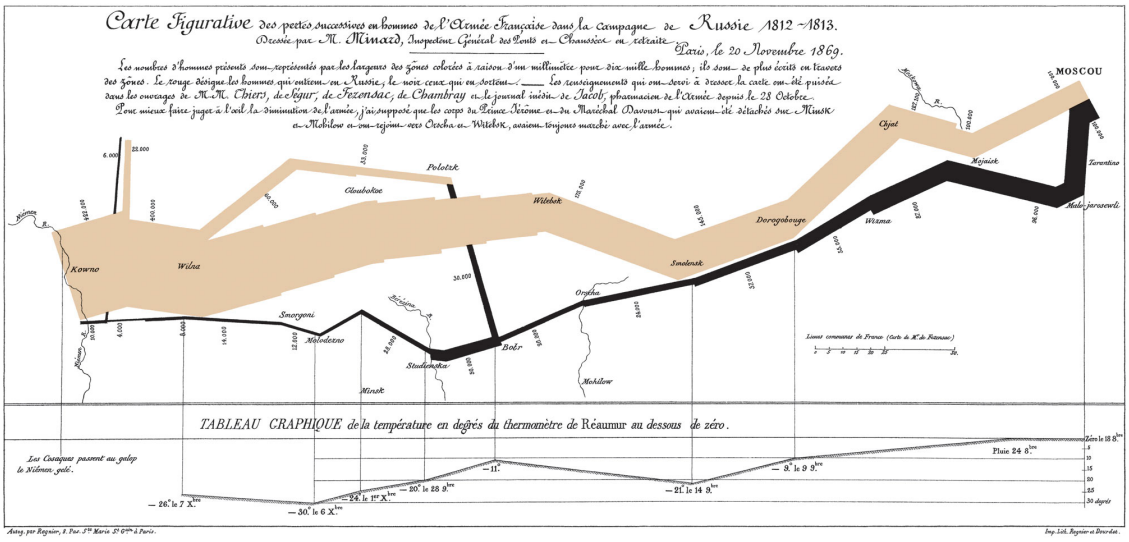
Data visualization has a long and storied history, from Florence Nightangle onwards. Dr. Nightangle was a pioneer in data visualization and developed the *rose plot* to represent causes of death in hospitals during the Crimean War.



John Snow, in 1854, famously visualized the cholera outbreak in London, which showed the geographic proximity of cholera prevalence with particular water wells.



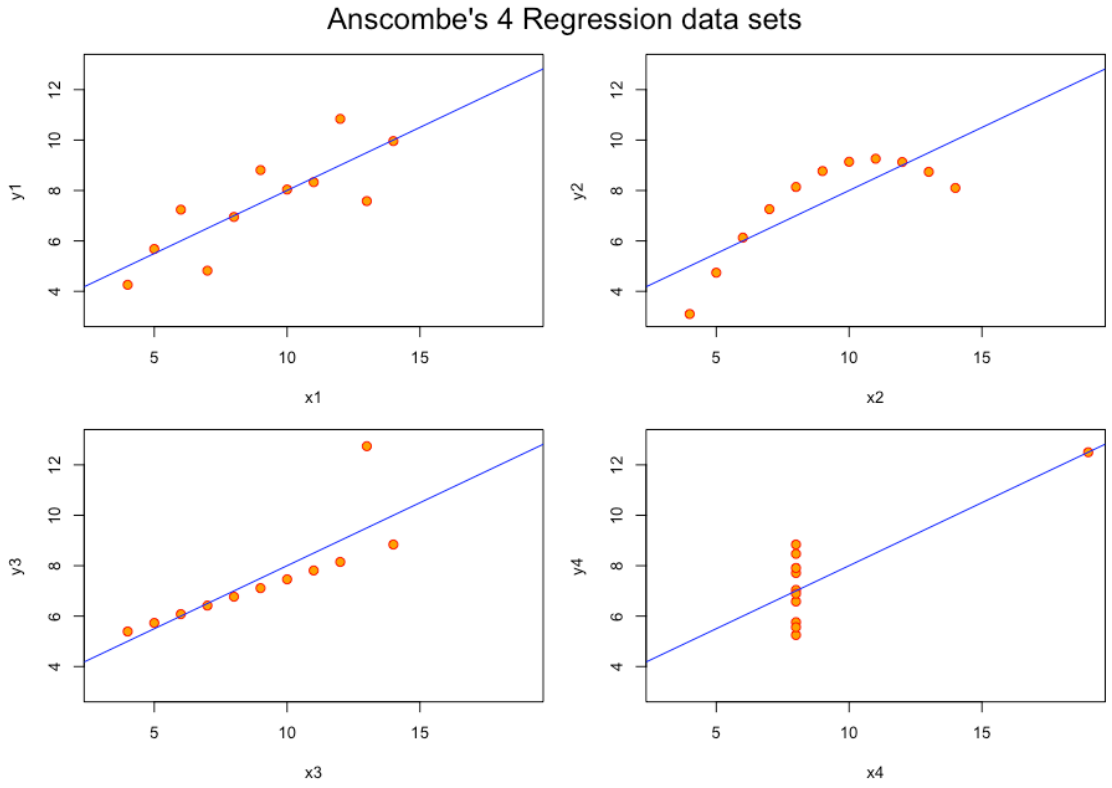
In one of the more famous visualizations, considered by many to be an optimal use of display ink and space, Minard visualized Napoleon's disastrous campaign to Russia



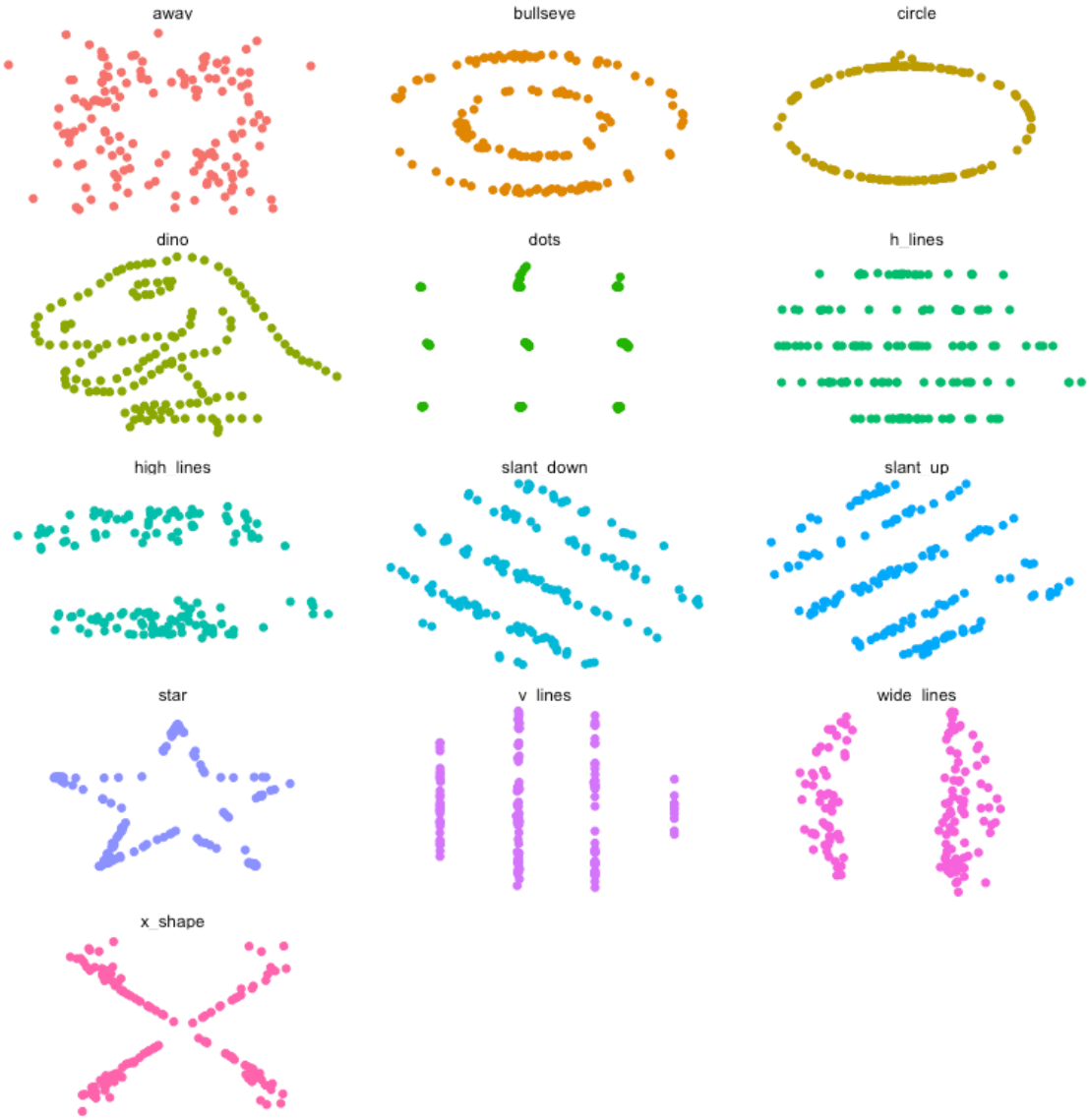
In more recent times, an employee at Facebook visualized all connections between users across the world, which clearly showed geographical associations with particular countries and regions.

1.1.2 Why visualize data?

We often rely on numerical summaries to help understand and distinguish datasets. In 1973, Anscombe published an influential set of 4 datasets, each with two variables and with the means, variances and correlations being identical. When you graphed these data, the differences in the datasets were clearly visible. This set is popularly known as Anscombe's quartet.



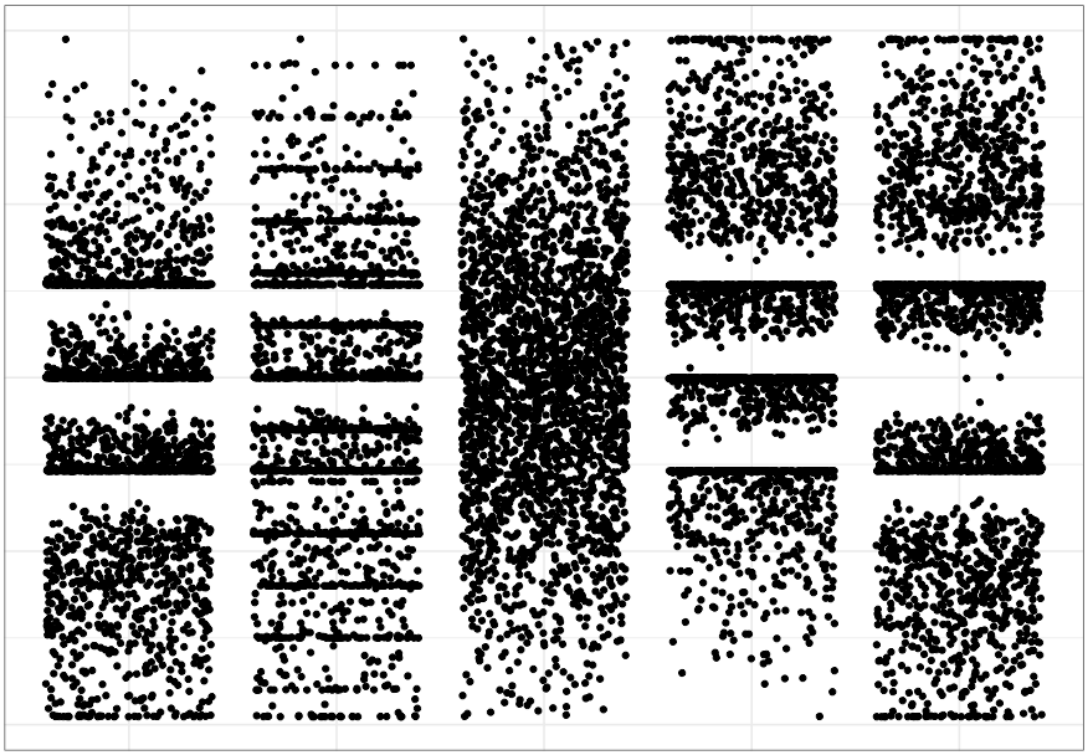
A more recent experiment in data construction by Matejka and Fitzmaurice (2017) started with a representation of a dinosaur and created 10 more bivariate datasets which all shared the same univariate means and variances and the same pairwise correlations.

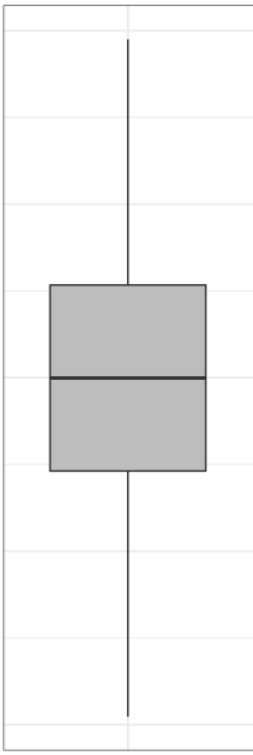


These examples clarify the need for visualization to better understand relationships between variables.

Even when using statistical visualization techniques, one has to be careful. Not all visualizations can discriminate between statistical characteristics. This was also explored by Matejka and Fitzmaurice.

Strip plot





1.1.3 Conceptual ideas

Begin with the consumer in mind

- You have a deep understanding of the data you're presenting
- The person seeing the visualization **doesn't**
- Develop simpler visualizations first that are easier to explain

Tell a story

- Make sure the graphic is clear
- Make sure the main point you want to make "pops"

A matter of perception

- Color (including awareness of color deficiencies)
- Shape
- Fonts

Some principles

1. Data-ink ratio
2. No mental gymnastics
    1. The graphic should be self-evident

    2. Context should be clear
3. Is a graph the wrong choice?
4. Focus on the consumer

   See my slides for some more opinionated ideas

1.2 Plotting in Python

Let's take a very quick tour before we get into the weeds. We'll use the mtcars dataset as an exemplar dataset that we can import using `pandas`

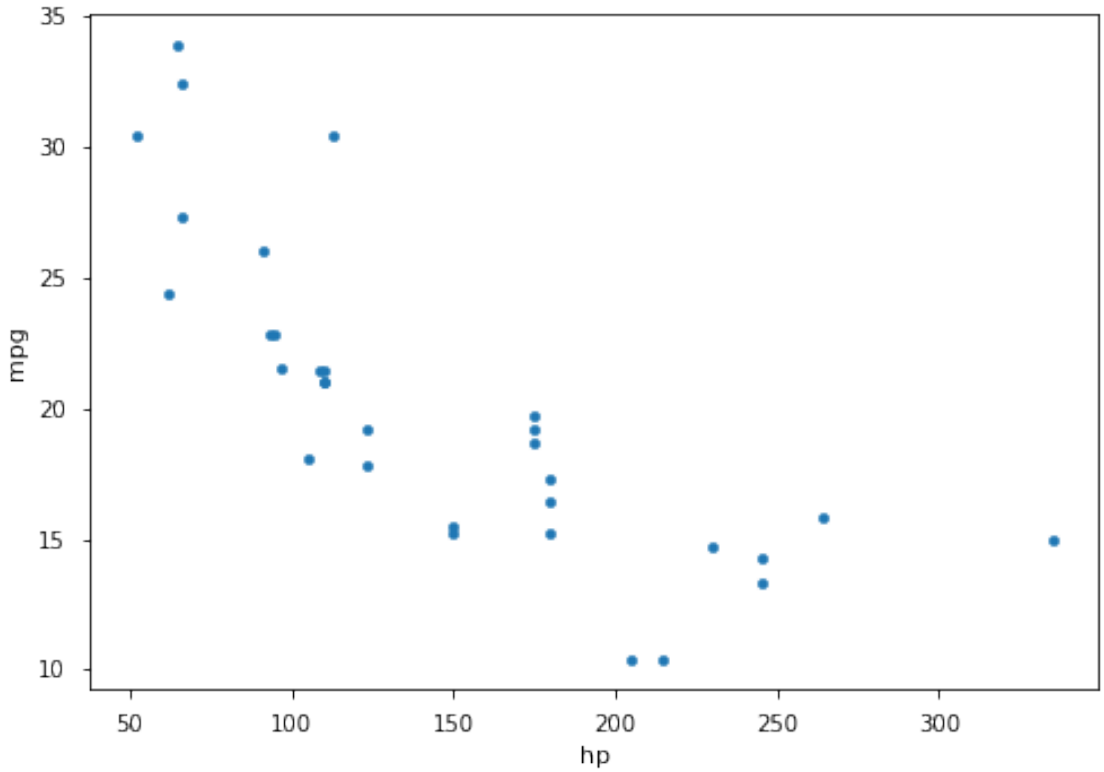
[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('seaborn-notebook')

mtcars = pd.read_csv('data/mtcars.csv')
```

1.2.1 Static plots

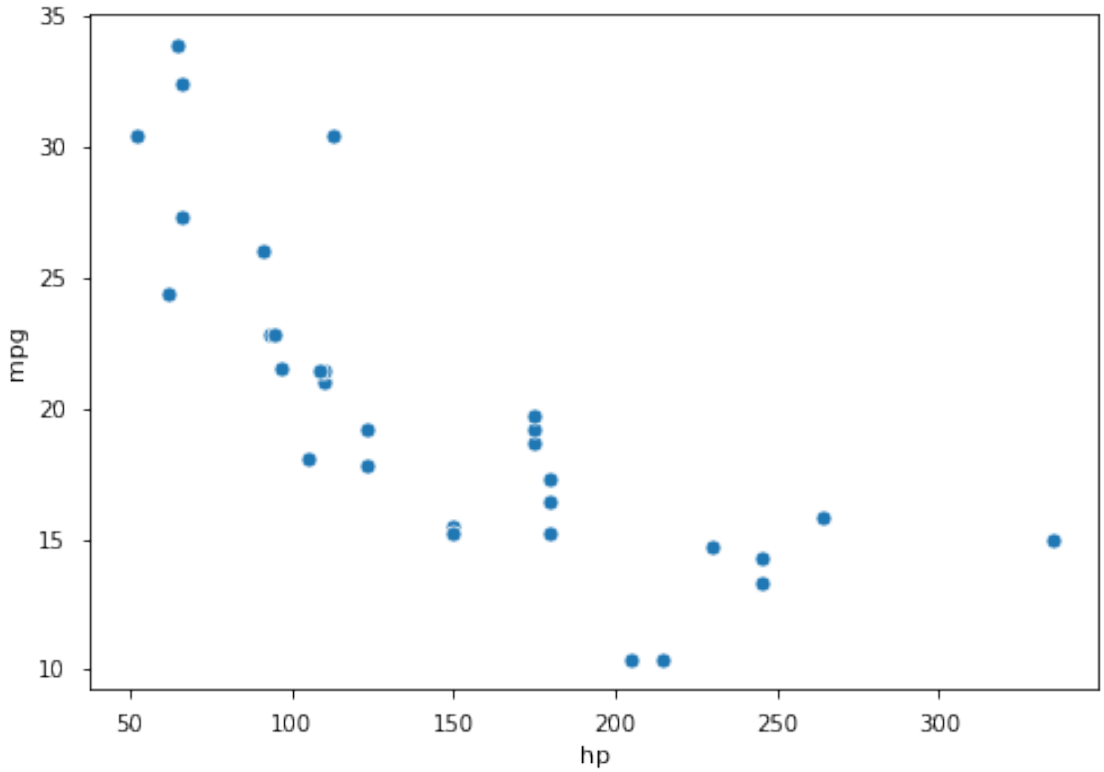
We will demonstrate plotting in what I'll call the `matplotlib` ecosystem. `matplotlib` is the venerable and powerful visualization package that was originally designed to emulate the Matlab plotting paradigm. It has since evolved and as become a bit more user-friendly. It is still quite granular and can facilitate a lot of custom plots once you become familiar with it. However, as a starting point, I think it's a bit much. We'll see a bit of what it can offer later.

We will consider two other options which are built on top of `matplotlib`, but are much more accessible. These are `pandas` and `seaborn`. The two packages have some different approaches, but both wrap `matplotlib` in higher-level code and decent choices so we don't need to get into the `matplotlib` trenches quite so much. We'll still call `matplotlib` in our code, since both these packages need it for some fine tuning. Both packages are also very much aligned to the `DataFrame` construct in `pandas`, so makes plotting a much more seamless experience.

[2]:
```
mtcars.plot.scatter(x = 'hp', y = 'mpg');
# mtcars.plot(x = 'hp', y = 'mpg', kind = 'scatter');
```



[4]:
```
import seaborn as sns
sns.scatterplot(data = mtcars, x = 'hp', y = 'mpg');
```

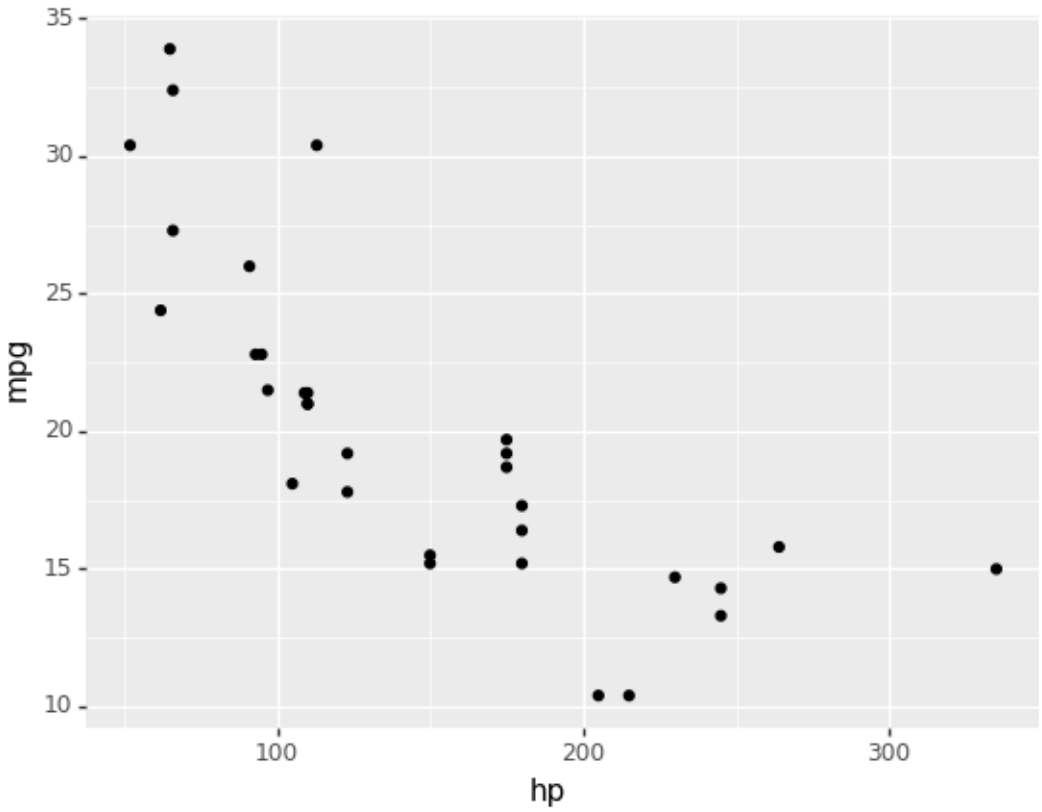


There are of course some other choices based on your background and preferences. For static plots, there are a couple of emulators of the popular R package `ggplot2`. These are `plotnine` and `ggplot`. `plotnine` seems a bit more developed and uses the `ggplot2` semantics of aesthetics and layers, with almost identical code syntax.

You can install `plotnine` using `conda`:

```
conda install -c conda-forge plotnine
```

[5]:
```
from plotnine import *

(ggplot(mtcars) +
  aes(x = 'hp', y = 'mpg') +
  geom_point())
```



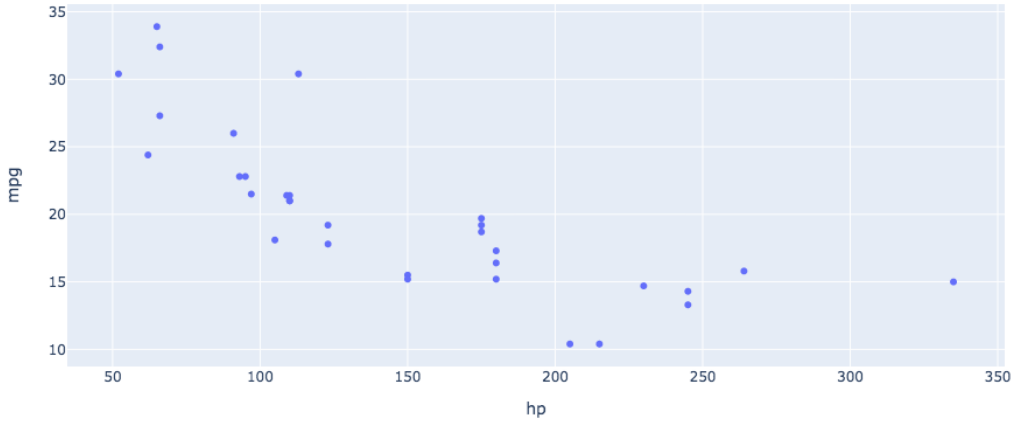
[5]: <ggplot: (305312696)>

1.2.2 Dynamic or interactive plots

There are several Python packages that wrap around Javascript plotting libraries that are so popular in web-based graphics like D3 and Vega. Three that deserve mention are `plotly`, `bokeh`, and `altair`.

`plotly` is a Python package developed by the company Plot.ly to interface with their interactive Javascript library either locally or via their web service. Plot.ly also develops an R package to interface with their products as well. It provides an intuitive syntax and ease of use, and is probably the more popular package for interactive graphics from both R and Python.

[6]:
```
import plotly.express as px

fig = px.scatter(mtcars, x = 'hp', y = 'mpg')
fig.show()
```



`bokeh` is an interactive visualization package developed by Anaconda. It is quite powerful, but its code can be rather verbose and granular

[7]:
```
from bokeh.plotting import figure, output_file
from bokeh.io import output_notebook, show
output_notebook()
p = figure()
p.xaxis.axis_label = 'Horsepower'
p.yaxis.axis_label = 'Miles per gallon'

p.circle(mtcars['hp'], mtcars['mpg'], size=10)

show(p)
```

`altair` that leverages ideas from Javascript plotting libraries and a distinctive code syntax that may appeal to some

[8]:
```
import altair as alt

alt.Chart(mtcars).mark_point().encode(
    x='hp',
    y='mpg'
).interactive()
```

[8]:
```
alt.Chart(…)
```

We won't focus on these dynamic packages in this workshop in the interests of time, but you can avail of several online resources for these.

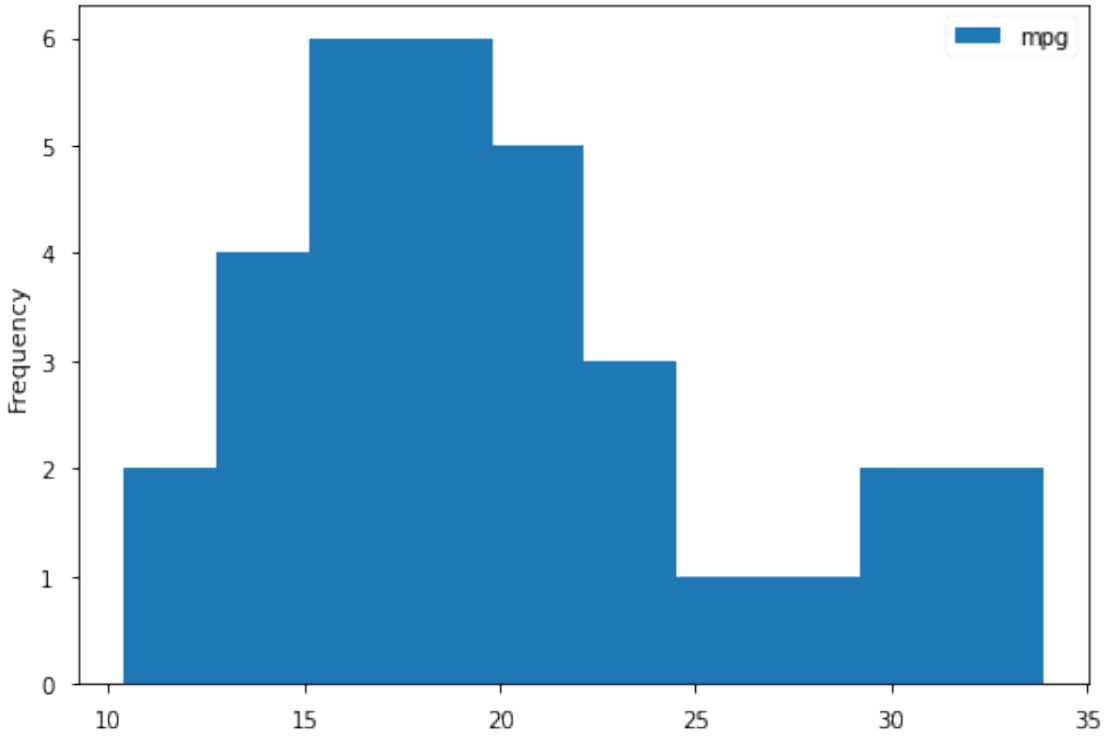
Package	Resources
plotly	Fundamentals
bokeh	Tutorial
altair	Overview

1.3 Univariate plots

1.3.1 pandas

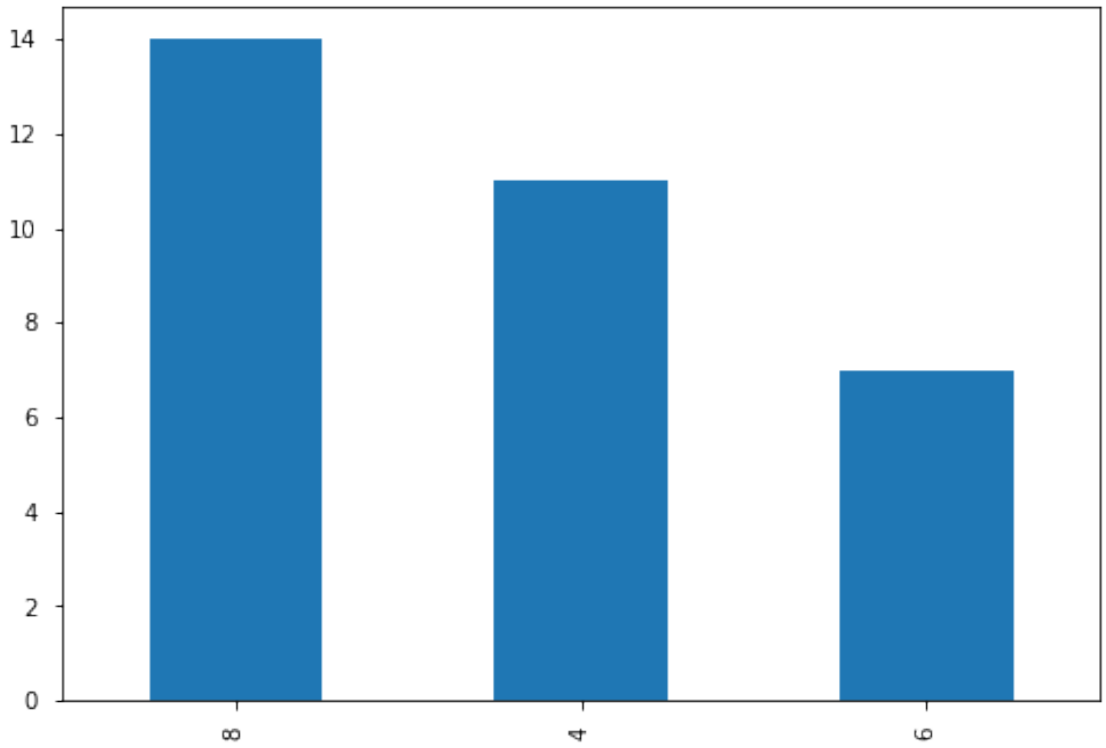
Histogram

[9]:
```
mtcars.plot.hist(y = 'mpg');

# mtcars.plot(y = 'mpg', kind = 'hist')
#mtcars['mpg'].plot(kind = 'hist')
```



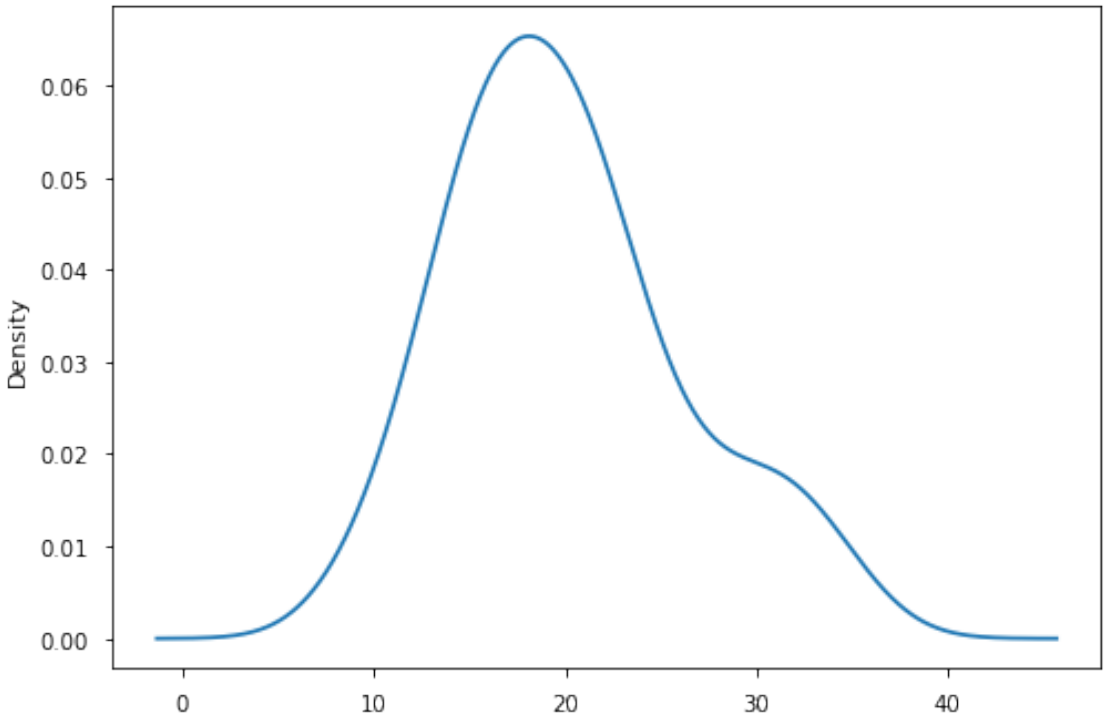
Bar plot

[10]:
```
mtcars['cyl'].value_counts().plot.bar();
```



Density plot

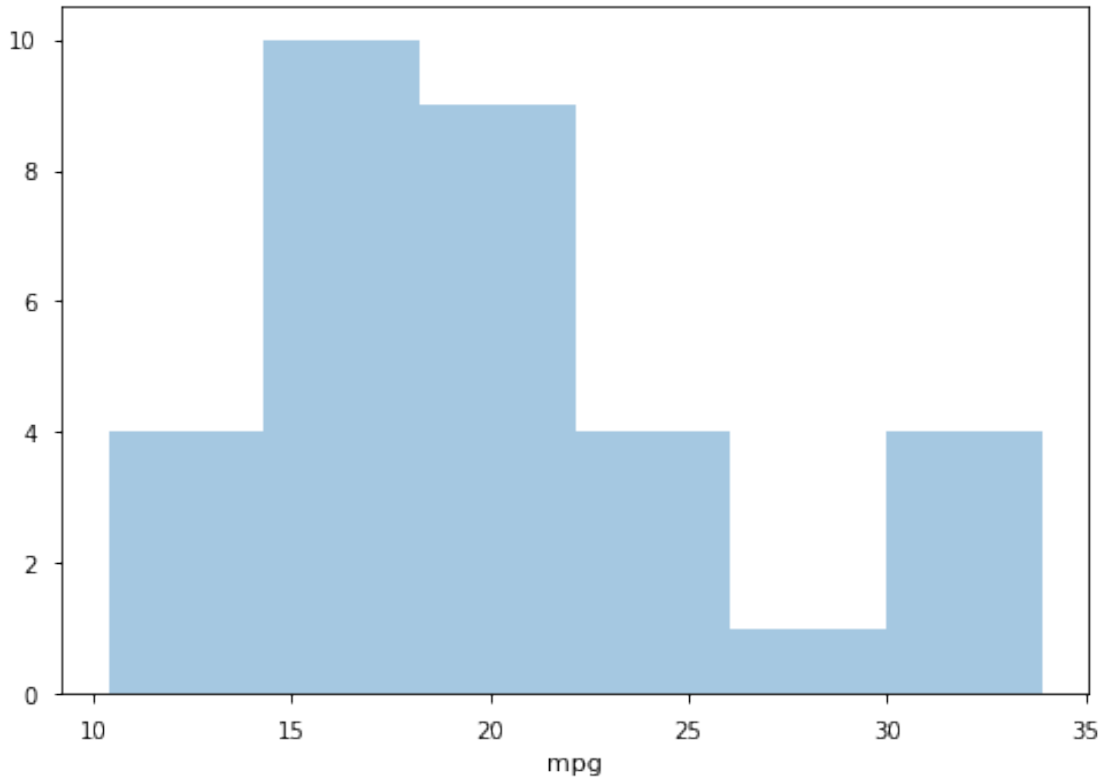
[11]:
```
mtcars['mpg'].plot( kind = 'density');
```



1.3.2 seaborn

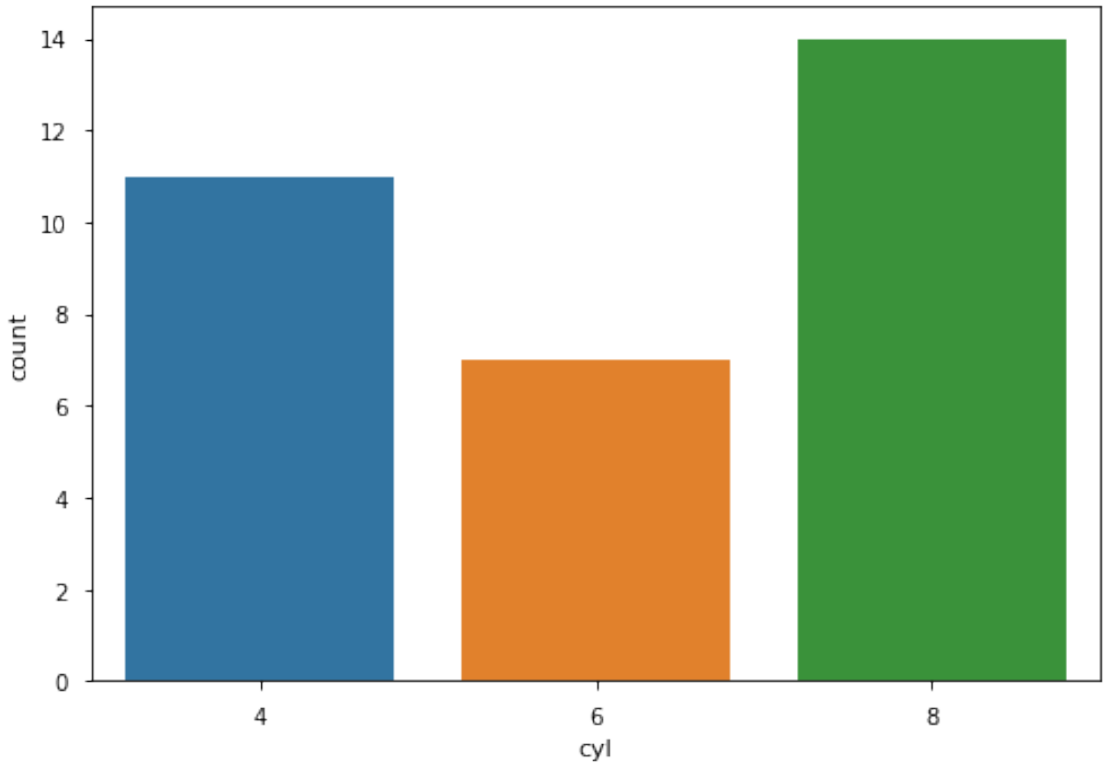
Histogram

[12]:
```
ax = sns.distplot(mtcars['mpg'], kde=False)
```



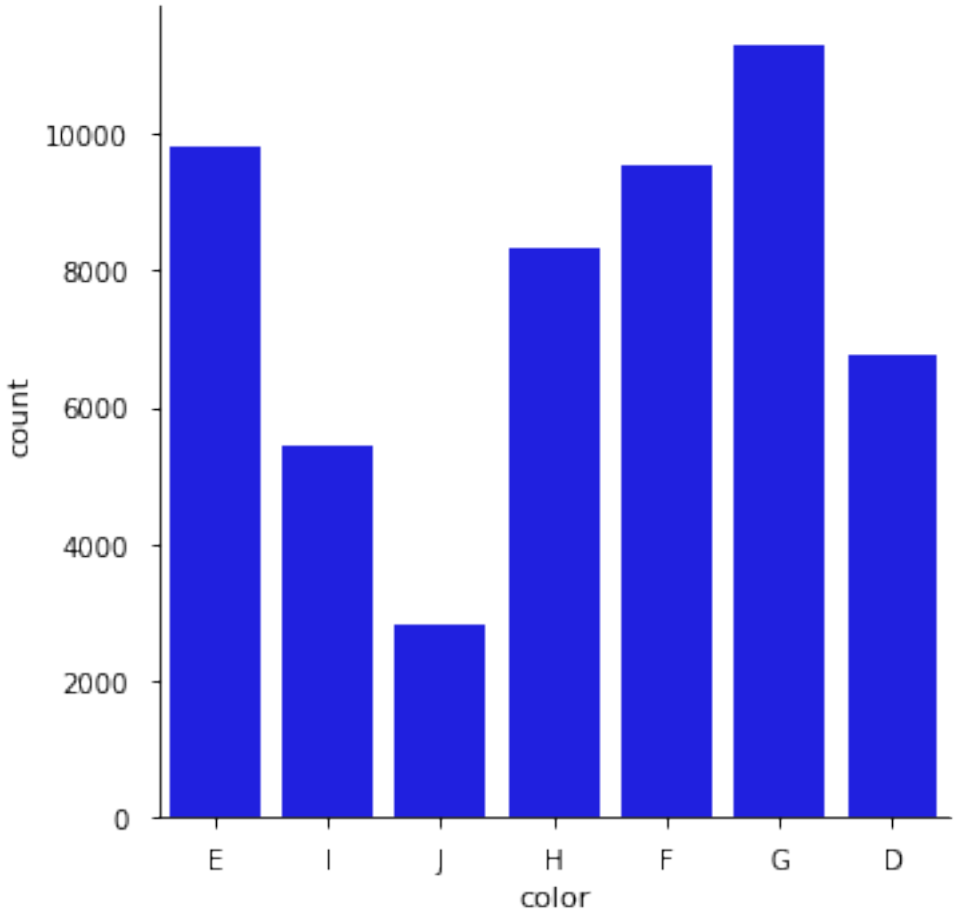
Bar plot

[46]:
```
sns.countplot(data = mtcars, x = 'cyl');
```



[50]:
```
diamonds = pd.read_csv('data/diamonds.csv.gz')
ordered_colors = ['E','F','G','H','I','J']
sns.catplot(data = diamonds, x = 'color', kind = 'count', color = 'blue')
```

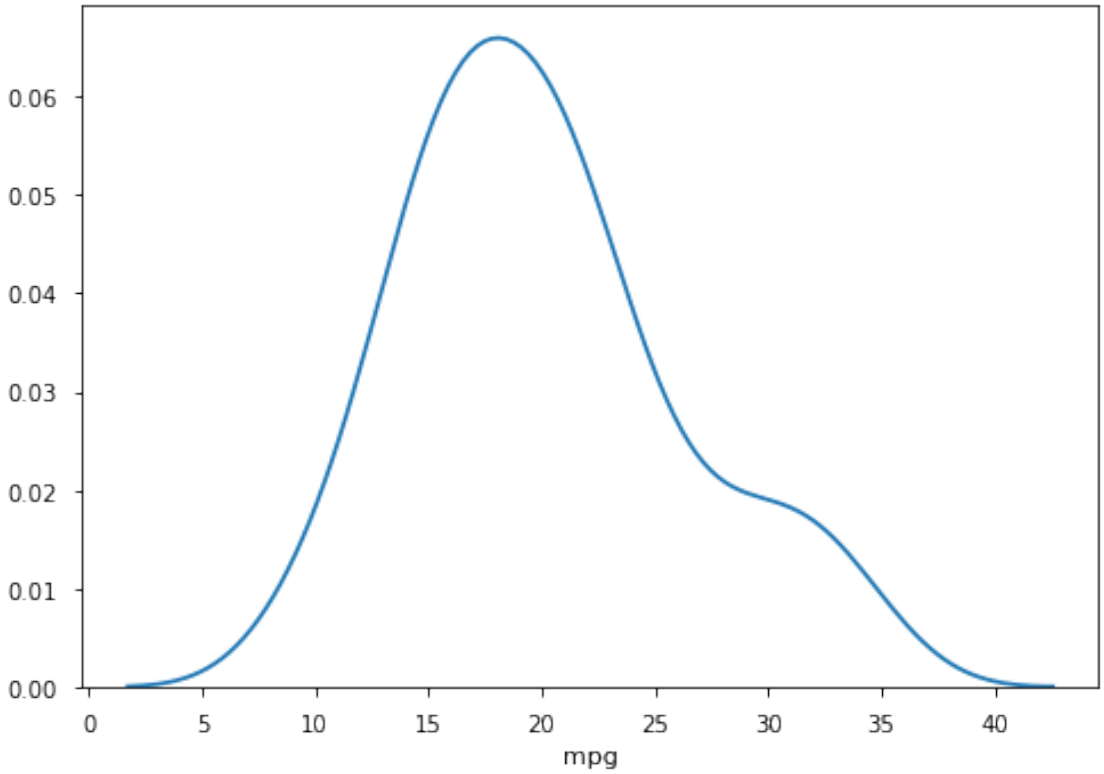
[50]: <seaborn.axisgrid.FacetGrid at 0x12ffbe550>



Density plot

[14]:
```
sns.distplot(mtcars['mpg'], hist=False)
```

[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x126320820>
```



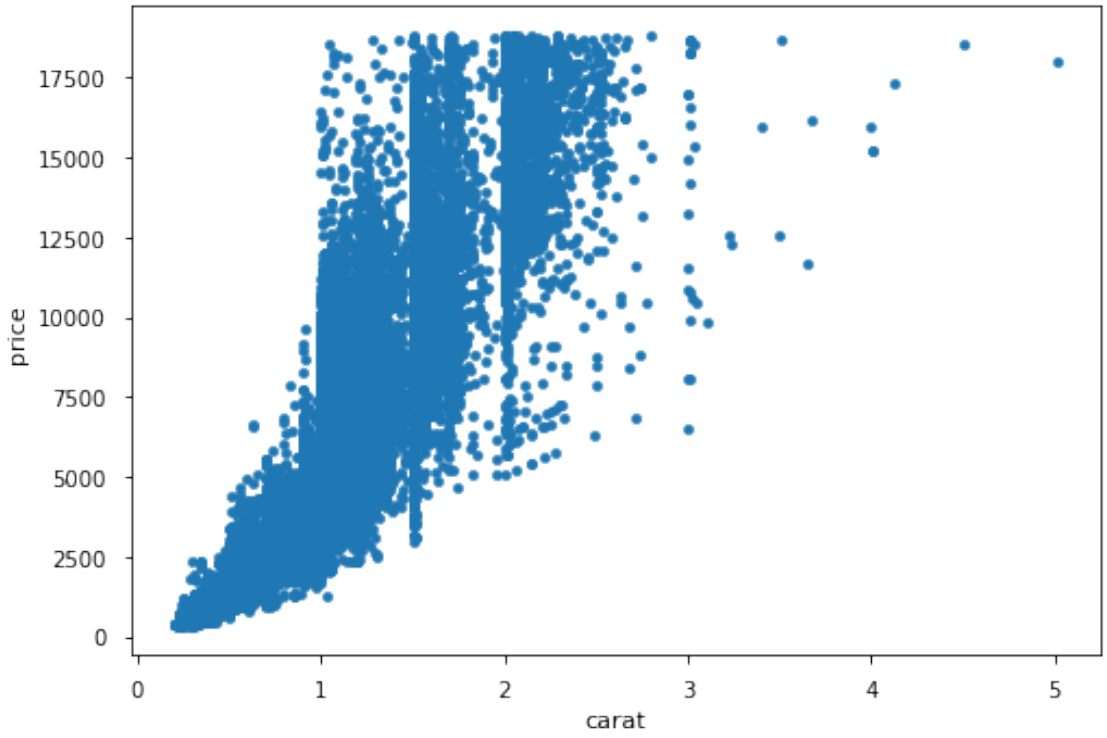
1.4 Bivariate plots

1.4.1 pandas

Scatter plot

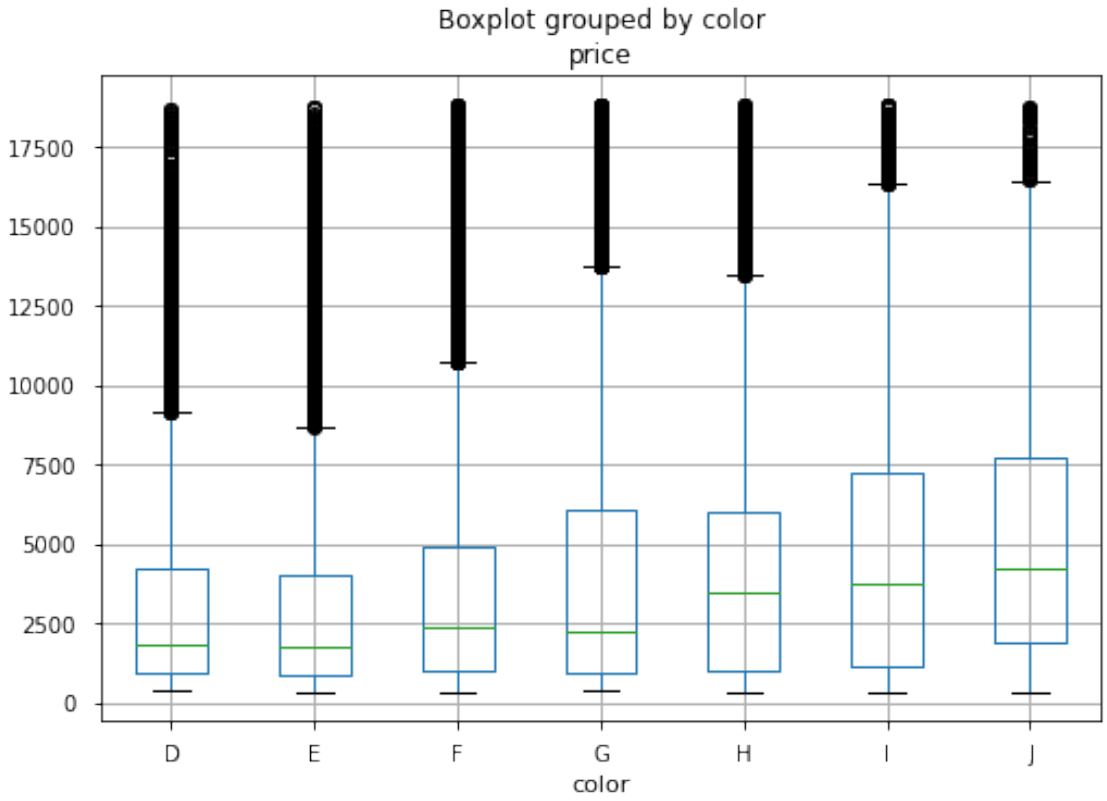
[15]:
```
diamonds = pd.read_csv('data/diamonds.csv.gz')
diamonds.plot(x = 'carat', y = 'price', kind = 'scatter')
```

[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x12642ac10>
```



Box plot

[16]:
```
diamonds.boxplot(column = 'price', by = 'color');
```

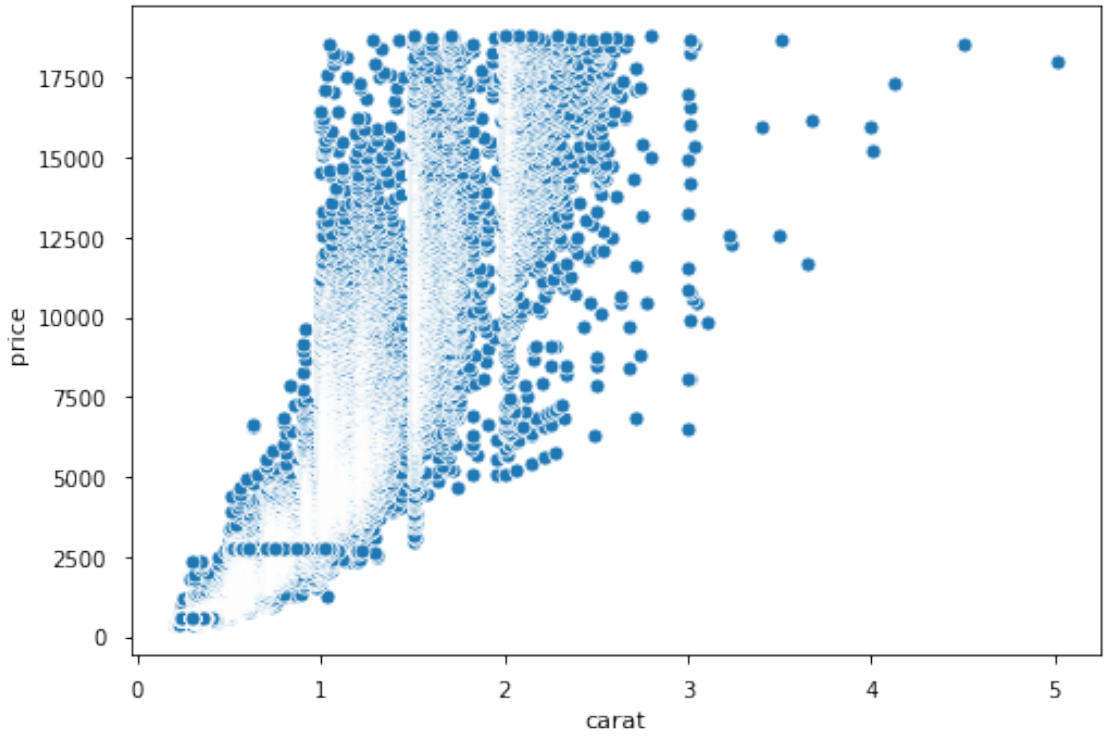


1.4.2 seaborn

Scatter plot

[17]:
```
sns.scatterplot(data = diamonds, x = 'carat', y = 'price')
```

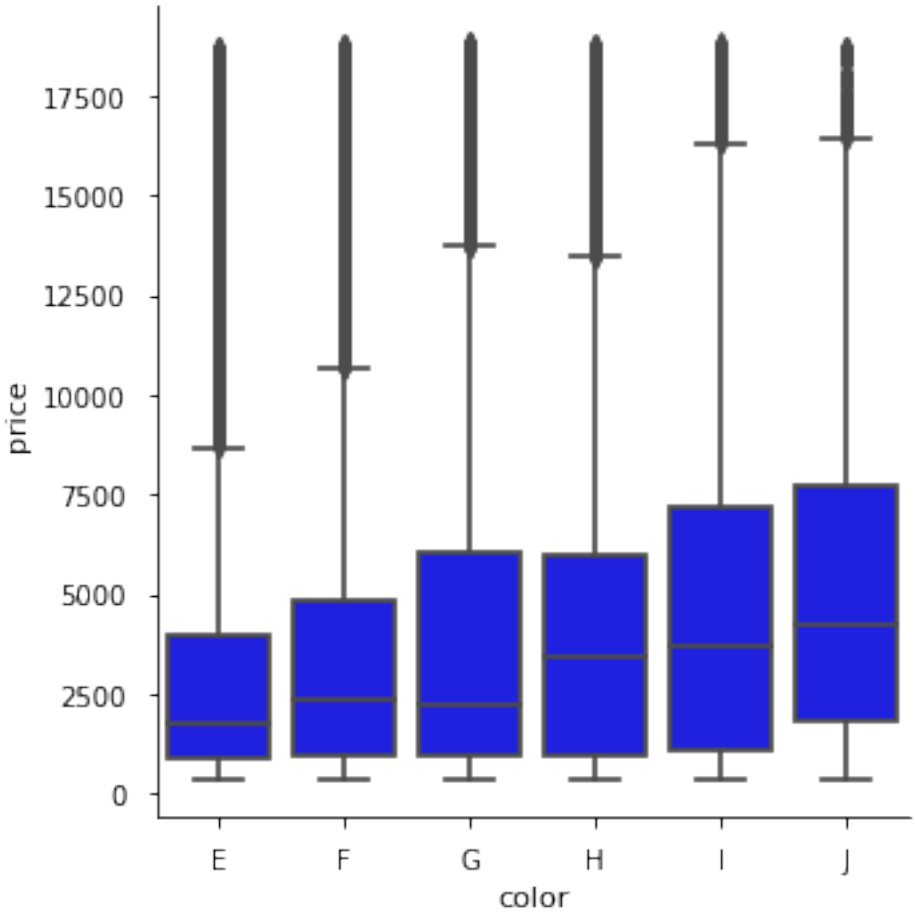
[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x127c25520>
```



Box plot

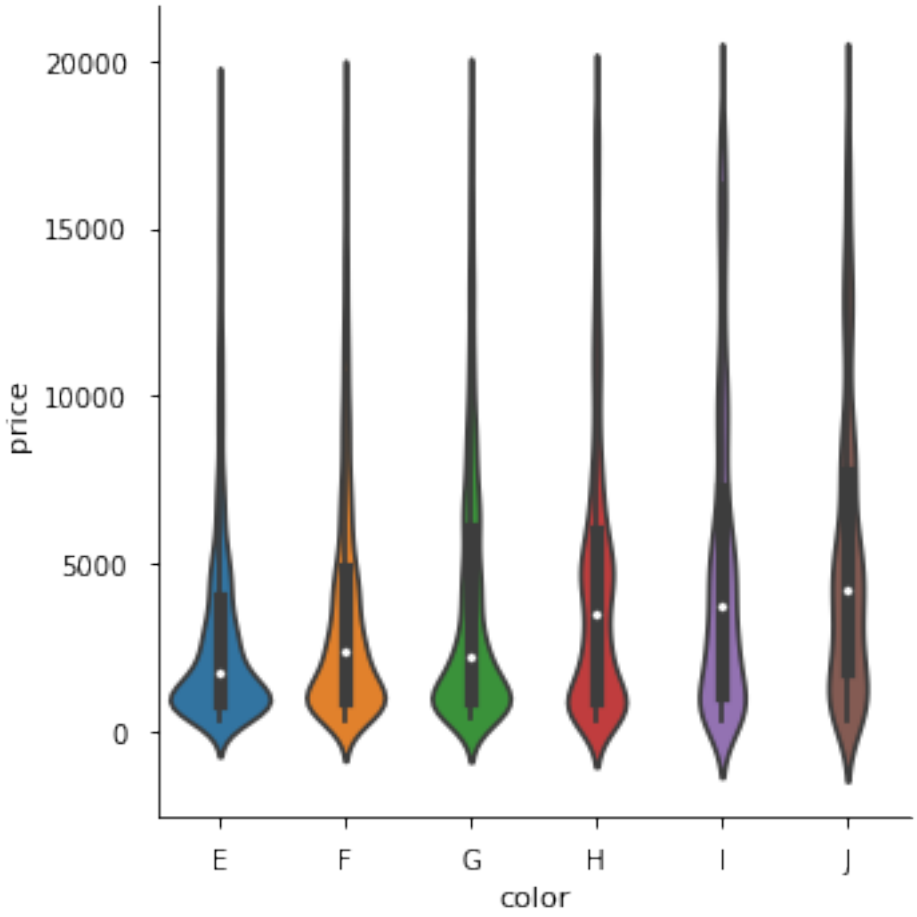
[57]:
```
ordered_color = ['E','F','G','H','I','J']
sns.catplot(data = diamonds, x = 'color', y = 'price', order = ordered_color,␣
 ↪color = 'blue', kind = 'box')
```

[57]: <seaborn.axisgrid.FacetGrid at 0x1302a9730>



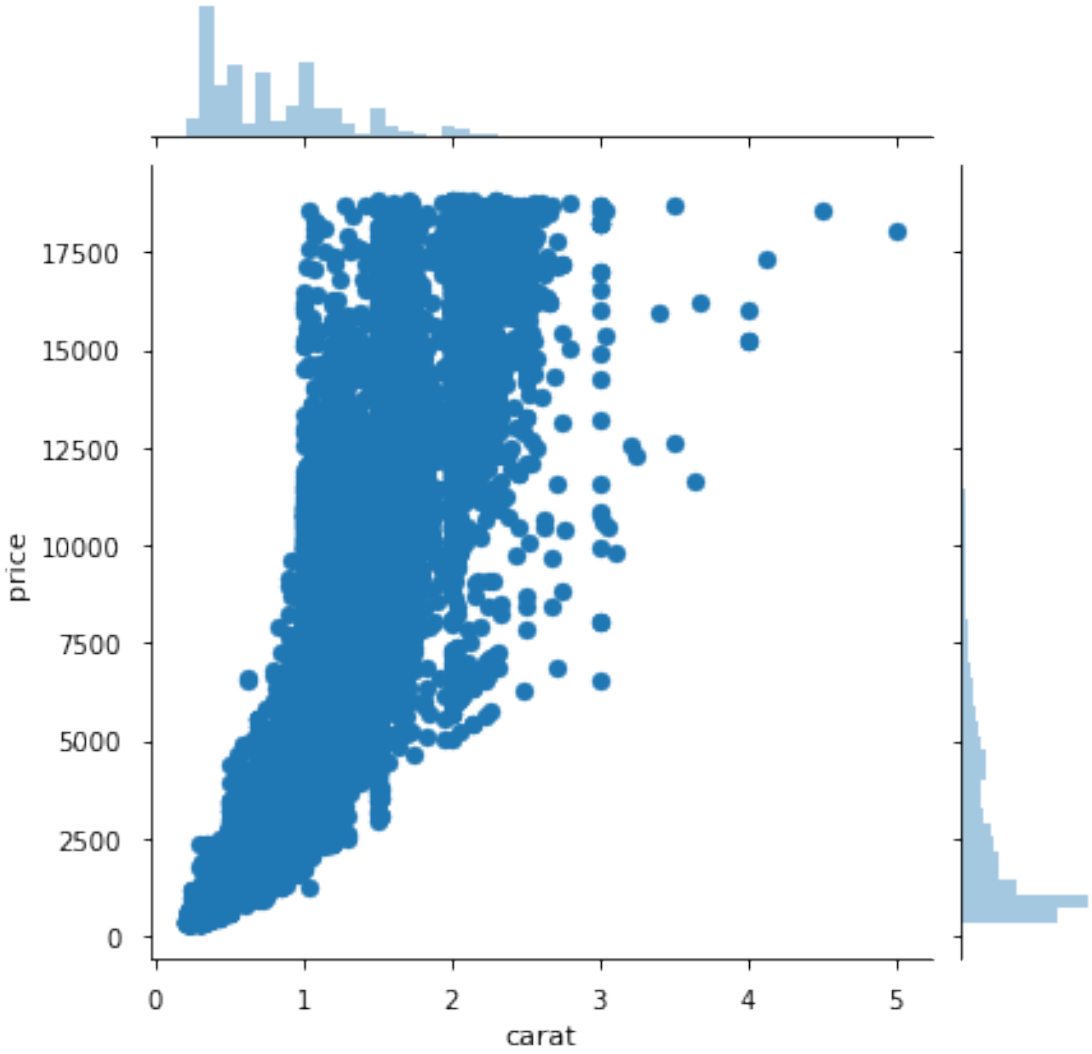
Violin plot

[59]:
```
g = sns.catplot(data = diamonds, x = 'color', y = 'price', kind = 'violin',␣
↪order = ordered_color);
```

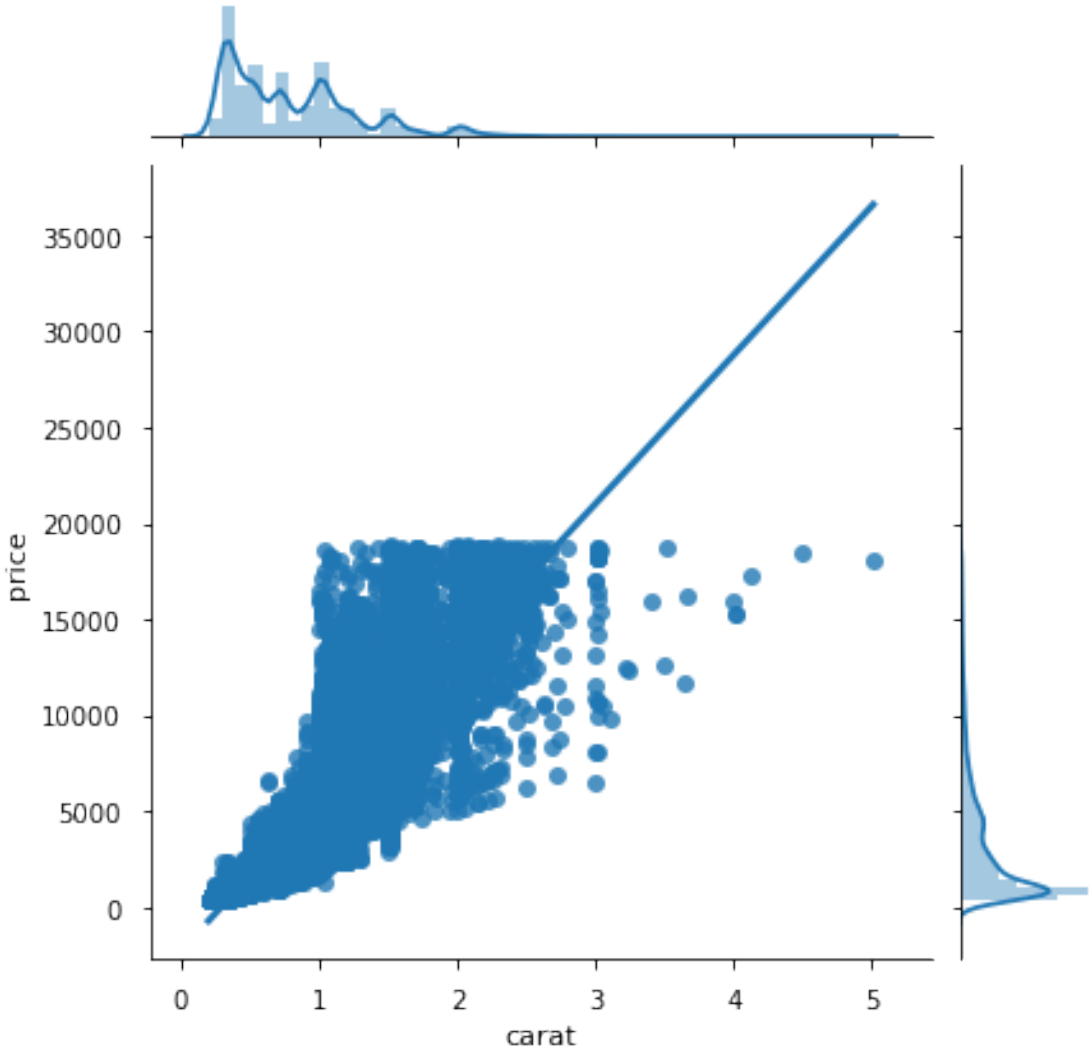


Joint plot

[20]:
```
sns.jointplot(data = diamonds, x = 'carat', y = 'price');
```

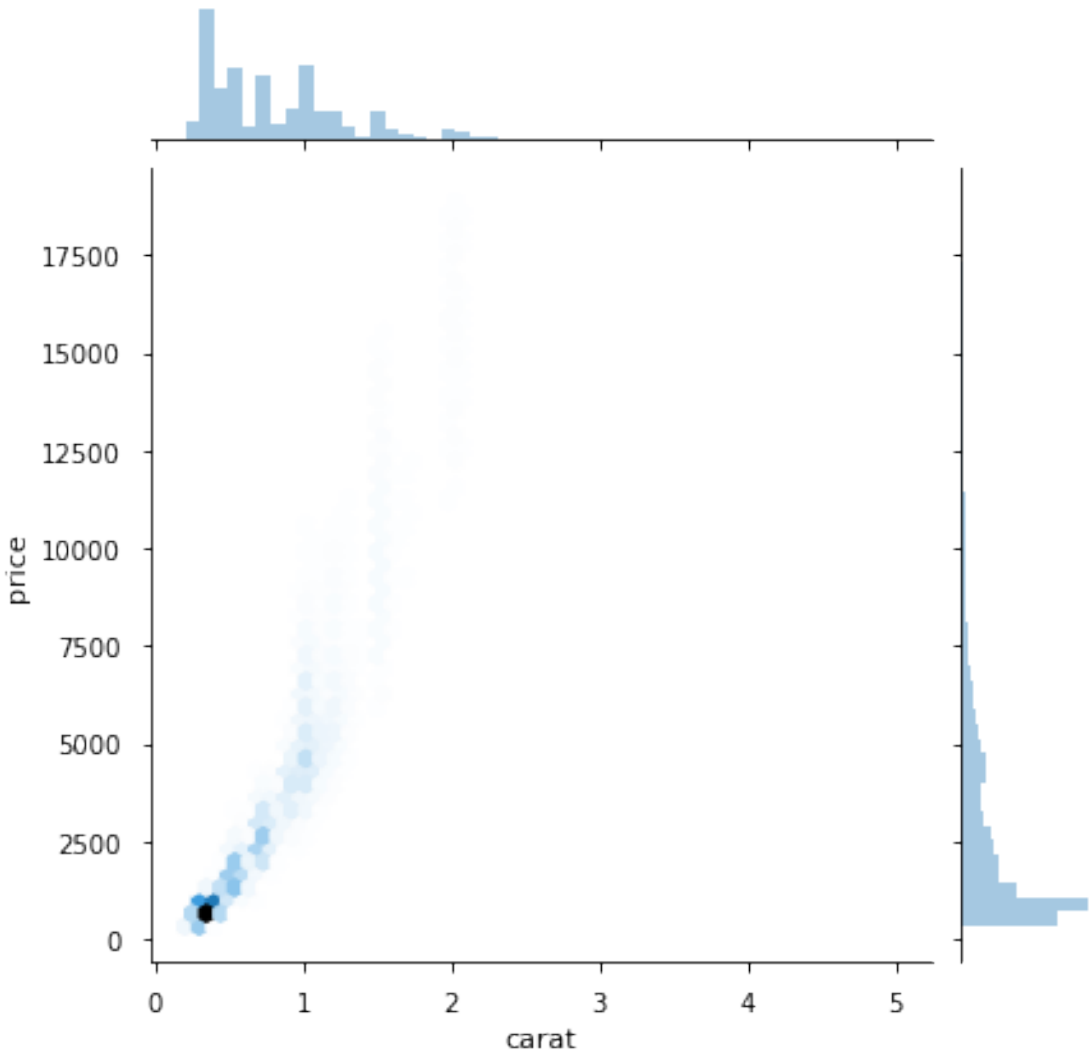


[21]:
```
sns.jointplot(data = diamonds, x = 'carat', y = 'price', kind = 'reg');
```



[22]:
```
sns.jointplot(data = diamonds, x = 'carat', y = 'price', kind = 'hex')
```

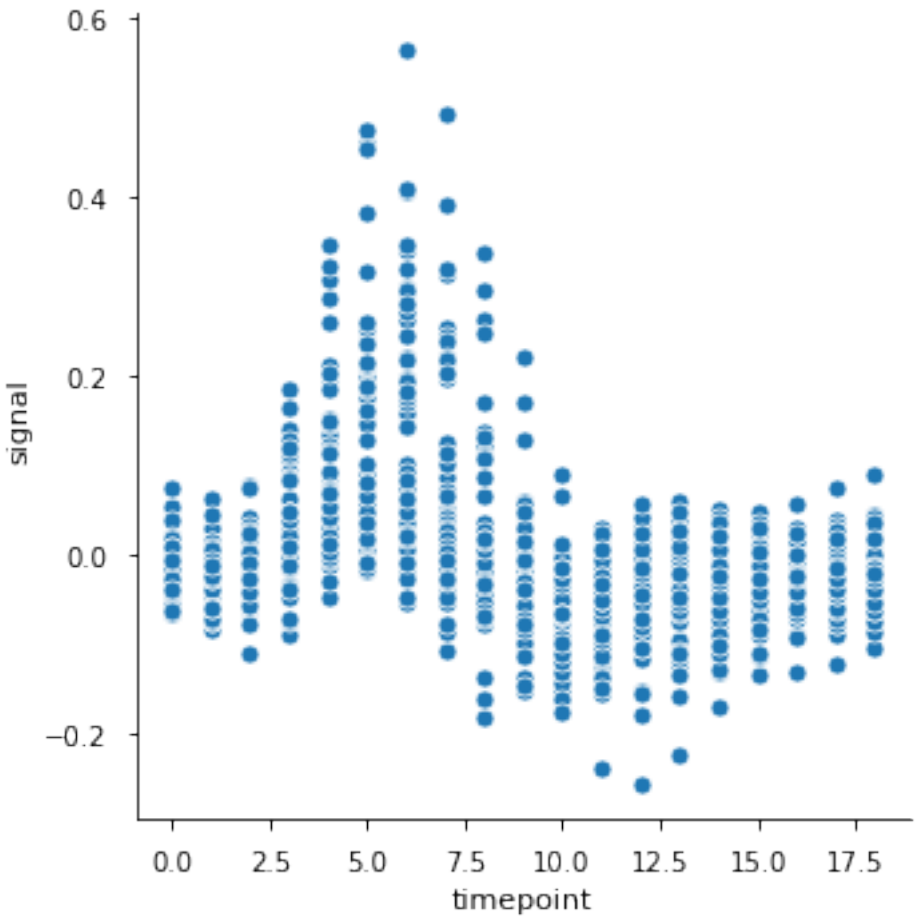
[22]:
```
<seaborn.axisgrid.JointGrid at 0x1281d85e0>
```



[23]:
```
fmri = sns.load_dataset('fmri')
```

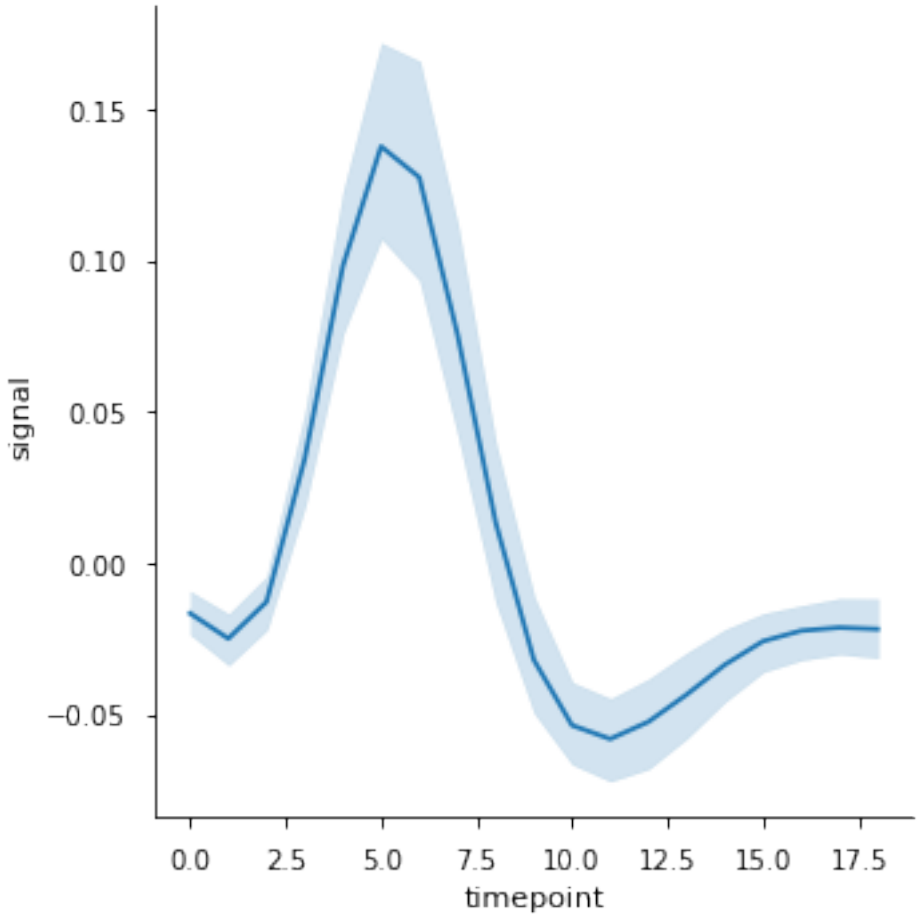
[25]:
```
plt.style.use('seaborn-notebook')
sns.relplot(x = 'timepoint', y = 'signal', data = fmri)
```

[25]:
```
<seaborn.axisgrid.FacetGrid at 0x126b3a580>
```



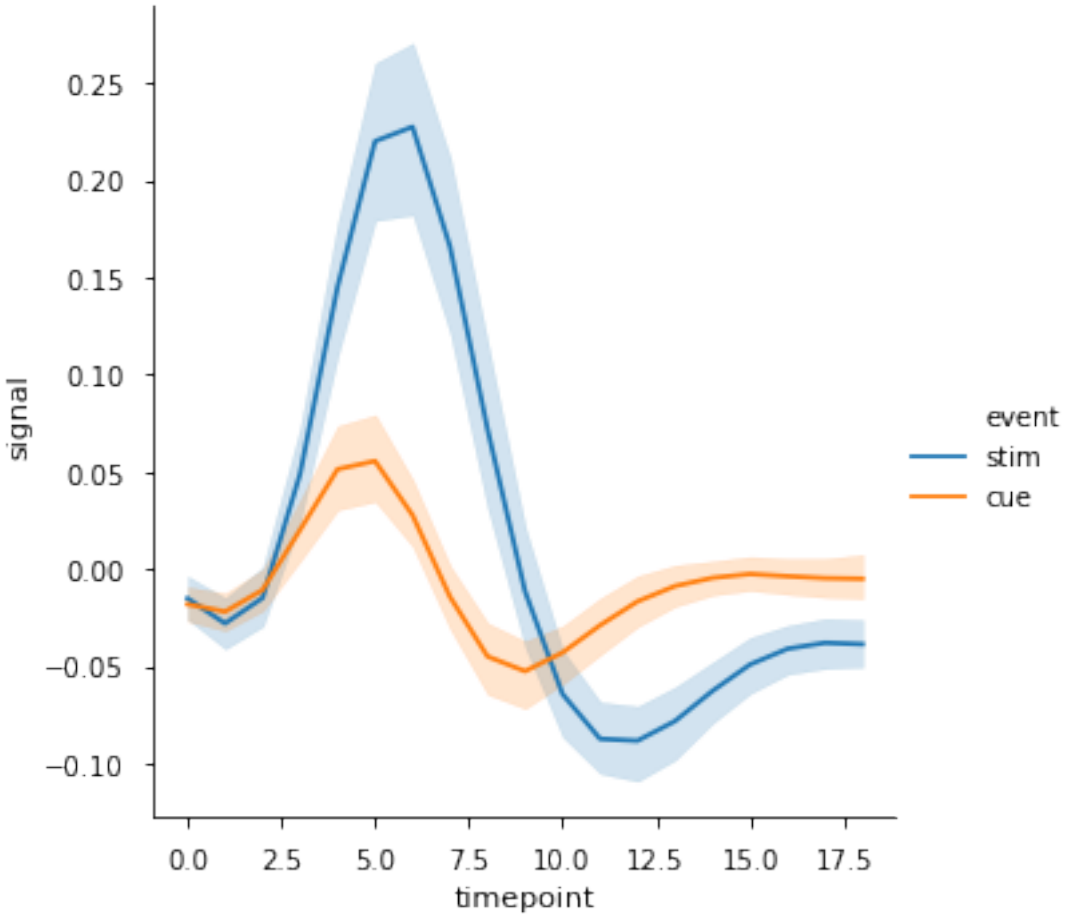
[26]:
```
sns.relplot(x = 'timepoint', y = 'signal', data = fmri, kind = 'line')
```

[26]: <seaborn.axisgrid.FacetGrid at 0x125d80b20>



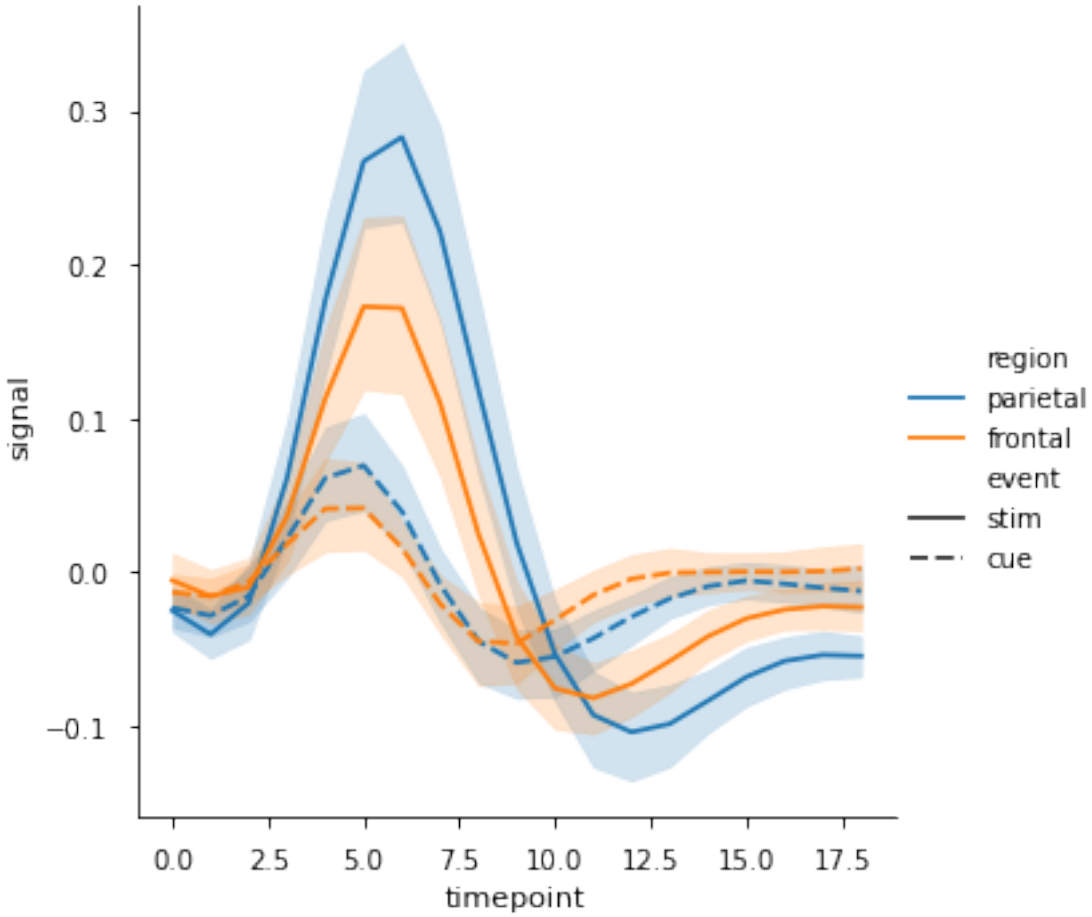
[27]:
```
sns.relplot(x = 'timepoint', y = 'signal', data = fmri, kind = 'line', hue␣
↪='event')
```

[27]: <seaborn.axisgrid.FacetGrid at 0x126b88490>



[28]:
```
sns.relplot(x = 'timepoint', y = 'signal', data = fmri, hue = 'region', style =␣
↪'event', kind = 'line')
```

[28]:
```
<seaborn.axisgrid.FacetGrid at 0x128120250>
```



[ ]:

1.5 Facets and multivariate data

[29]:
```
ts = pd.read_csv('data/ts.csv')
ts.head()
```

[29]:

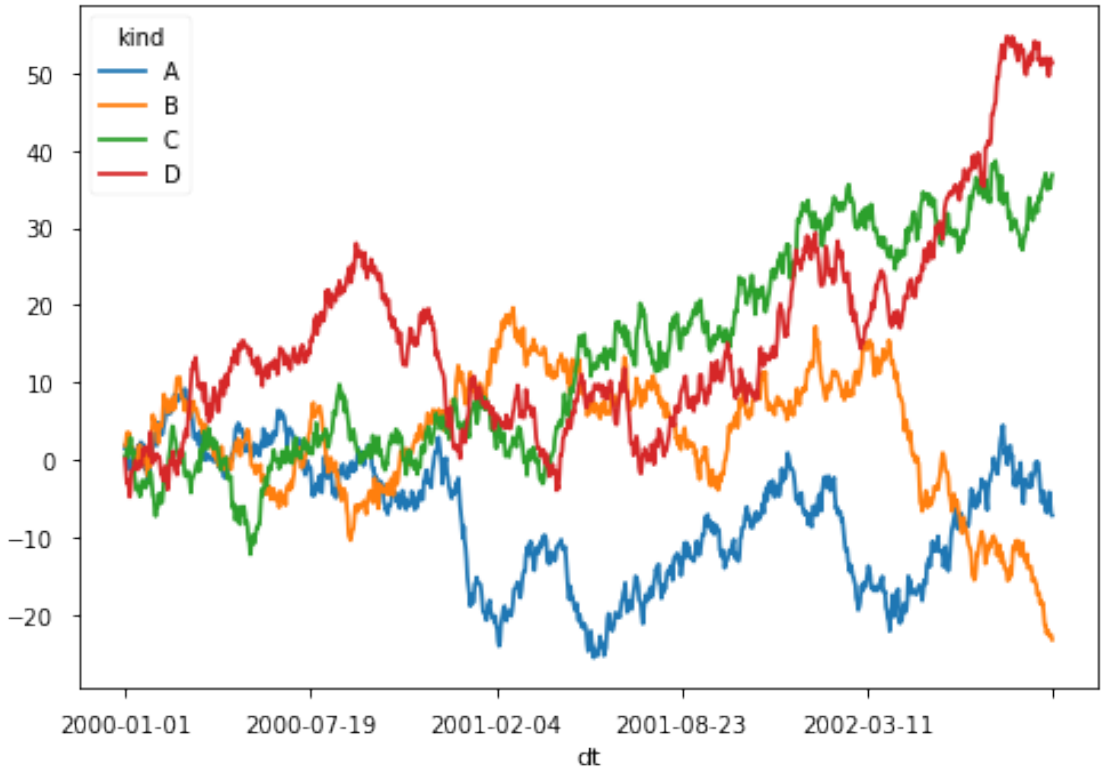
	dt	kind	value
0	2000-01-01	A	1.442521
1	2000-01-02	A	1.981290
2	2000-01-03	A	1.586494
3	2000-01-04	A	1.378969
4	2000-01-05	A	-0.277937

[30]:
```
dfp = ts.pivot(index = 'dt', columns = 'kind', values = 'value')
dfp.head()
```

[30]:
```
kind                A         B         C         D
dt
2000-01-01   1.442521  1.808741  0.437415  0.096980
2000-01-02   1.981290  2.277020  0.706127 -1.523108
2000-01-03   1.586494  3.474392  1.358063 -3.100735
2000-01-04   1.378969  2.906132  0.262223 -2.660599
2000-01-05  -0.277937  3.489553  0.796743 -3.417402
```

[31]:
```
fig, ax = plt.subplots()
dfp.plot(ax=ax)
```

[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x127819d90>
```

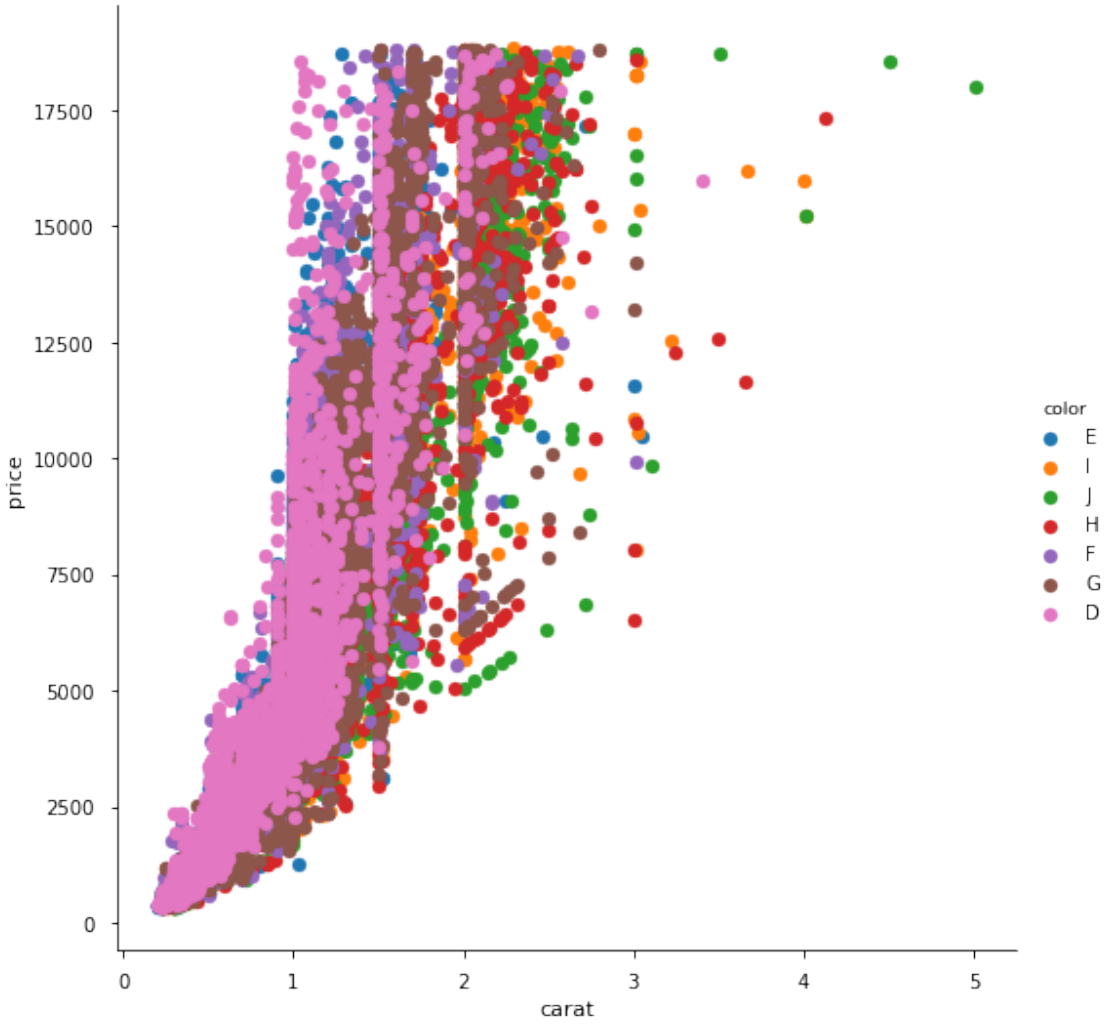


[32]:
```
g = sns.FacetGrid(ts, hue = 'kind', height = 5, aspect = 1.5)
g.map(plt.plot, 'dt', 'value').add_legend()
g.ax.set(xlabel = 'Date',
         ylabel = 'Value',
         title = 'Time series')
```

[32]:
```
[Text(25.37146990740741, 0.5, 'Value'),
 Text(0.5, 20.800000000000004, 'Date'),
 Text(0.5, 1.0, 'Time series')]
```

Scatter plots by group

[33]:
```
g = sns.FacetGrid(diamonds, hue = 'color', height = 7.5)
g.map(plt.scatter, 'carat', 'price').add_legend();
```



[34]:
```
clarity_ranking = ["I1", "SI2", "SI1", "VS2", "VS1", "VVS2", "VVS1", "IF"]
sns.scatterplot(x="carat", y="price",
                hue="clarity", size="depth",
                palette="ch:r=-.2,d=.3_r",
                hue_order=clarity_ranking,
                sizes=(1, 8), linewidth=0,
                data=diamonds, ax=ax)
```

[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x127819d90>
```

Facets

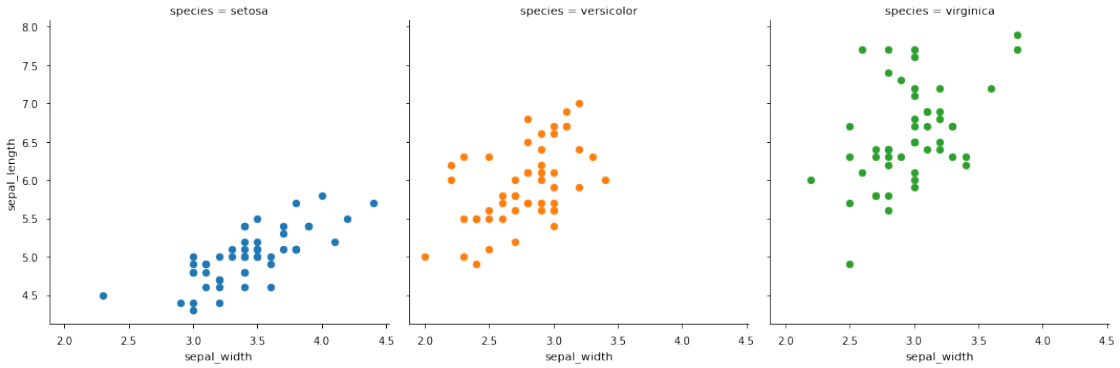
[35]:
```
iris = pd.read_csv('data/iris.csv')
iris.head()
```

[35]:

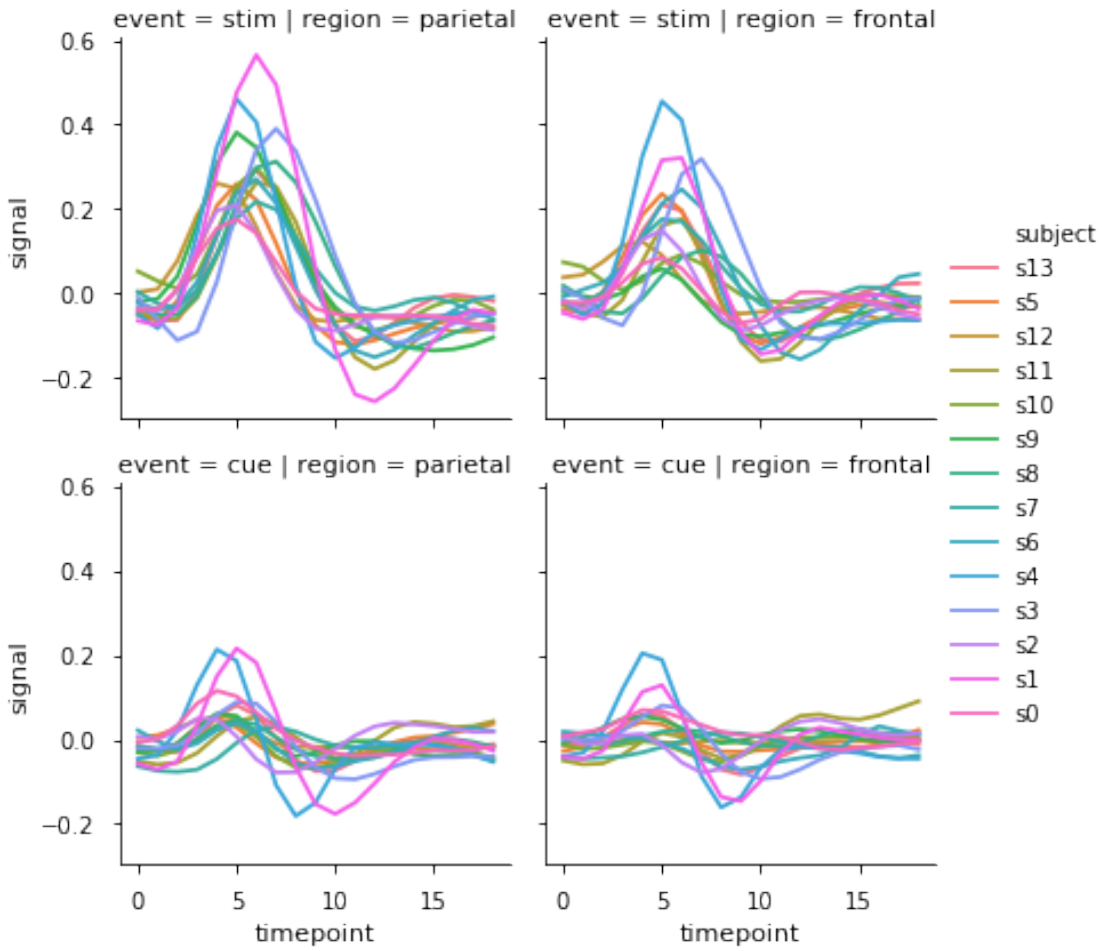
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

[36]:
```
g = sns.FacetGrid(iris, col = 'species', hue = 'species', height = 5)
g.map(plt.scatter, 'sepal_width', 'sepal_length')
```

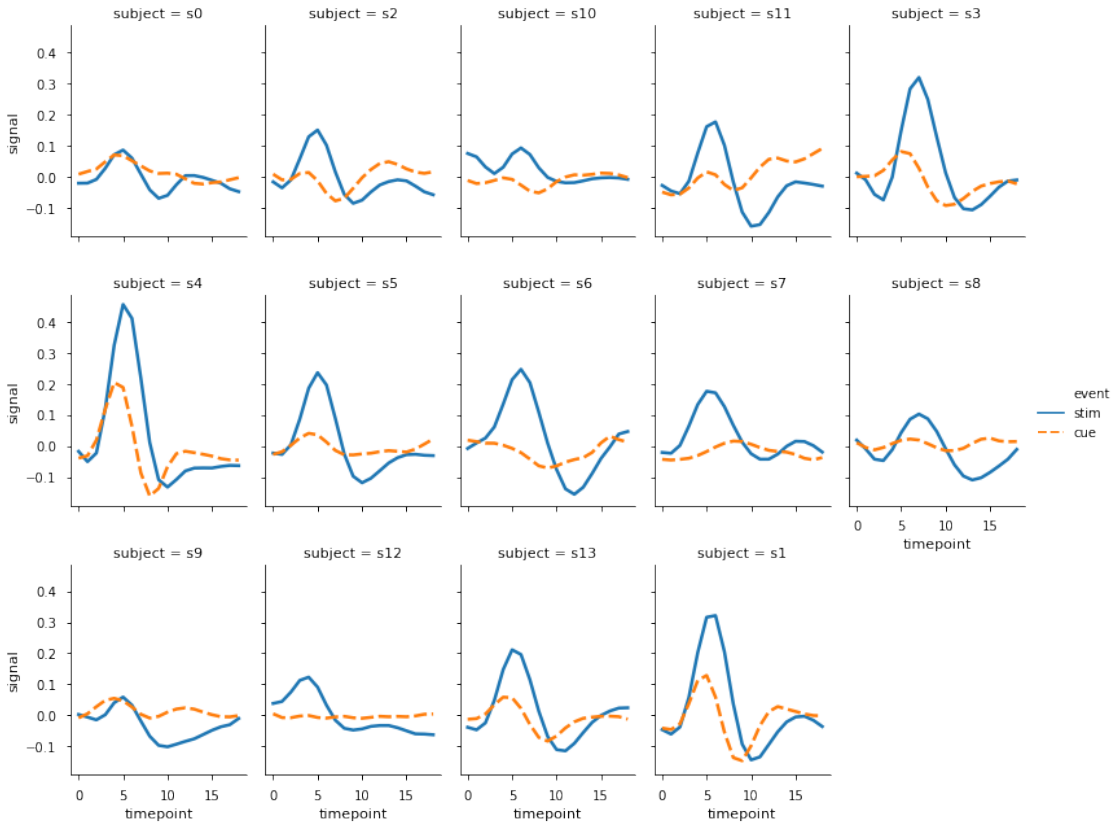
[36]: <seaborn.axisgrid.FacetGrid at 0x129825940>



[38]:
```
sns.relplot(x="timepoint", y="signal", hue="subject",
            col="region", row="event", height=3,
            kind="line", estimator=None, data=fmri);
```

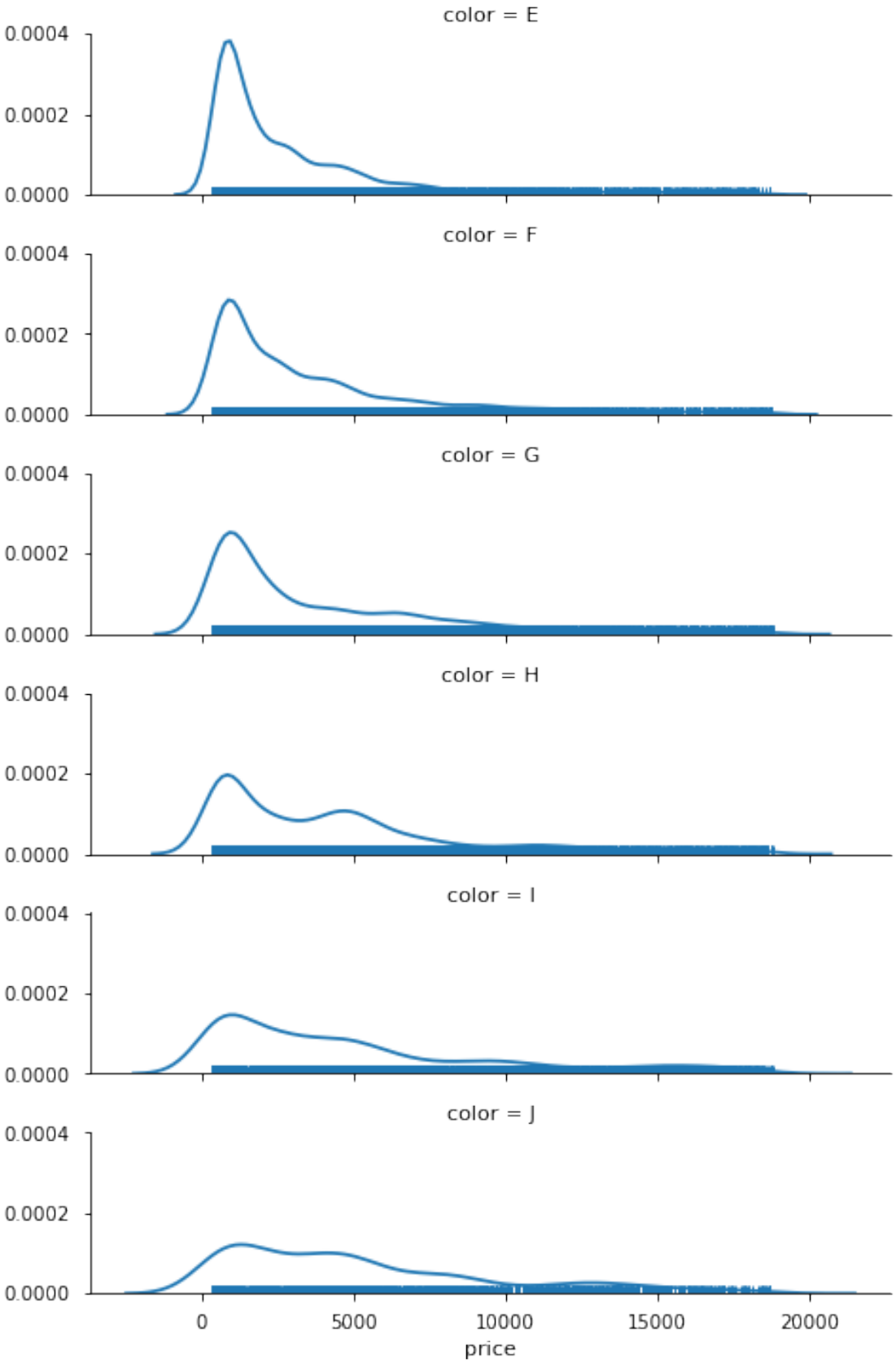


[40]:
```
sns.relplot(x="timepoint", y="signal", hue="event", style="event",
            col="subject", col_wrap=5,
            height=3, aspect=.75, linewidth=2.5,
            kind="line", data=fmri.query("region == 'frontal'"));
```



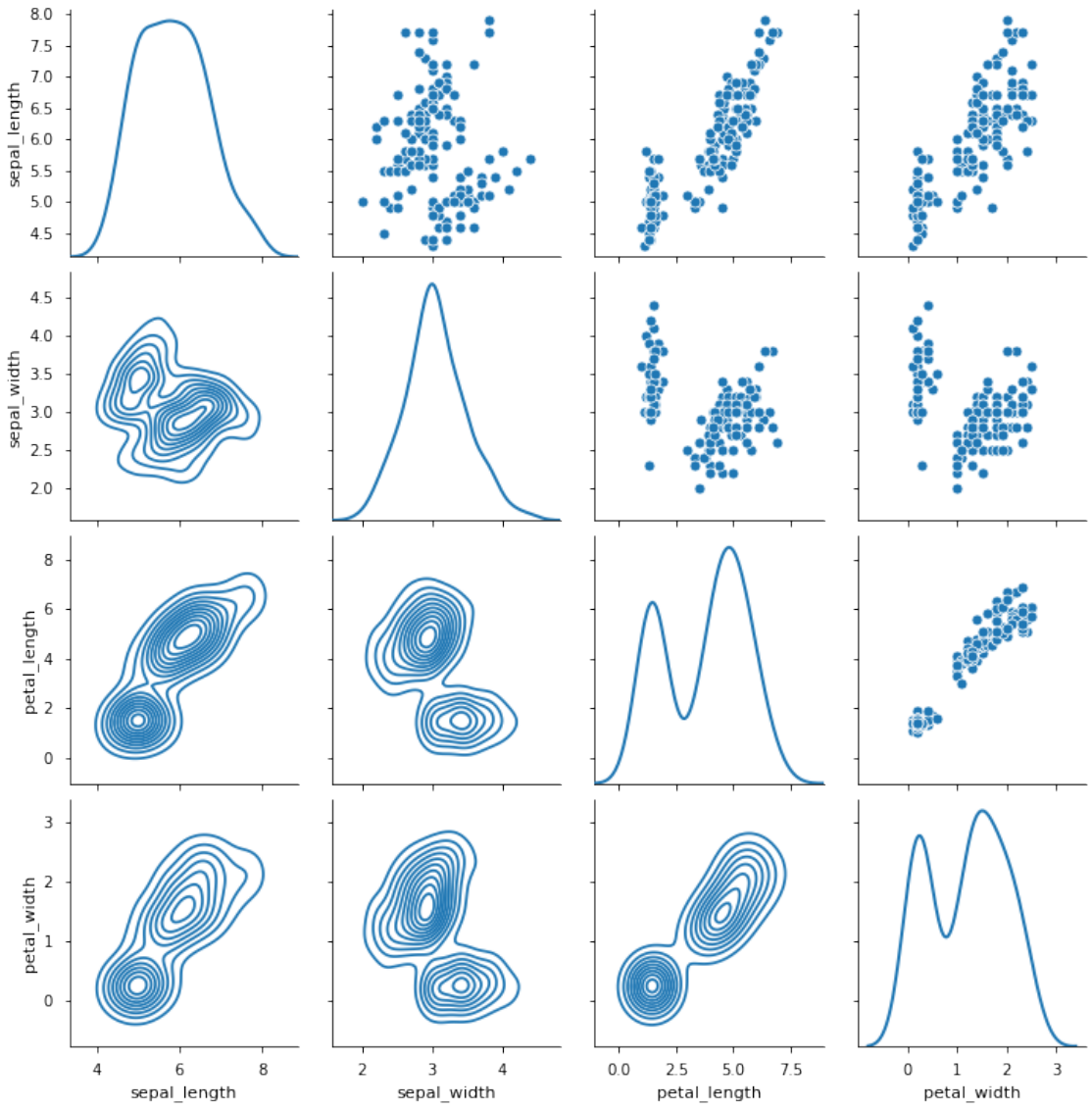
[44]:
```
ordered_colors = ['E','F','G','H','I','J']
g = sns.FacetGrid(data = diamonds, row = 'color', height = 1.7, aspect = 4,␣
 ↪row_order = ordered_colors)
g.map(sns.distplot, 'price', hist = False, rug = True)
```

[44]: <seaborn.axisgrid.FacetGrid at 0x12d82fca0>


color = E
color = F
color = G
color = H
color = I
color = J
0.0004
0.0002
0.0000
0
5000
10000
15000
20000
price

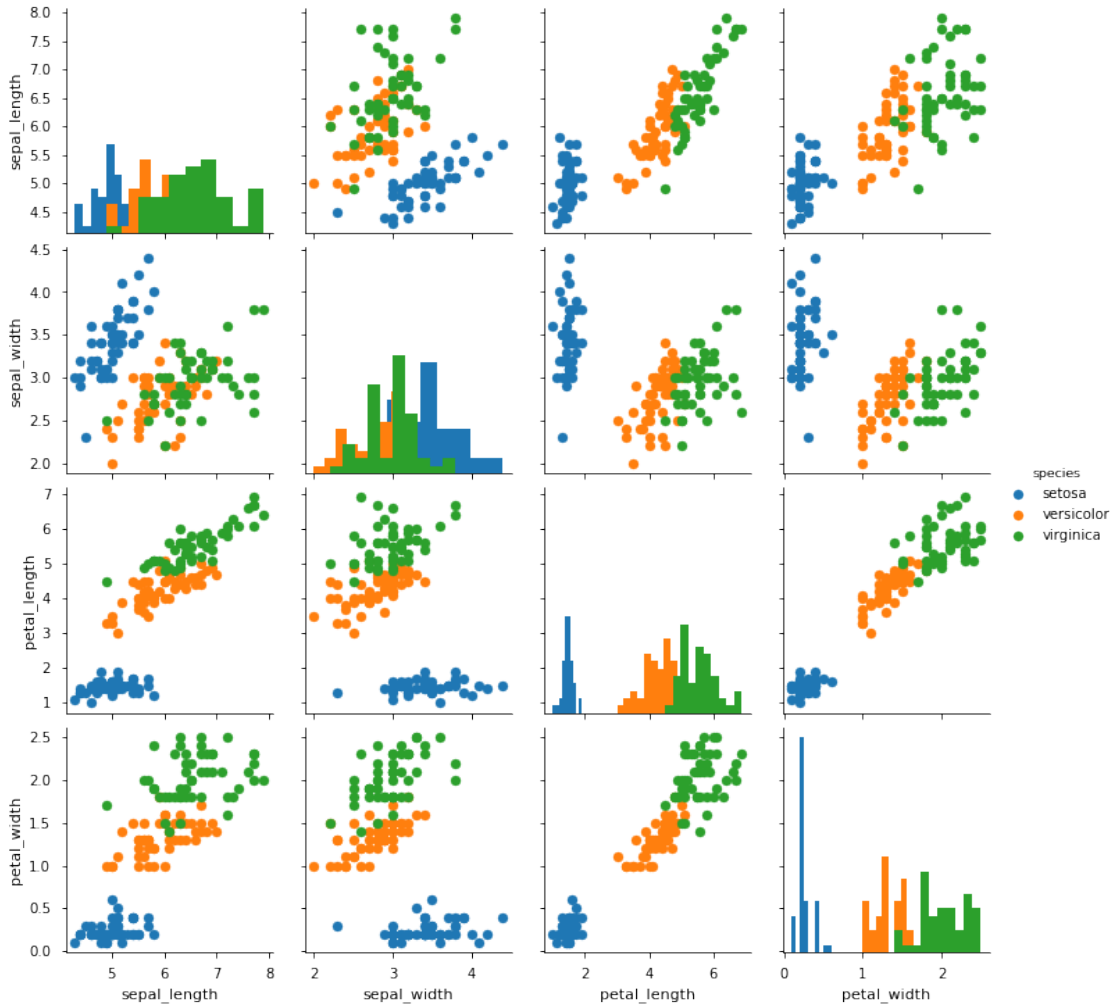
[39]:
```
g = sns.PairGrid(iris, diag_sharey=False)
g.map_upper(sns.scatterplot)
g.map_lower(sns.kdeplot, colors="C0")
g.map_diag(sns.kdeplot, lw=2)
```

[39]: <seaborn.axisgrid.PairGrid at 0x129855ca0>



[45]:
```
g = sns.PairGrid(iris, hue="species")
g.map_diag(plt.hist)
g.map_offdiag(plt.scatter)
```

```
g.add_legend();
```



1.6 Customizing the look

1.6.1 Themes

[ ]:
```
plt.style.available
```

[ ]:
```
plt.style.use('fivethirtyeight')
sns.scatterplot(data = iris, x = 'sepal_width', y = 'sepal_length')
```

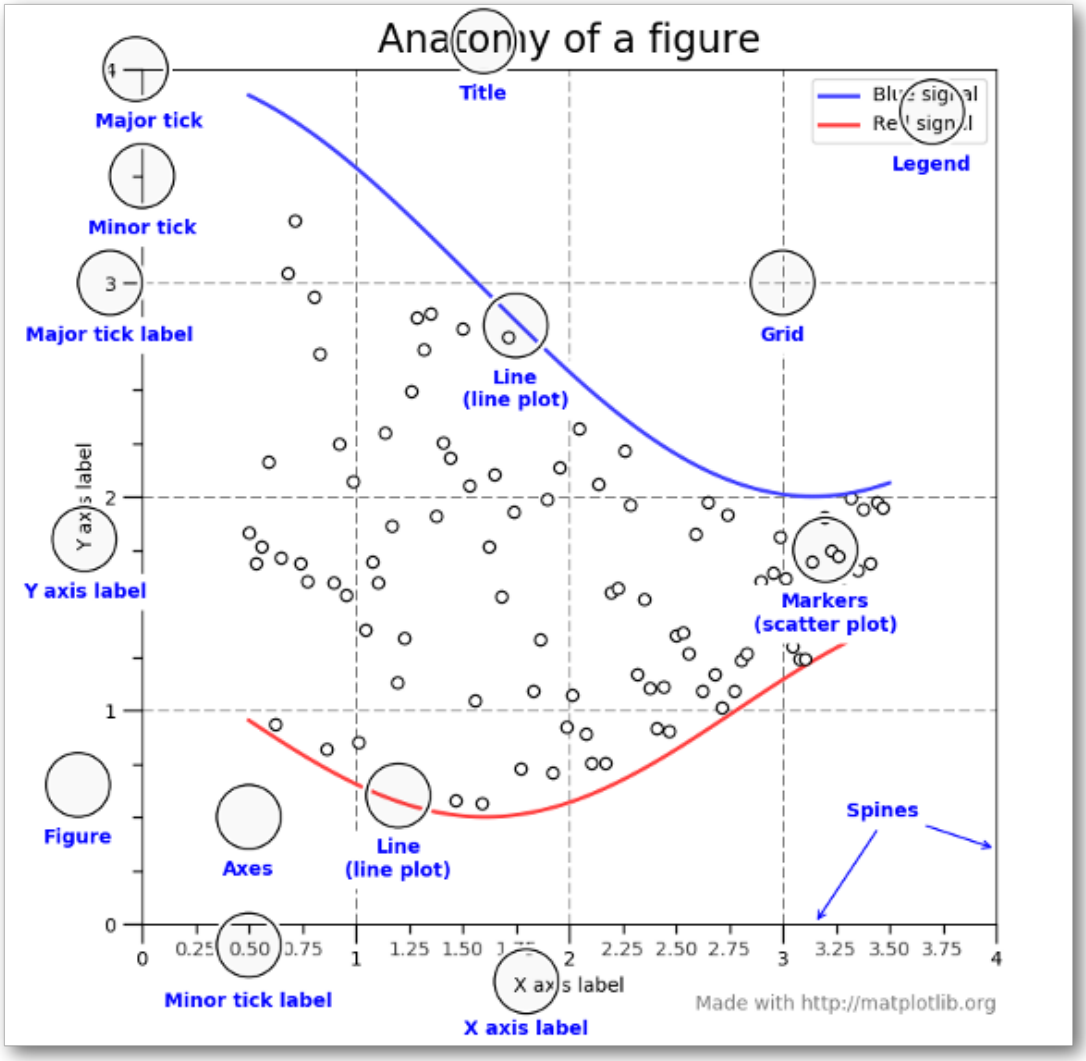
[ ]:
```
plt.style.use('bmh')
sns.scatterplot(data = iris, x = 'sepal_width', y = 'sepal_length')
```

[ ]:
```
plt.style.use('classic')
sns.scatterplot(data = iris, x = 'sepal_width', y = 'sepal_length')
```

```
[ ]: plt.style.use('ggplot')
     sns.scatterplot(data = iris, x = 'sepal_width', y = 'sepal_length')
```

```
[ ]: plt.style.use('Solarize_Light2')
     sns.scatterplot(data = iris, x = 'sepal_width', y = 'sepal_length')
```

1.7 Finer control with matplotlib

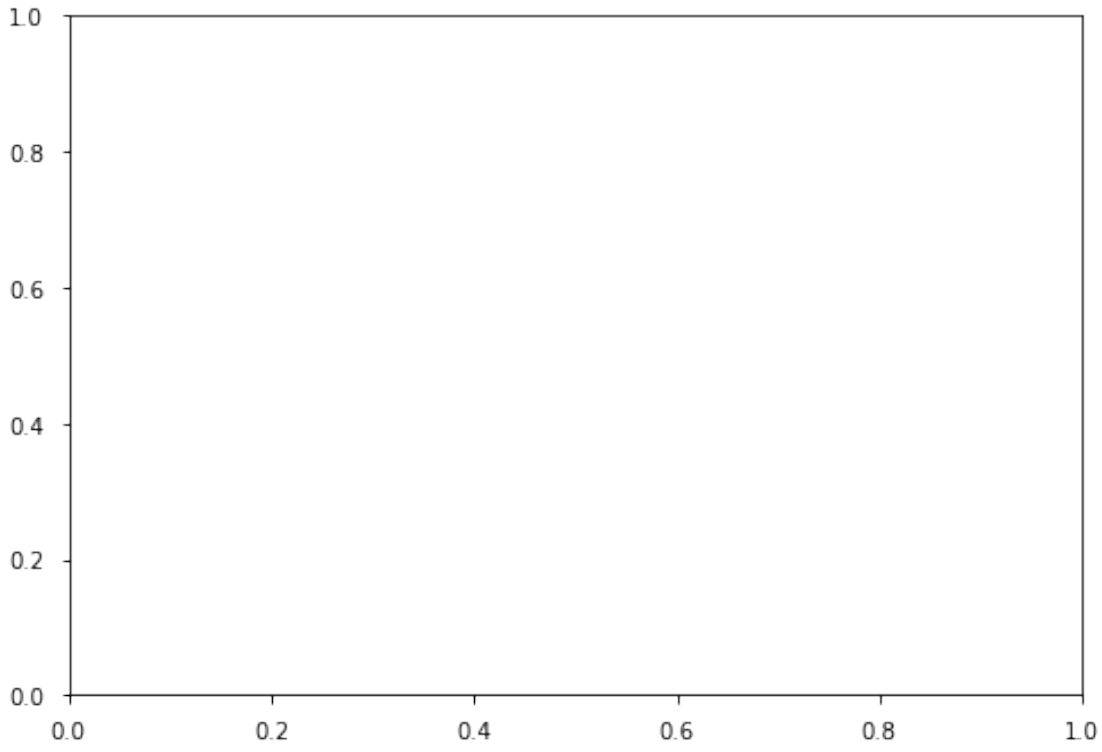


A pathway to learning (Chris Moffit)

1. Learn the basic matplotlib terminology, specifically what is a `Figure` and an `Axes` .
2. Always use the object-oriented interface. Get in the habit of using it from the start of your analysis.
3. Start your visualizations with basic pandas plotting.
4. Use seaborn for the more complex statistical visualizations.
5. Use matplotlib to customize the pandas or seaborn visualization.

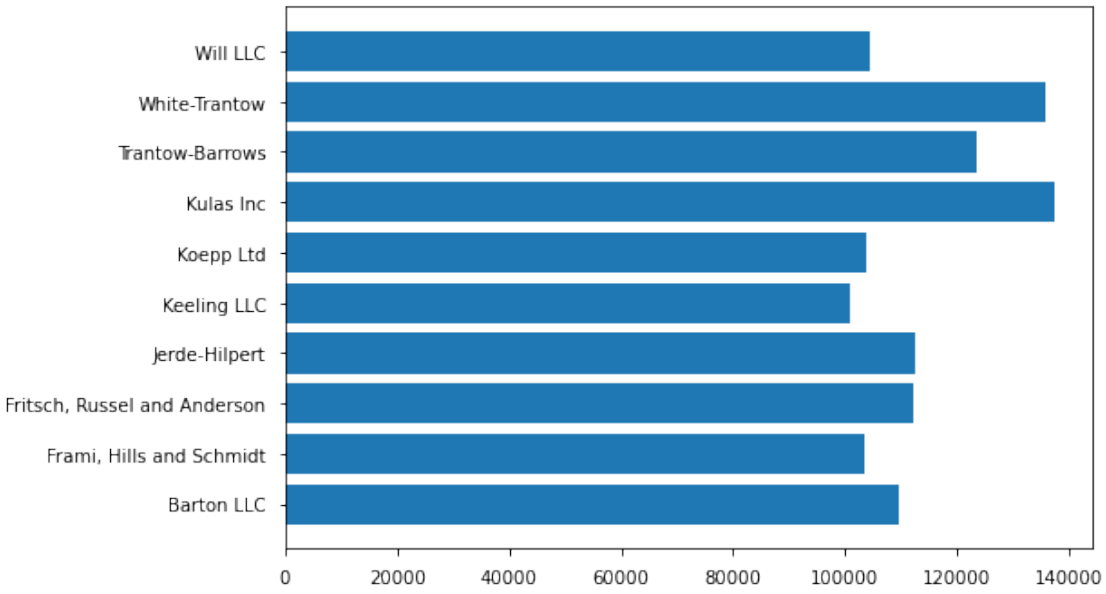
[60]:
```
from matplotlib.ticker import FuncFormatter

data = {'Barton LLC': 109438.50,
        'Frami, Hills and Schmidt': 103569.59,
        'Fritsch, Russel and Anderson': 112214.71,
        'Jerde-Hilpert': 112591.43,
        'Keeling LLC': 100934.30,
        'Koepp Ltd': 103660.54,
        'Kulas Inc': 137351.96,
        'Trantow-Barrows': 123381.38,
        'White-Trantow': 135841.99,
        'Will LLC': 104437.60}
group_data = list(data.values())
group_names = list(data.keys())
group_mean = np.mean(group_data)
```

[61]:
```
fig, ax = plt.subplots()
```

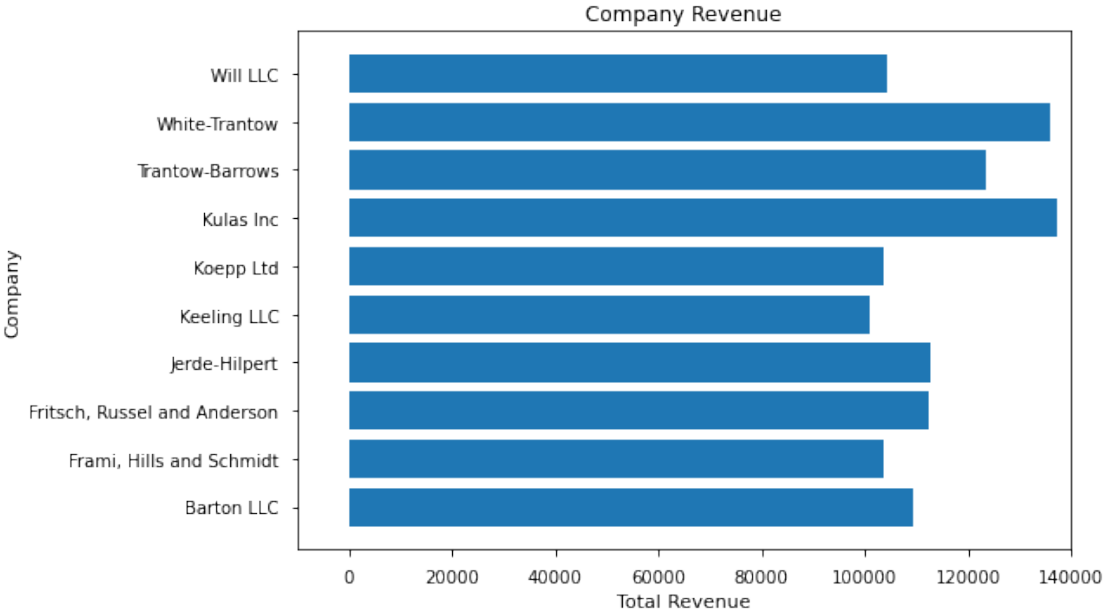


[63]:
```
fig, ax = plt.subplots()
ax.barh(group_names, group_data)
```

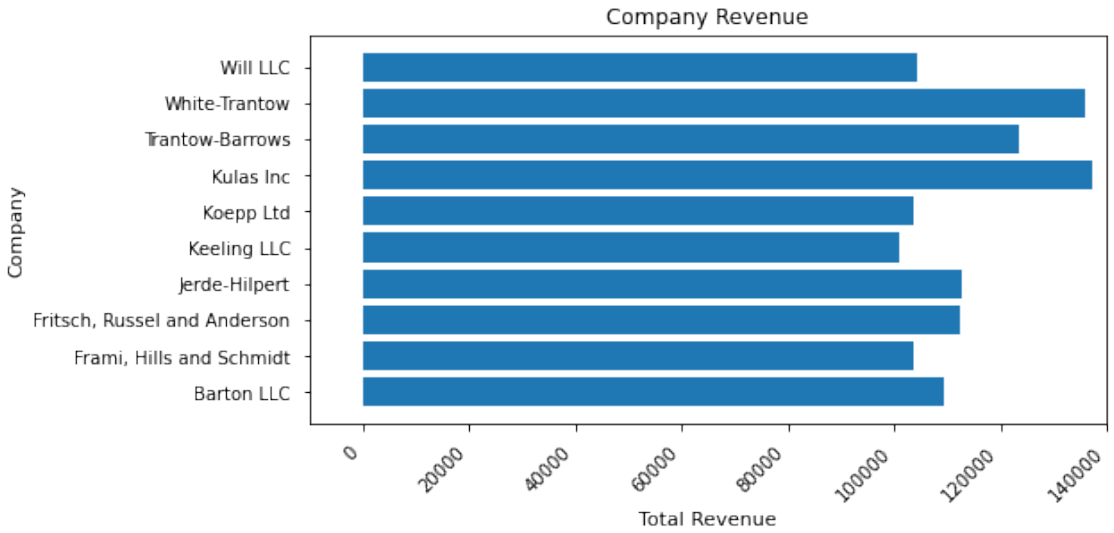
[63]:
```
<BarContainer object of 10 artists>
```



[66]:
```
fig, ax = plt.subplots()
ax.barh(group_names, group_data)
ax.set(xlim = [-10000, 140000], xlabel = 'Total Revenue', ylabel = 'Company',␣
↪title = 'Company Revenue');
```



[69]:
```
fig, ax = plt.subplots(figsize=(8, 4))
ax.barh(group_names, group_data)
labels = ax.get_xticklabels()
plt.setp(labels, rotation=45, horizontalalignment='right')
ax.set(xlim=[-10000, 140000], xlabel='Total Revenue', ylabel='Company',
       title='Company Revenue');
```



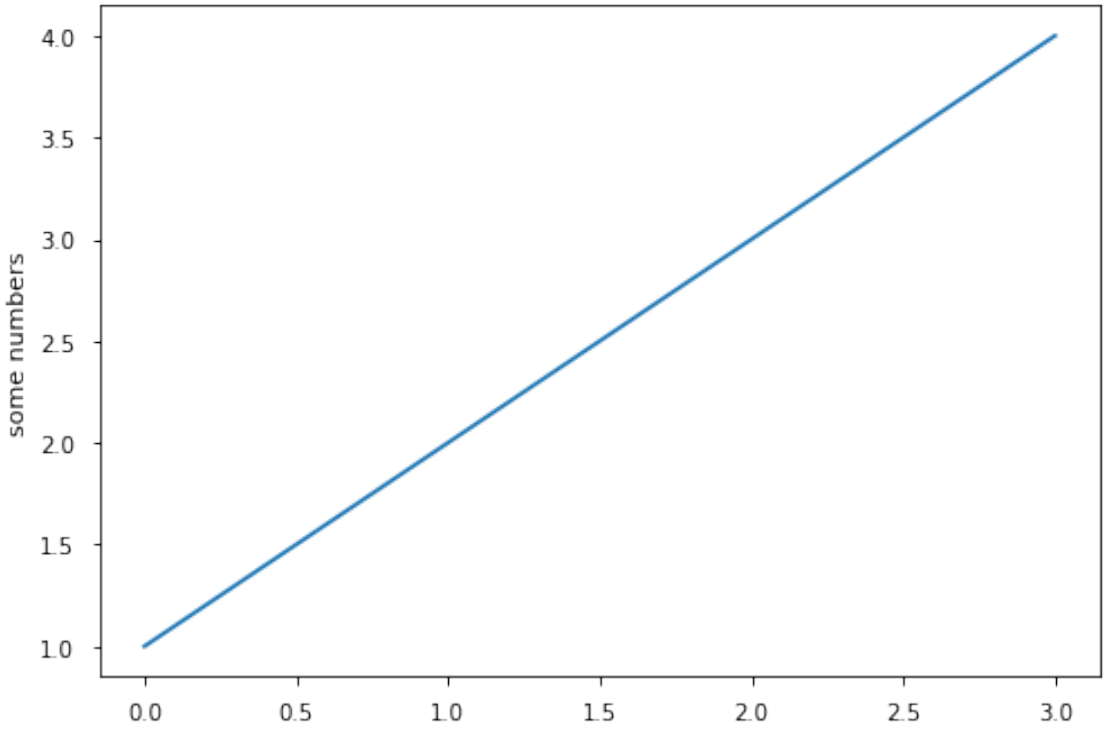
[70]:
```
fig.canvas.get_supported_filetypes()
```

[70]:
```
{'ps': 'Postscript',
 'eps': 'Encapsulated Postscript',
 'pdf': 'Portable Document Format',
 'pgf': 'PGF code for LaTeX',
 'png': 'Portable Network Graphics',
 'raw': 'Raw RGBA bitmap',
 'rgba': 'Raw RGBA bitmap',
 'svg': 'Scalable Vector Graphics',
 'svgz': 'Scalable Vector Graphics',
 'jpg': 'Joint Photographic Experts Group',
 'jpeg': 'Joint Photographic Experts Group',
 'tif': 'Tagged Image File Format',
 'tiff': 'Tagged Image File Format'}
```

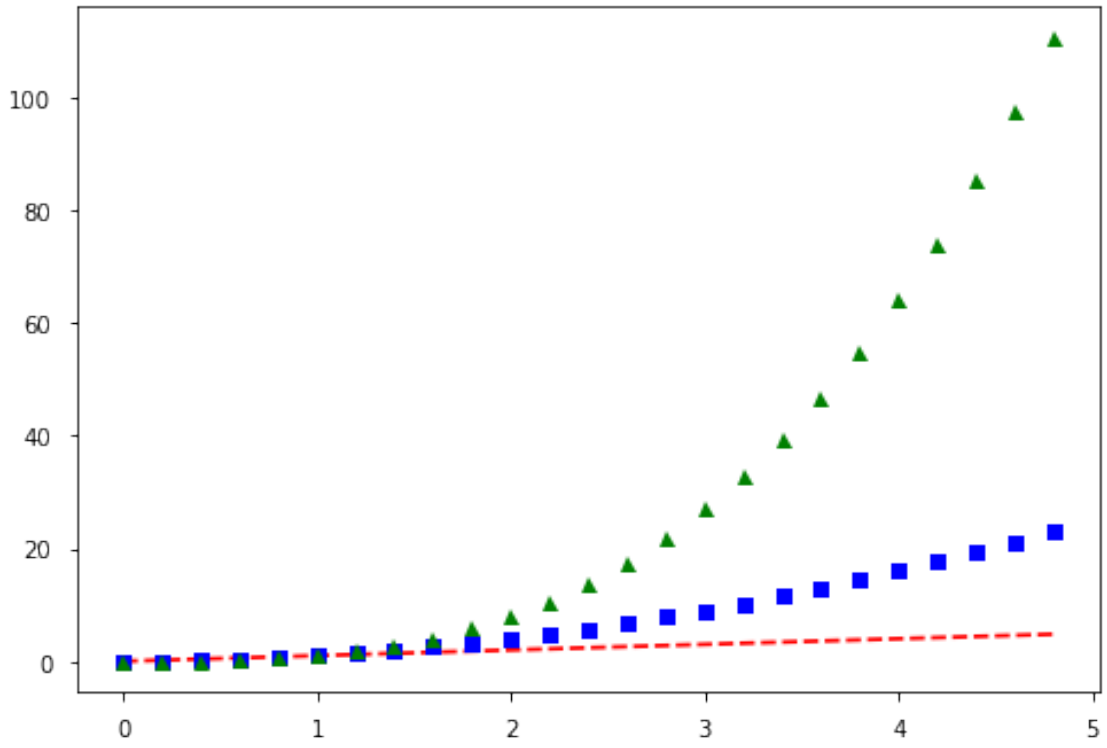
[71]:
```
# fig.savefig('sales.png', dpi = 300, bbox_inches = 'tight')
```

1.7.1 Matlab-like plotting

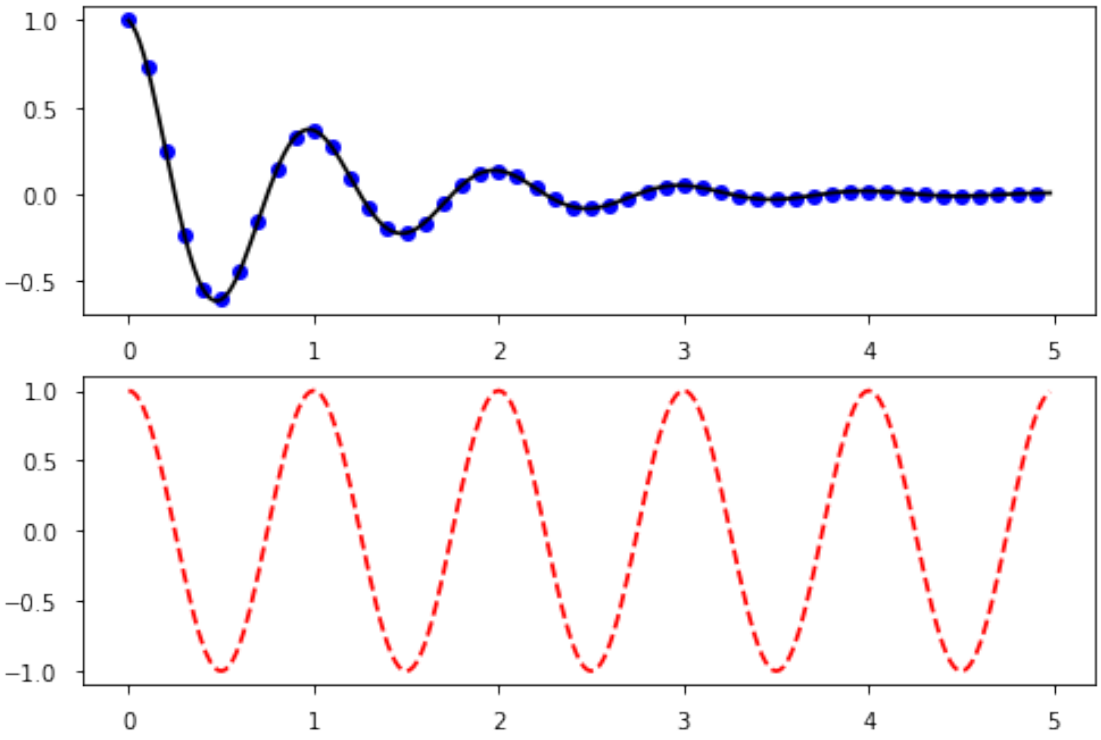
[75]:
```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```



[76]:
```
import numpy as np

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



[74]:
```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure()
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```


1.0
0.5
0.0
−0.5
0
1
2
3
4
5
1.0
0.5
0.0
−0.5
−1.0
0
1
2
3
4
5

[ ]: