

Visualization 3

Thomas Schwarz, SJ

Review

- Create figures in `matplotlib.pyplot` imported as `plt`
 - Use pandas, i.e. `df.column.plot()`
 - Limited functionality, but easiest to use
 - Use `plt.plot`
 - Matplotlib style
- Use the OO interface
 - `fig, ax = plt.subplots(8, 8, figsize=(6, 6))`

Review

- Scripts:
 - Need to call `plt.show()` but only once
- IPython
 - Use the `%matplotlib` magic command
 - `plt` commands now run in place to update a plot
 - Might need to force updates with `plt.draw()`
- IPython notebook
 - `%matplotlib notebook`
 - Interactive plots embedded in notebook
 - `% matplotlib inline`
 - Static images embedded in notebook

Review

- plot for line plots: give x- and y-values
- scatter plot with setting marker as third argument
 - color codes and line codes
 - `color`, `linestyle`, or abbreviation
- control with

`plt`

`ax`

`plt.xlabel()` `ax.set_xlabel()`

`plt.xlim()` `ax.set_xlim()`

`plt.title()` `ax.set_title()`

Multiple Subplots

- Can use `fig.add_axes()` to manually create subplots.
 - Create a figure
 - `fig = plt.figure()`
 - Add axes:
 - First argument is a list with x-offset, y-offset, x-height, y-height out of 0 ... 1
 - `ax1 = fig.add_axes([0.1, 0.1, 0.88, 0.88],`
...
 - Remember to leave space for ticks

Multiple Subplots

- We can then add limits, ticks and labels

```
ax1 = fig.add_axes([0.1, 0.1, 0.88, 0.88],  
                  xlim = (0, np.pi),  
                  ylim = (-1.1, 1.1),  
                  xticks=[0, np.pi/2, np.pi],  
                  xticklabels=['0', 'pi/2', 'pi'])
```

- This is an axes with origin at 0.1, 0.1 going to 0.98, 0.98

Multiple Subplots

- Can add another axes in the right upper corner

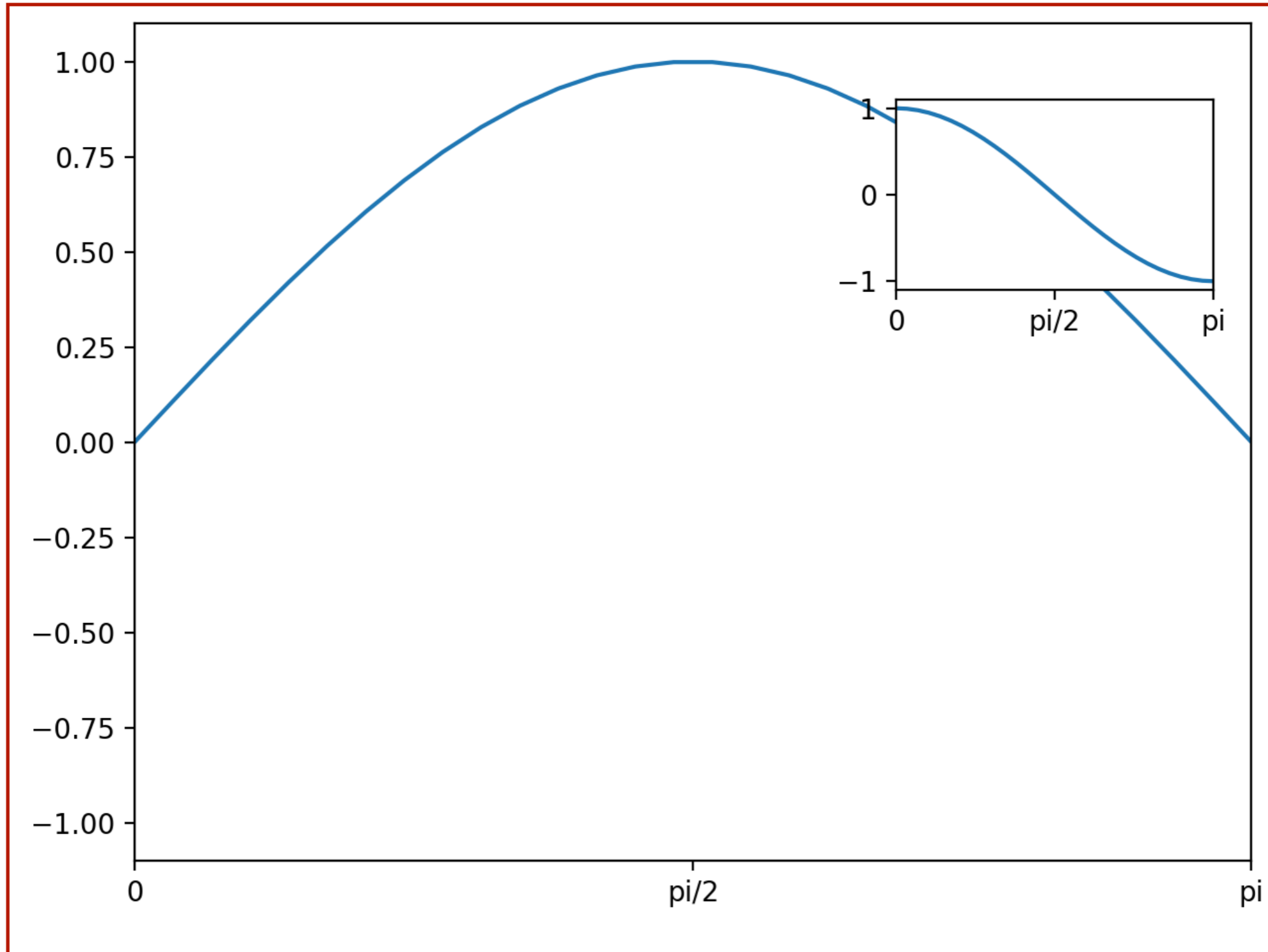
```
ax2 = fig.add_axes([0.7, 0.7, 0.25, 0.2],  
                   xlim = (0, np.pi),  
                   ylim = (-1.1, 1.1),  
                   xticks=[0, np.pi/2, np.pi],  
                   xticklabels=['0', 'pi/2', 'pi'])
```

Multiple Subplots

- Finally, plot random (trig) functions

```
x = np.linspace(0, np.pi, 30)
ax1.plot(x, np.sin(x))
ax2.plot(x, np.cos(x))
plt.show()
```


Multiple Subplots



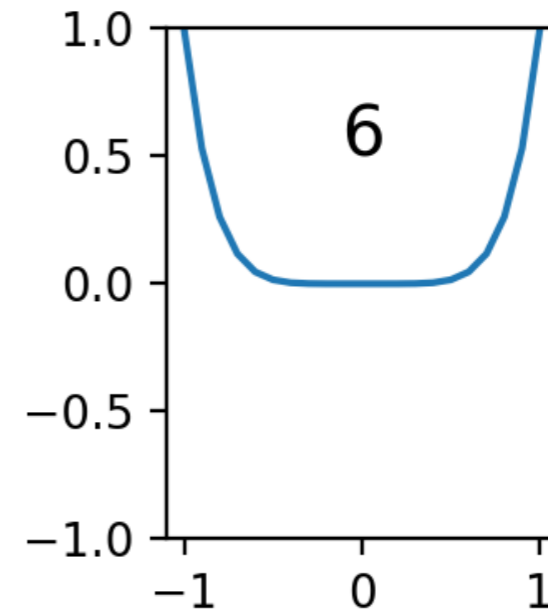
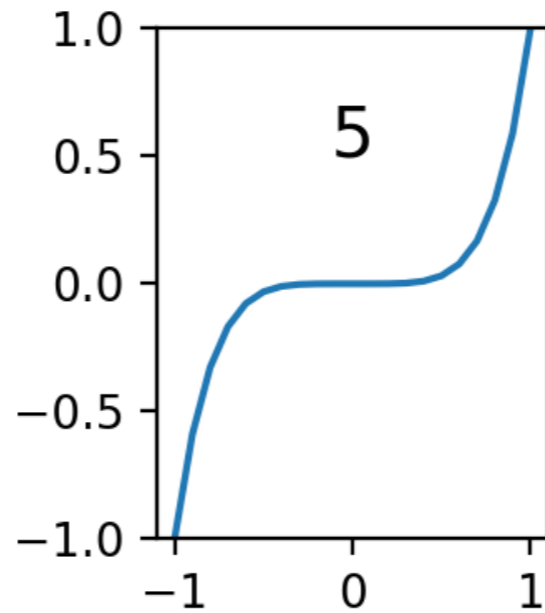
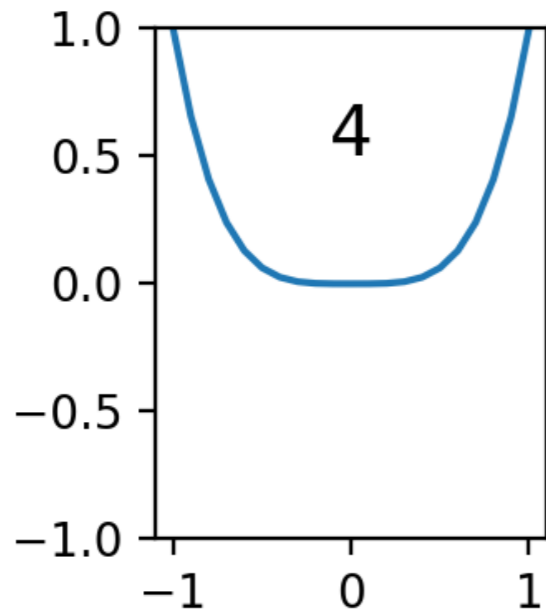
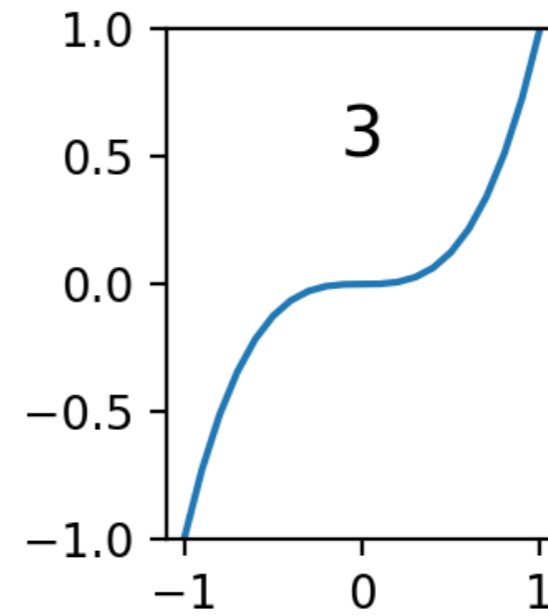
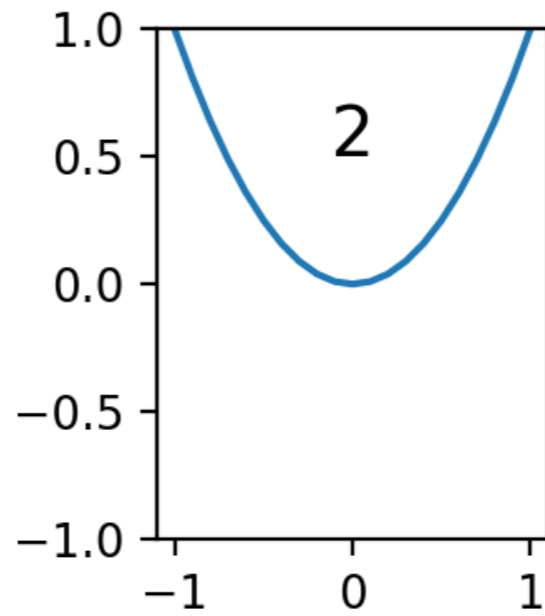
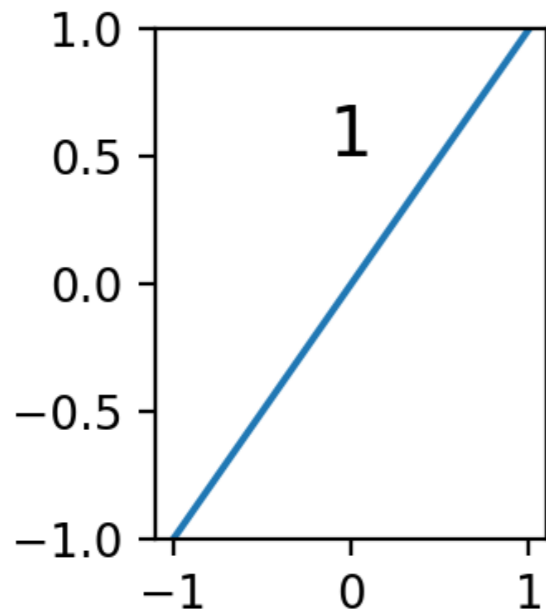
Multiple Subplots

- Can create figure with a subplot
 - `fig = plt.figure()`
- Adjust layout
 - `hspace` - space between rows of subplots (height)
 - `wspace` - space between columns of subplots (width)
 - `fig.subplots_adjust(hspace=0.4,
wspace = 0.6)`

Multiple Subplots

- Create 6 axes in two rows and three columns
 - `for i in range(1, 7):`
`ax = fig.add_subplot(2, 3, i, ylim = (-1, 1))`
- Write an interior text
 - The position is relative to the grid in the subplot
 - `ax.text(0, 0.5, str(i),`
`fontSize = 15,`
`ha='center')`
 - Then plot x^{*i} `x = np.linspace(-1, 1, 21)`
`ax.plot(x, np.power(x, i))`

Multiple Subplots



Multiple Subplots

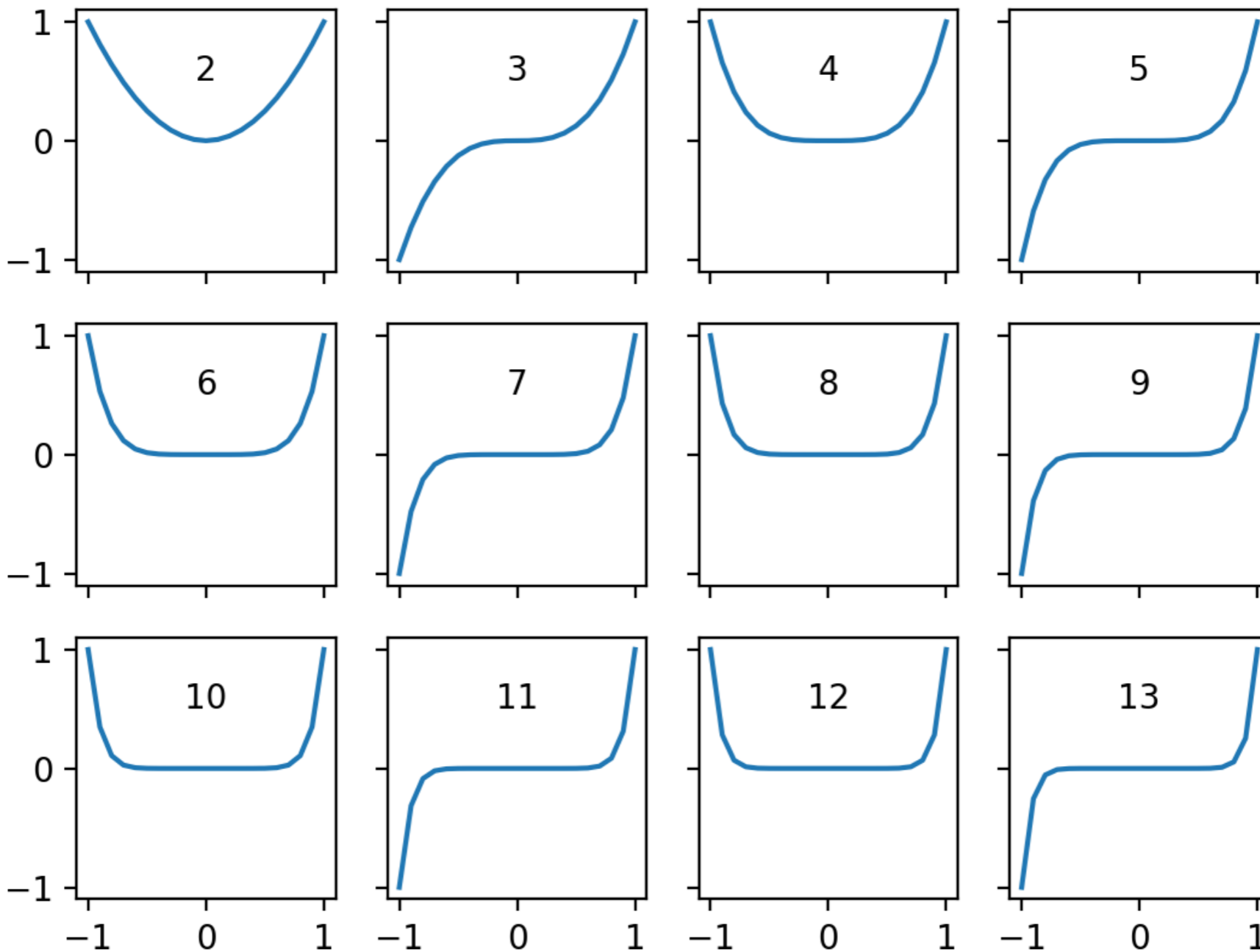
- Can use subplots (with a plural-s) to create a number of plots
- Can make share coordinate layout
 - `fig, axes = plt.subplots(3, 4, sharex='col', sharey='row')`

Multiple Subplots

- The axes is a 3 by 4 matrix
 - Notice the way we are accessing them by a single bracket

```
for i in range(3):  
    for j in range(4):  
        degree = 4*i+j+2  
        axes[i,j].text(0, 0.5, str(degree), ha='center')  
        axes[i,j].plot(x, np.power(x, degree))
```

Multiple Subplots



Multiple Subplots

- For more complicated arrangements, look at `plt.GridSpec`

Controlling Ticks

- Figure: bounding box, contains
 - Axes objects, which contain
 - A variety of other objects representing plot contents
 - Such as xaxis, yaxis
 - contains properties of lines, ticks, and labels
 - Has major and minor locator object
 - Has major and minor formatter object

Controlling Ticks

- No ticks at all:
 - `ax = plt.axes(yscale='log')`
`ax.xaxis.set_major_locator(plt.NullLocator())`

Controlling Ticks

- Example: Handwriting recognition set
 - In `sklearn.datasets`

```
from sklearn.datasets import load_digits
```

- Create a figure and pack it densely

```
fig = plt.figure(figsize=(6,6)) #in inches
fig.subplots_adjust(left=0, right=1, bottom=0,
                    top=1, hspace=0.05,
                    wspace = 0.05)
```

Controlling Ticks

- Create 64 axes

```
for i in range(64):  
    ax = fig.add_subplot(8, 8, i+1)
```

- No ticks

```
ax.xaxis.set_major_locator(plt.NullLocator())  
ax.yaxis.set_major_locator(plt.NullLocator())
```

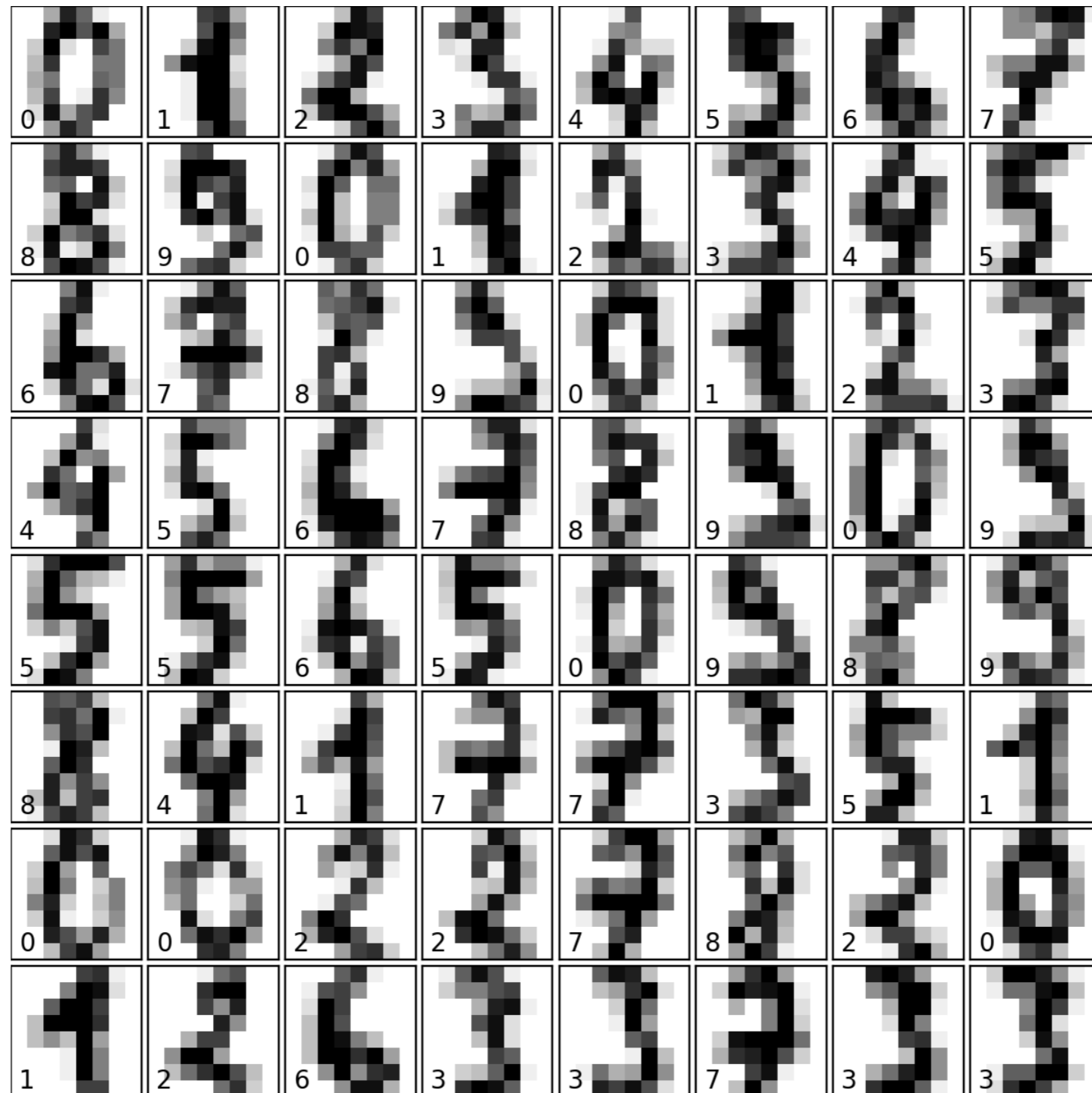
- Show images

```
ax.imshow(digits.images[i],  
          cmap=plt.cm.binary,  
          interpolation='nearest')
```

- Show target text

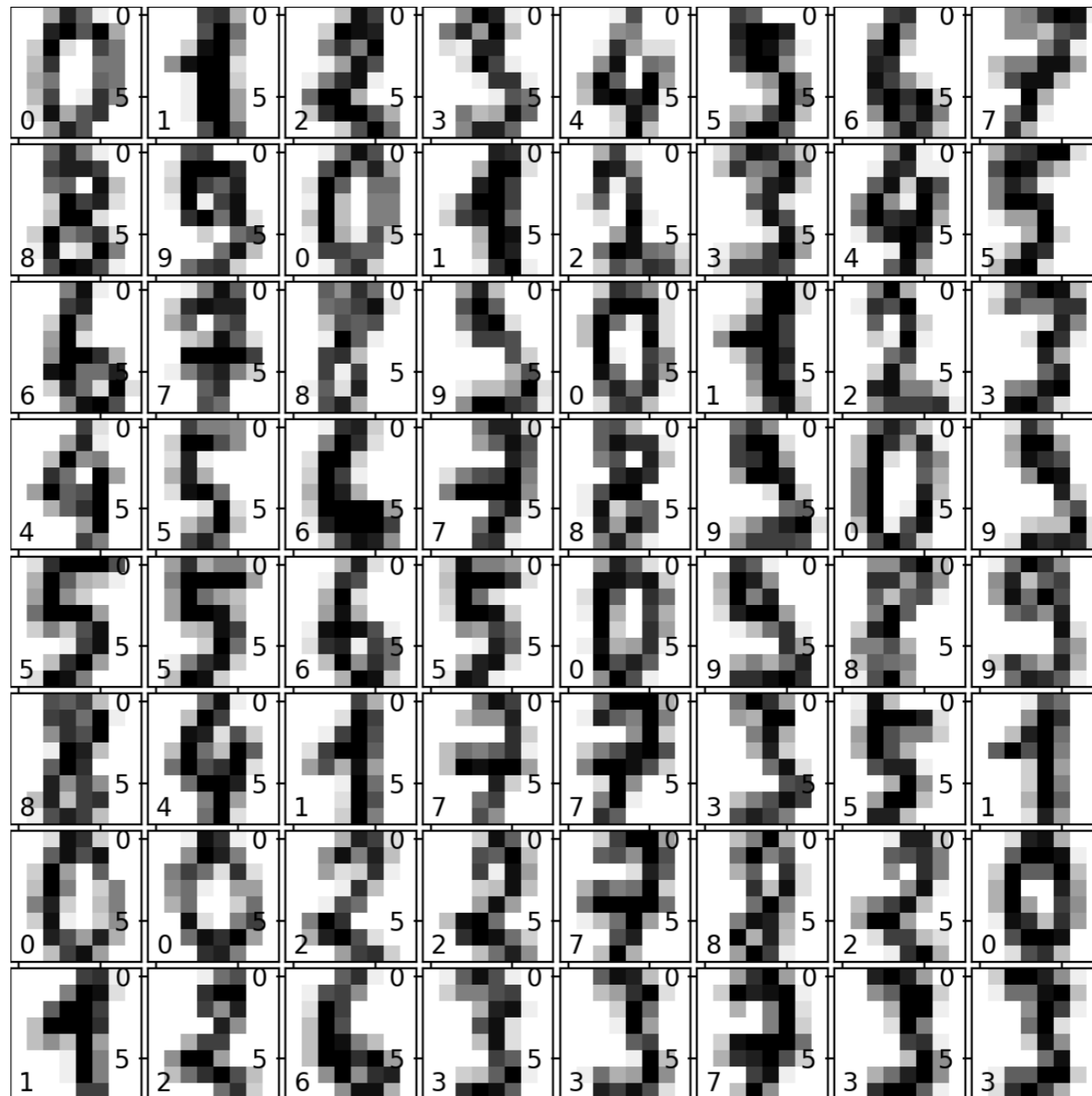
```
ax.text(0, 7, str(digits.target[i]))
```

Controlling Ticks



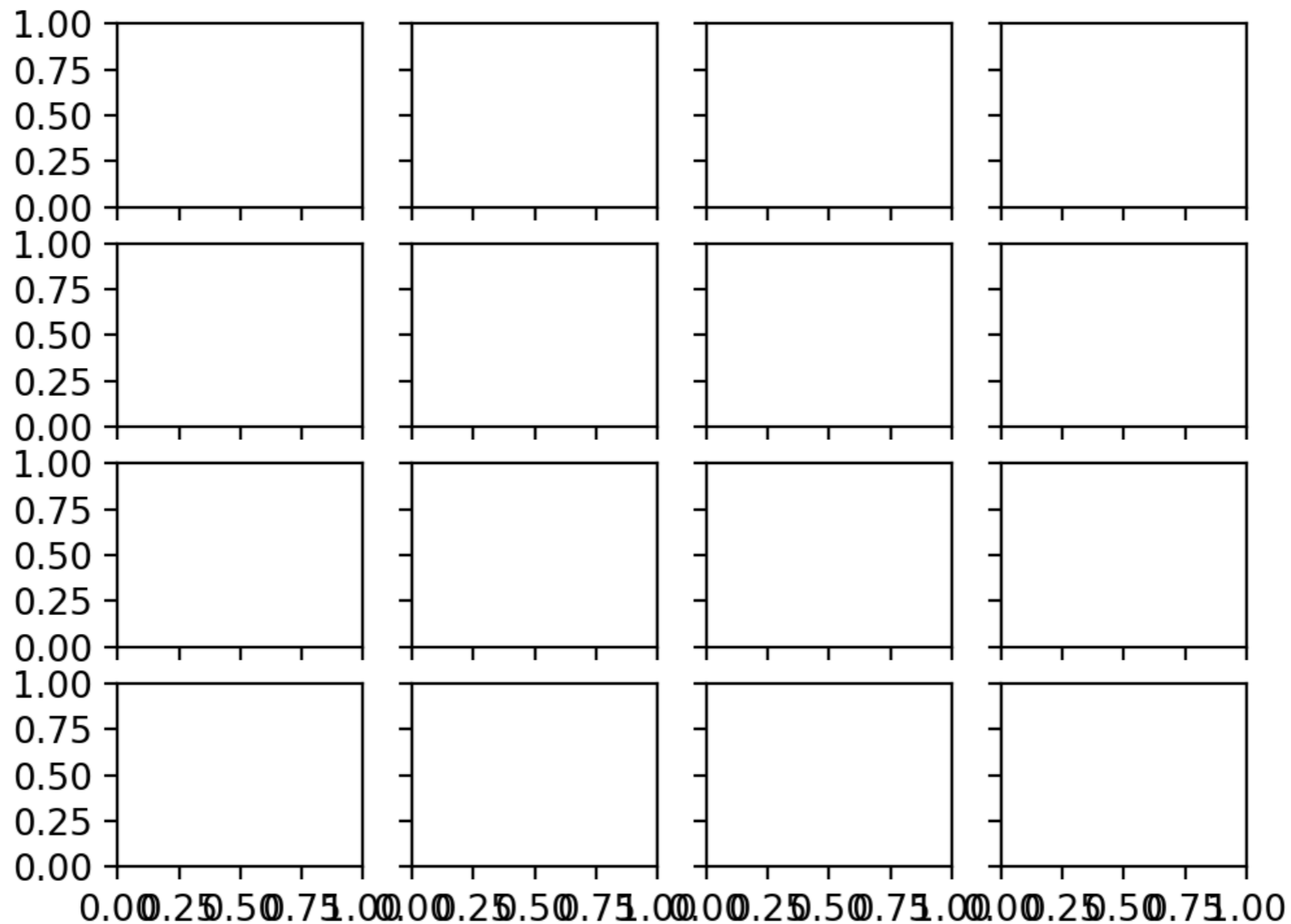
Controlling Ticks

- Without suppressing ticks



Controlling Ticks

- Sometimes, ticks can be too close together



Controlling Ticks

- Use the locator to determine the number of ticks

```
fig, axes = plt.subplots(4,4, sharex=True, sharey=True)
for ax in axes.flat:
    ax.xaxis.set_major_locator(plt.MaxNLocator(4))
    ax.yaxis.set_major_locator(plt.MaxNLocator(4))
```

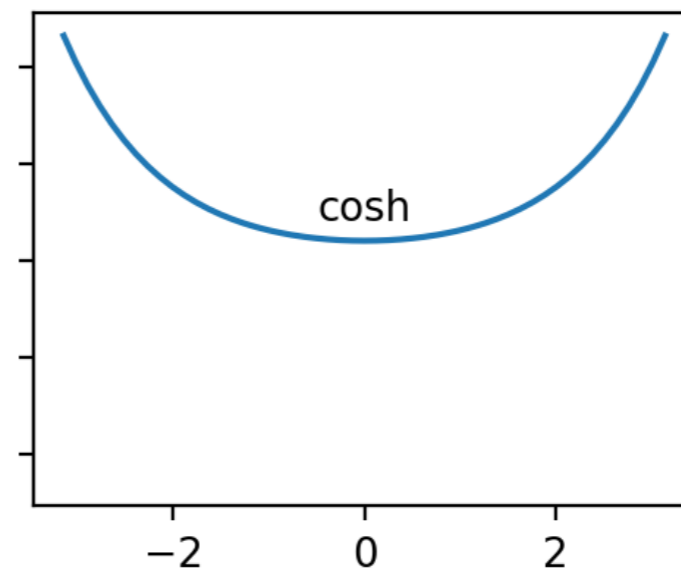
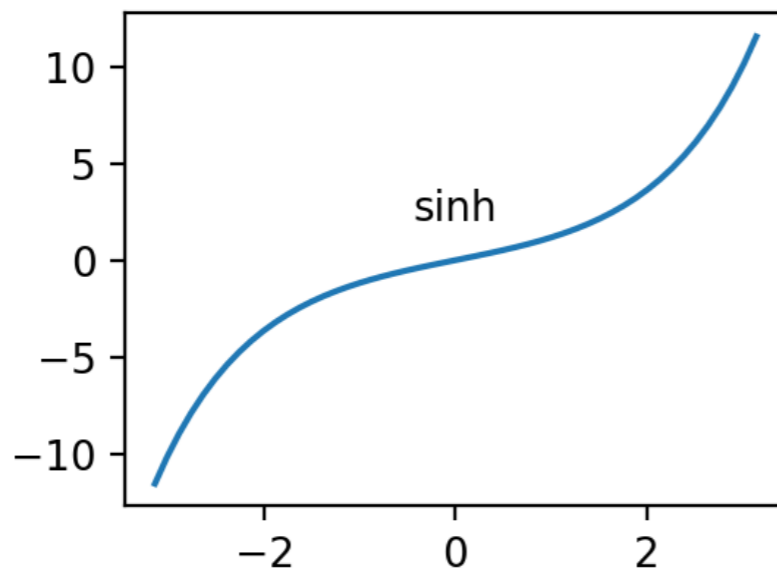
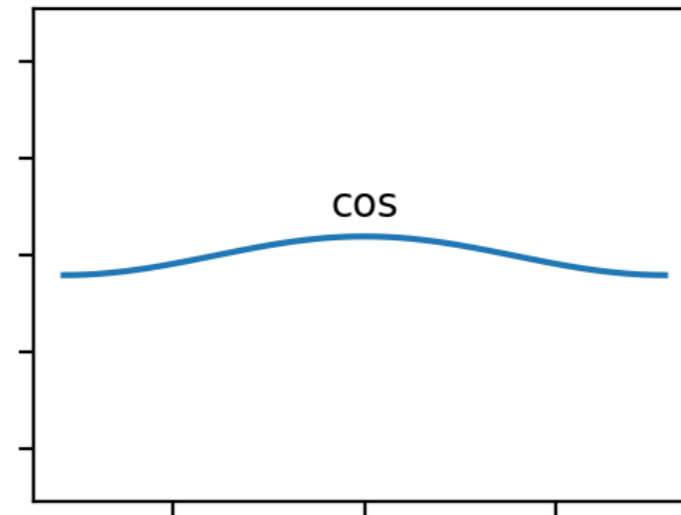
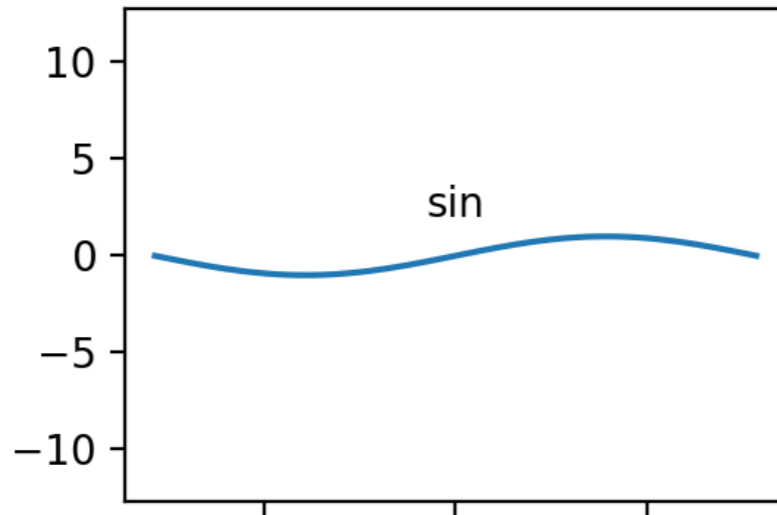

Controlling Ticks

- Sometimes, need more control

```
x= np.linspace(-np.pi, np.pi, 51)
fig, axes = plt.subplots(2,2, sharex=True, sharey=True)
axes[0,0].plot(x, np.sin(x))
axes[0,1].plot(x, np.cos(x))
axes[1,0].plot(x, np.sinh(x))
axes[1,1].plot(x, np.cosh(x))
axes[0,0].text(0,2, 'sin', ha='center')
axes[0,1].text(0,2, 'cos', ha='center')
axes[1,0].text(0,2, 'sinh', ha='center')
axes[1,1].text(0,2, 'cosh', ha='center')
```

Controlling Ticks

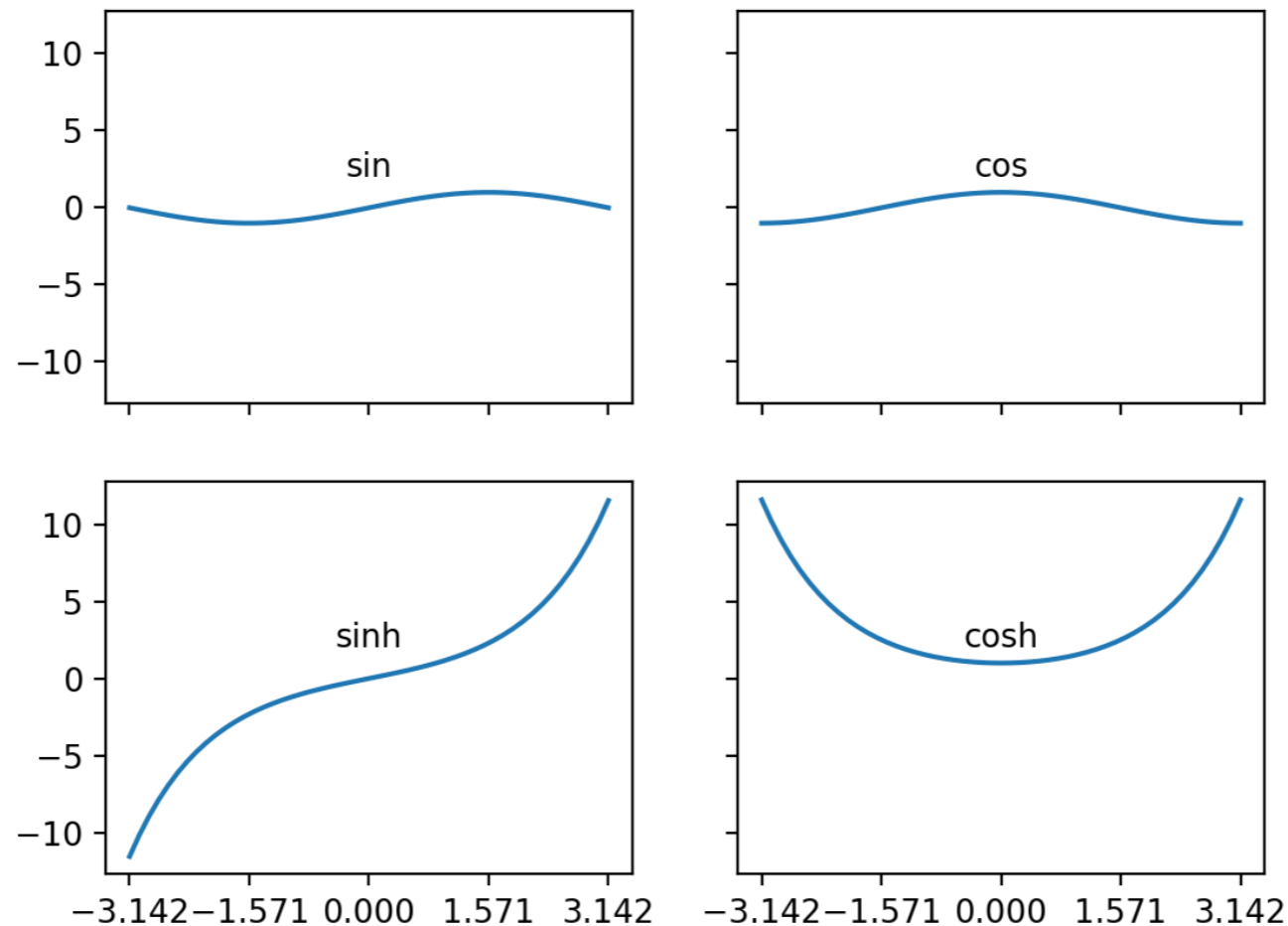
- More natural to have the x-axis use multiple of π



Controlling Ticks

- Set the multiplelocator object

```
axes[0,0].xaxis.set_major_locator(  
    plt.MultipleLocator(np.pi/2))
```



Controlling Ticks

- The ticks are fine, but need to write in terms of multiples of π
- Create our own format function using TeX

```
def my_format_func(value, tick_number):
    nn = int(np.round(2*value / np.pi))
    if nn == 0:
        return '0'
    elif nn == 1:
        return r'$\pi/2$'
    elif nn%2:
        return r'${0}\pi/2$'.format(nn)
    else:
        return r'${0}\pi$'.format(nn//2)
```

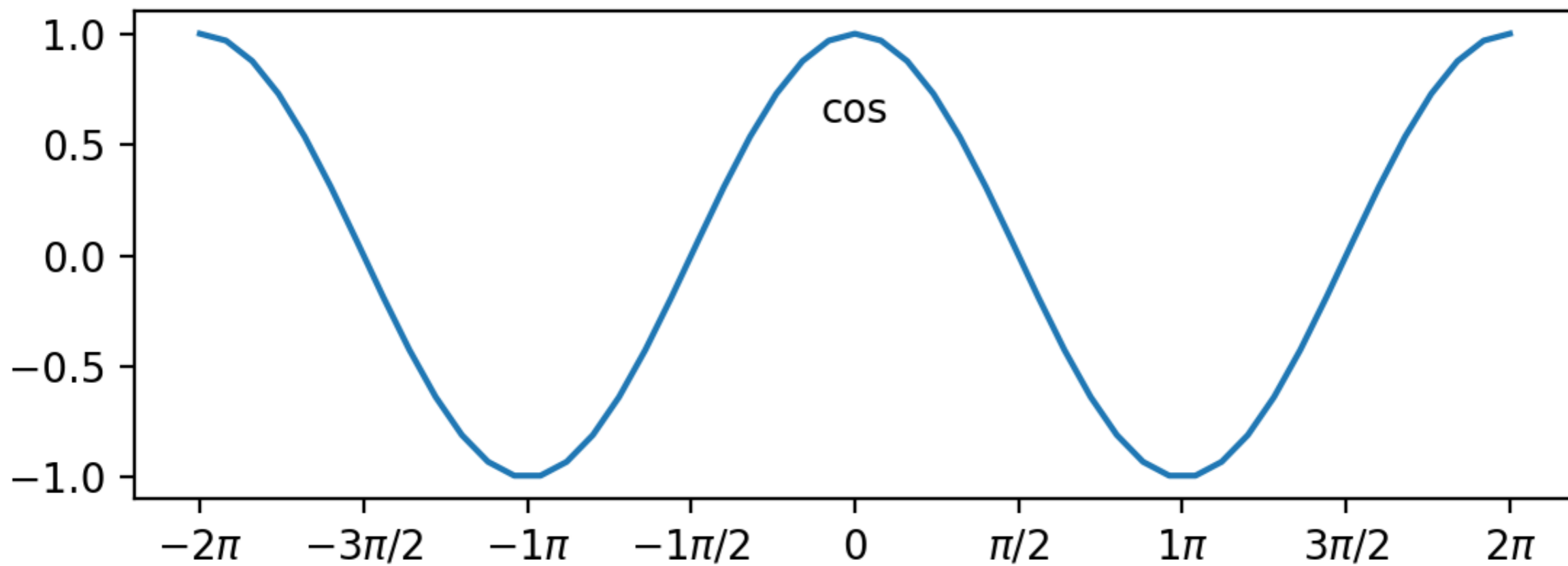
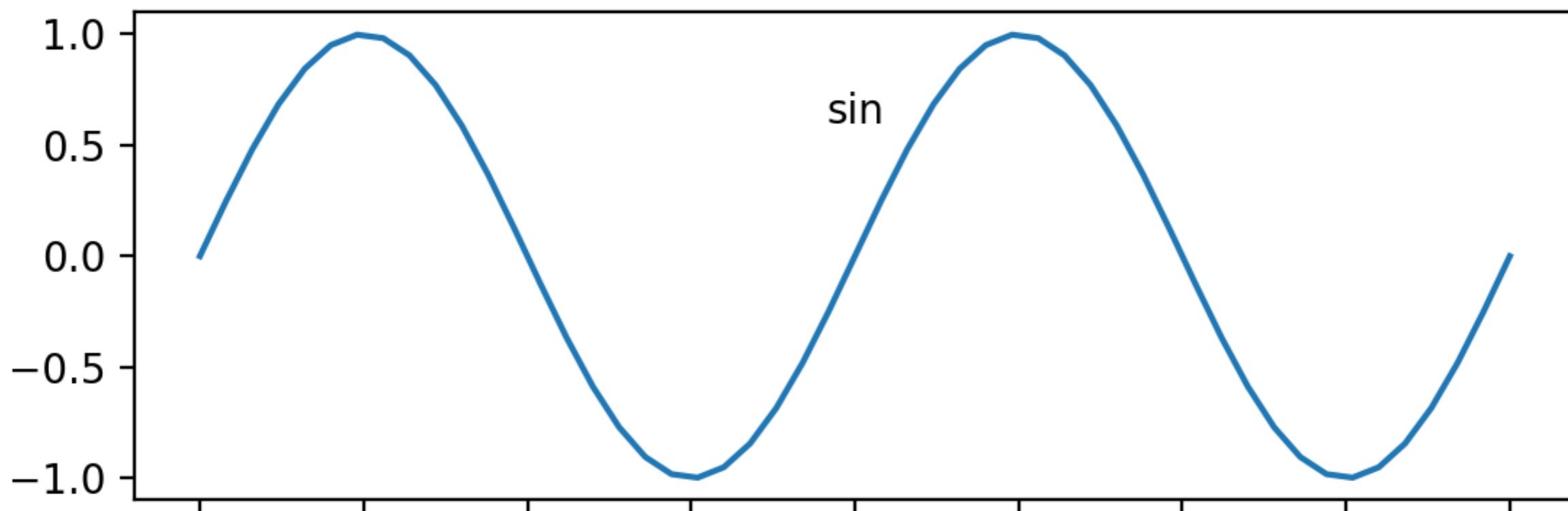
Controlling Ticks

- Now it works:

```
x= np.linspace(-2*np.pi, 2*np.pi, 51)
fig, axes = plt.subplots(2,1, sharex=True, sharey=True)
axes[0].plot(x, np.sin(x))
axes[1].plot(x, np.cos(x))

axes[0].text(0,0.6, 'sin', ha='center')
axes[1].text(0,0.6, 'cos', ha='center')

axes[0].xaxis.set_major_locator(plt.MultipleLocator
    (np.pi/2))
axes[0].xaxis.set_major_formatter(plt.FuncFormatter
    (my_format_func))
```



Seaborn

- Matplotlib is based on an old version of matlab
 - Defaults are not all that pretty
 - Calculate random walk

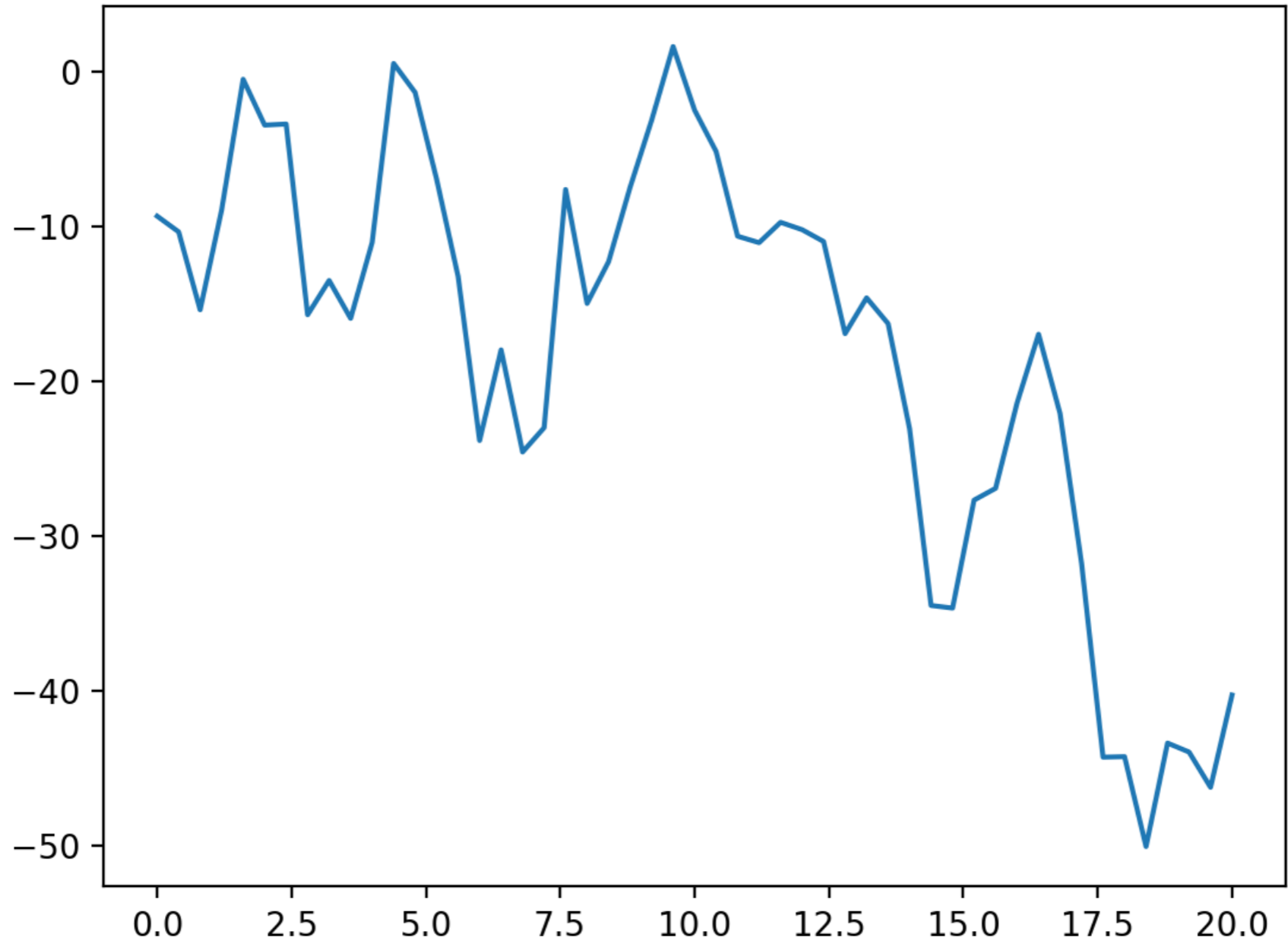
```
x = np.linspace(0, 20, 51)
y = np.cumsum(np.random.normal(0, 5, 51))

plt.plot(x, y)

plt.show()
```

Seaborn

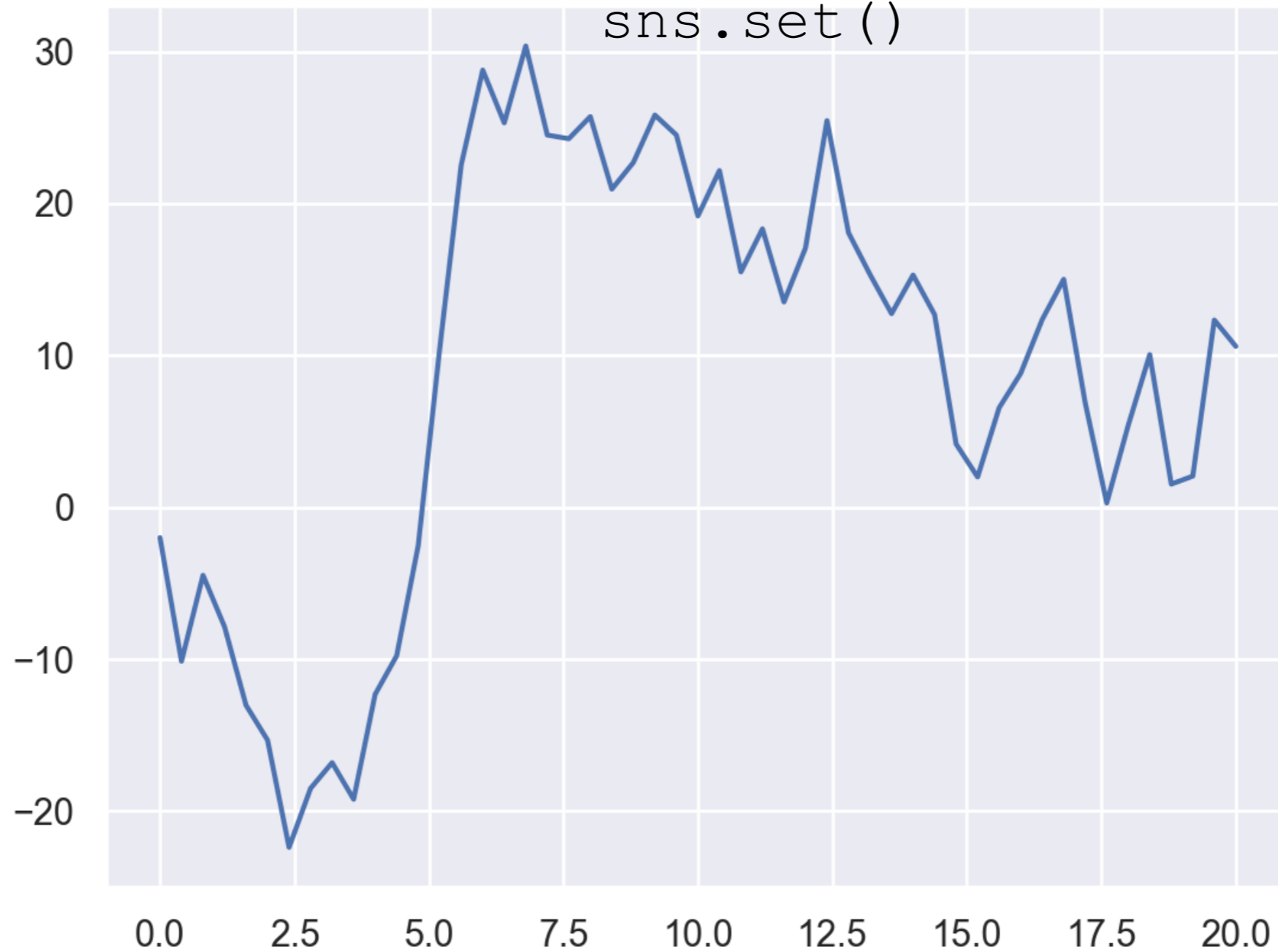
- Without seaborn



Seaborn

- With Seaborn

```
import seaborn as sns  
sns.set()
```



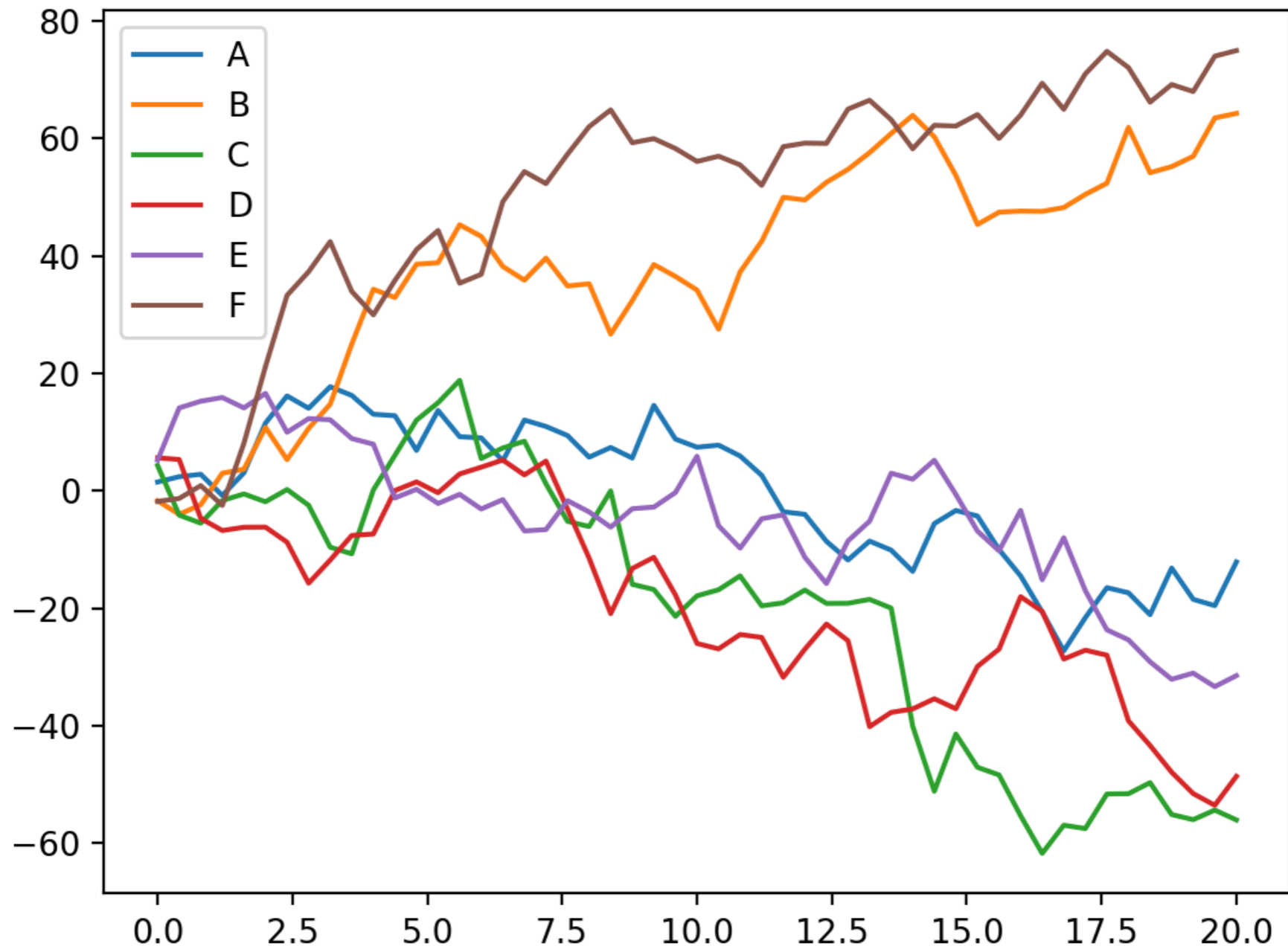
Seaborn

- The difference extends to labeling

```
sns.set()  
x = np.linspace(0, 20, 51)  
label = 'ABCDEF'  
for letter in label:  
    y = np.cumsum(np.random.normal(0, 5, 51))  
    plt.plot(x, y, label=letter)  
plt.legend()
```

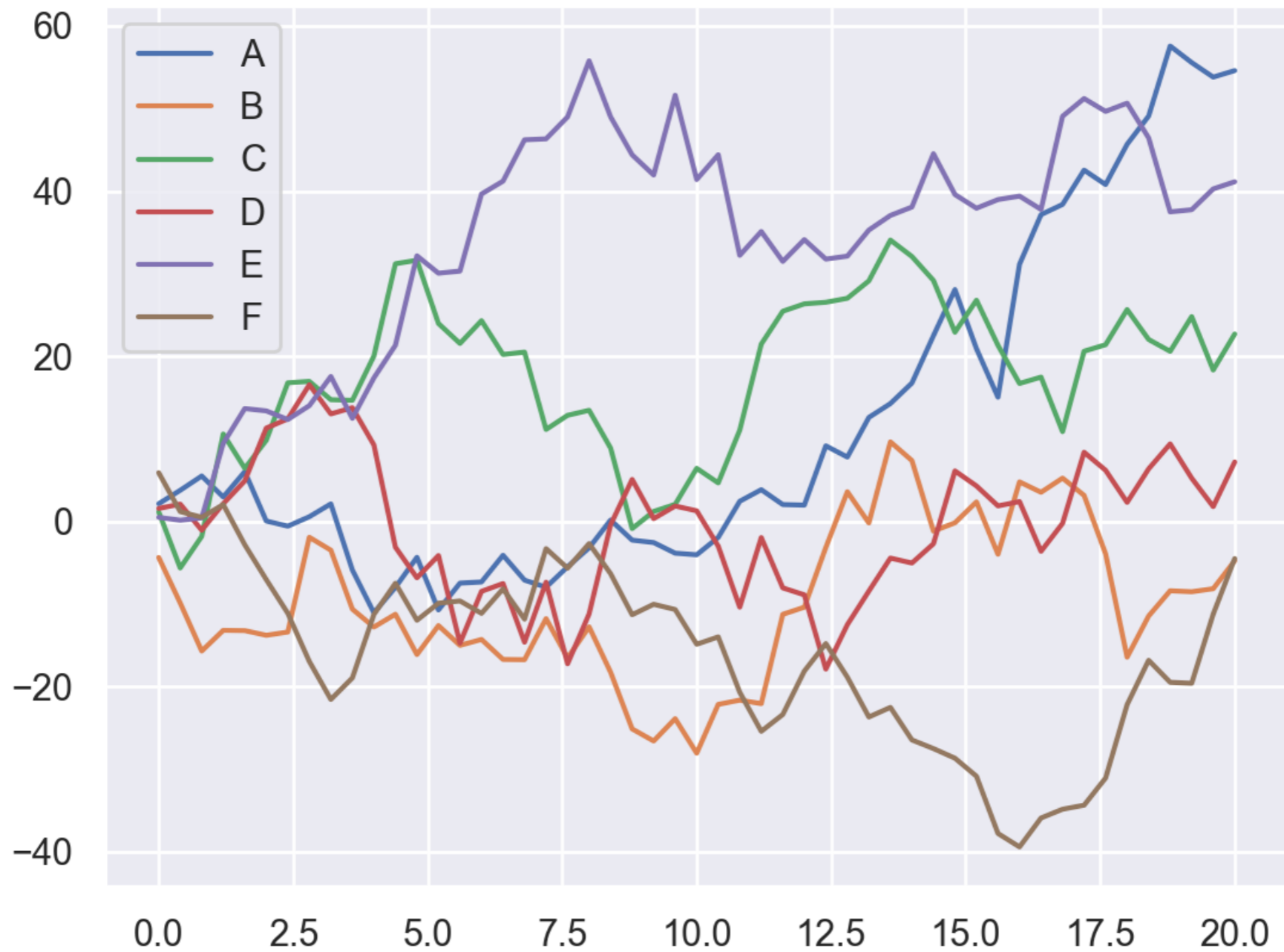
Seaborn

- without Seaborn



Seaborn

- With Seaborn



Seaborn

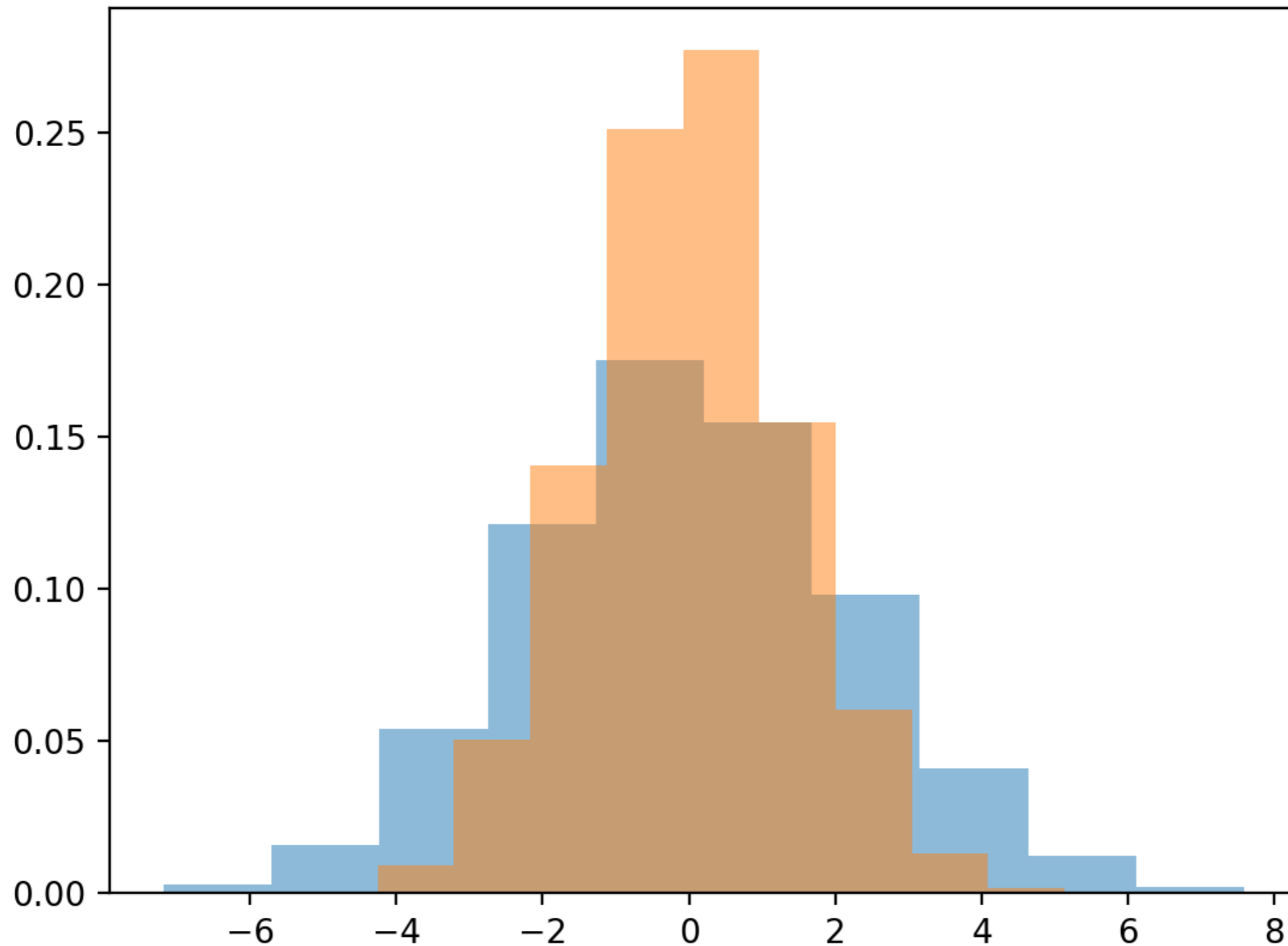
- Seaborn provides many statistical visualization tools
 - Example: Generate a multivariate normal distribution with covariance matrix $\begin{pmatrix} 5 & 2 \\ 2 & 2 \end{pmatrix}$ around the origin
 - ```
data=np.random.multivariate_normal(
 [0,0],
 [[5,2],[2,2]],
 size = 5000)
```

# Seaborn

- With plt, easy to generate histogram

```
df = pd.DataFrame(data, columns=['x', 'y'])
for col in ['x', 'y']:
 plt.hist(df[col], normed=True, alpha = 0.5)
```

# Seaborn



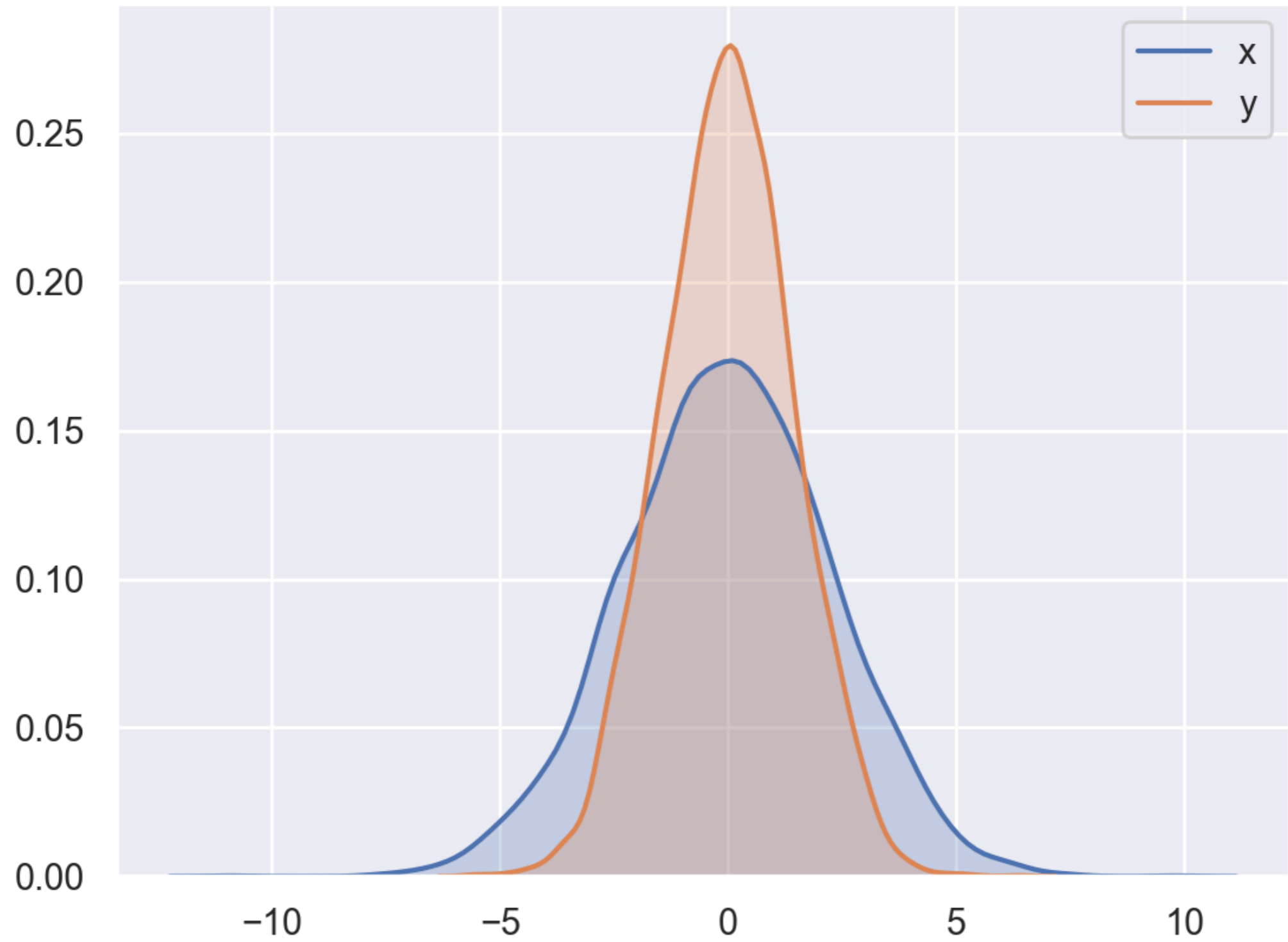
# Seaborn

- With Seaborn, we can use a "kernel density estimation"

```
for col in ['x', 'y']:
 sns.kdeplot(df[col], shade=True)
```



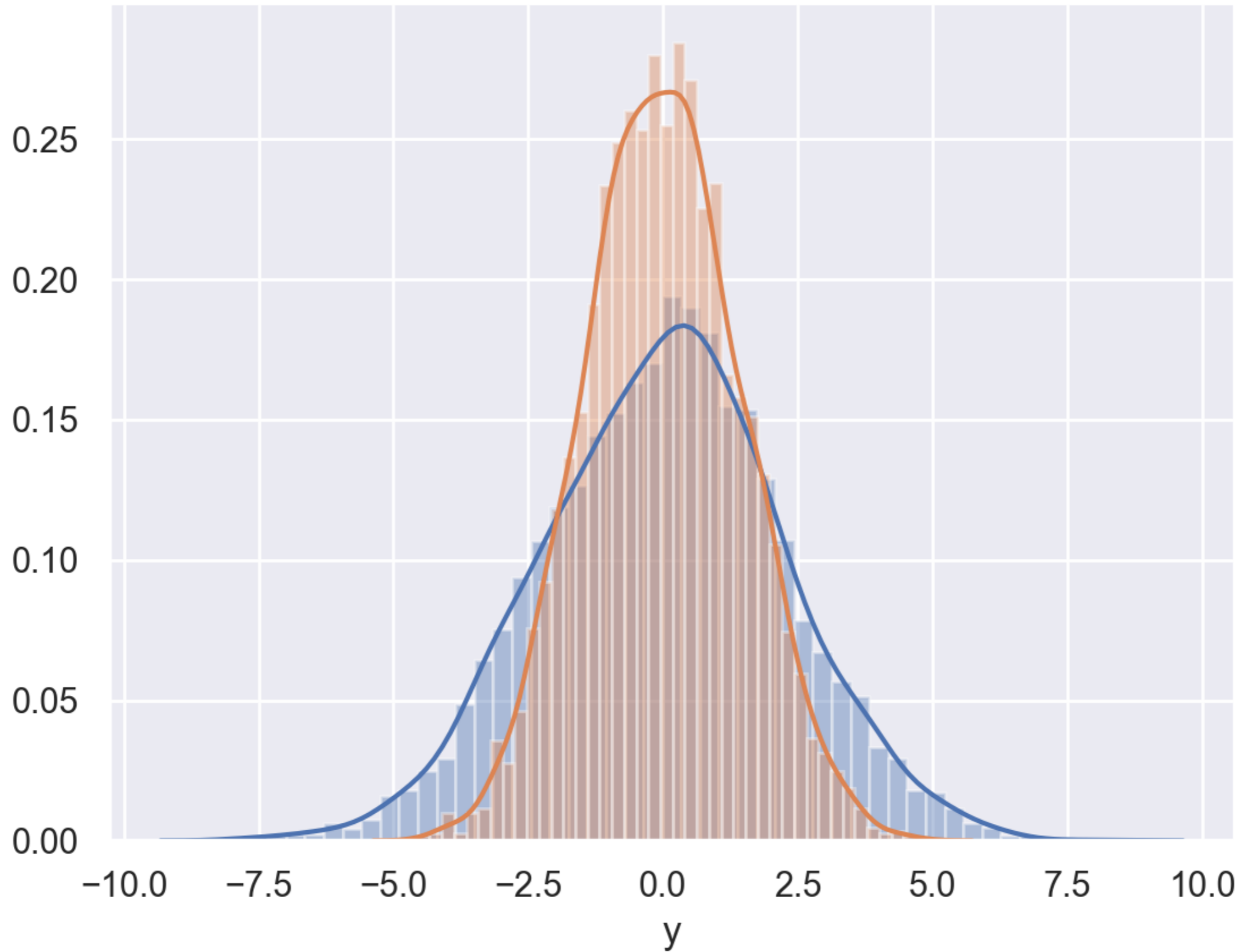
# Seaborn



# Seaborn

- Can combine with distplot
- ```
for col in ['x', 'y']:  
    sns.distplot(df[col])
```

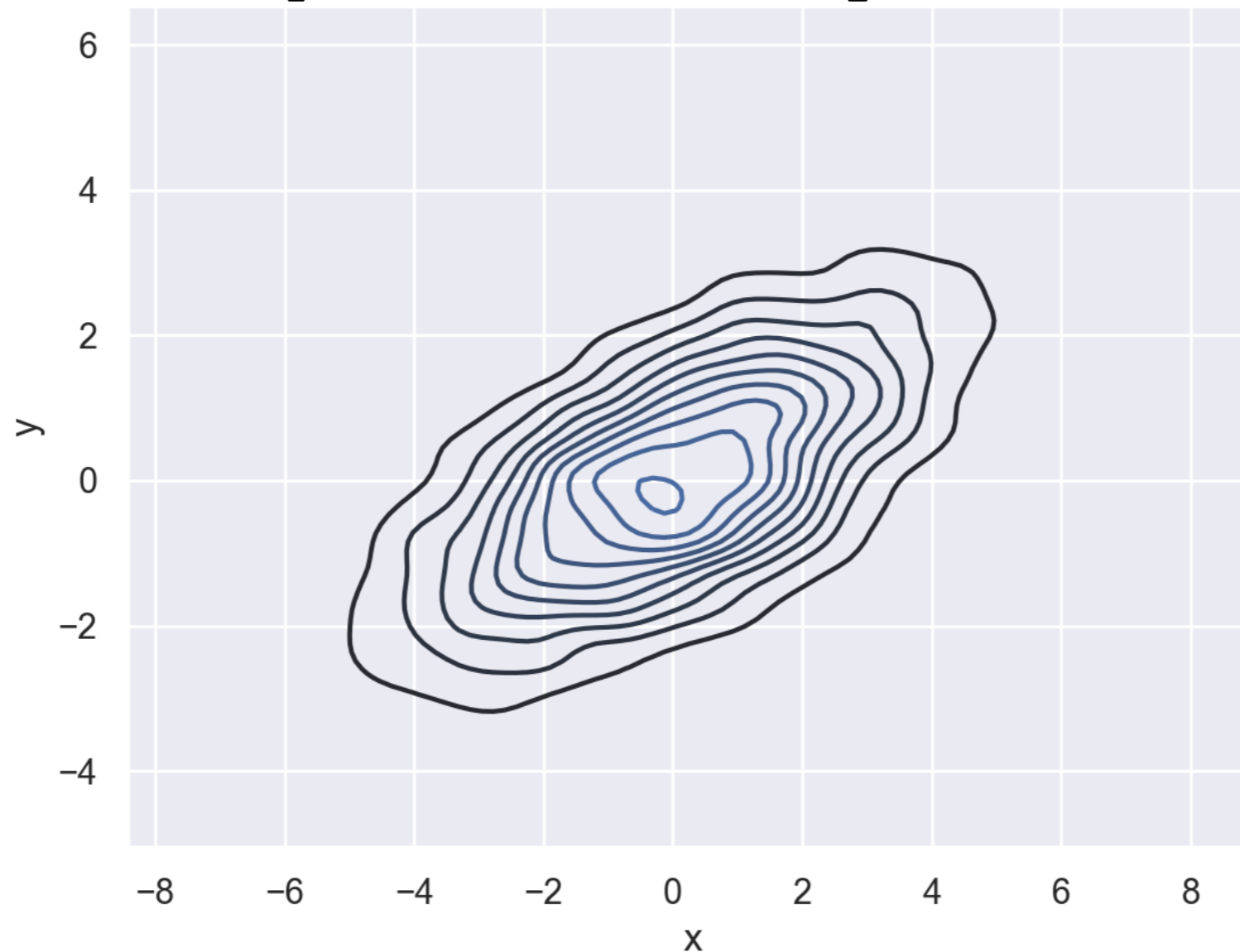
Seaborn



Seaborn

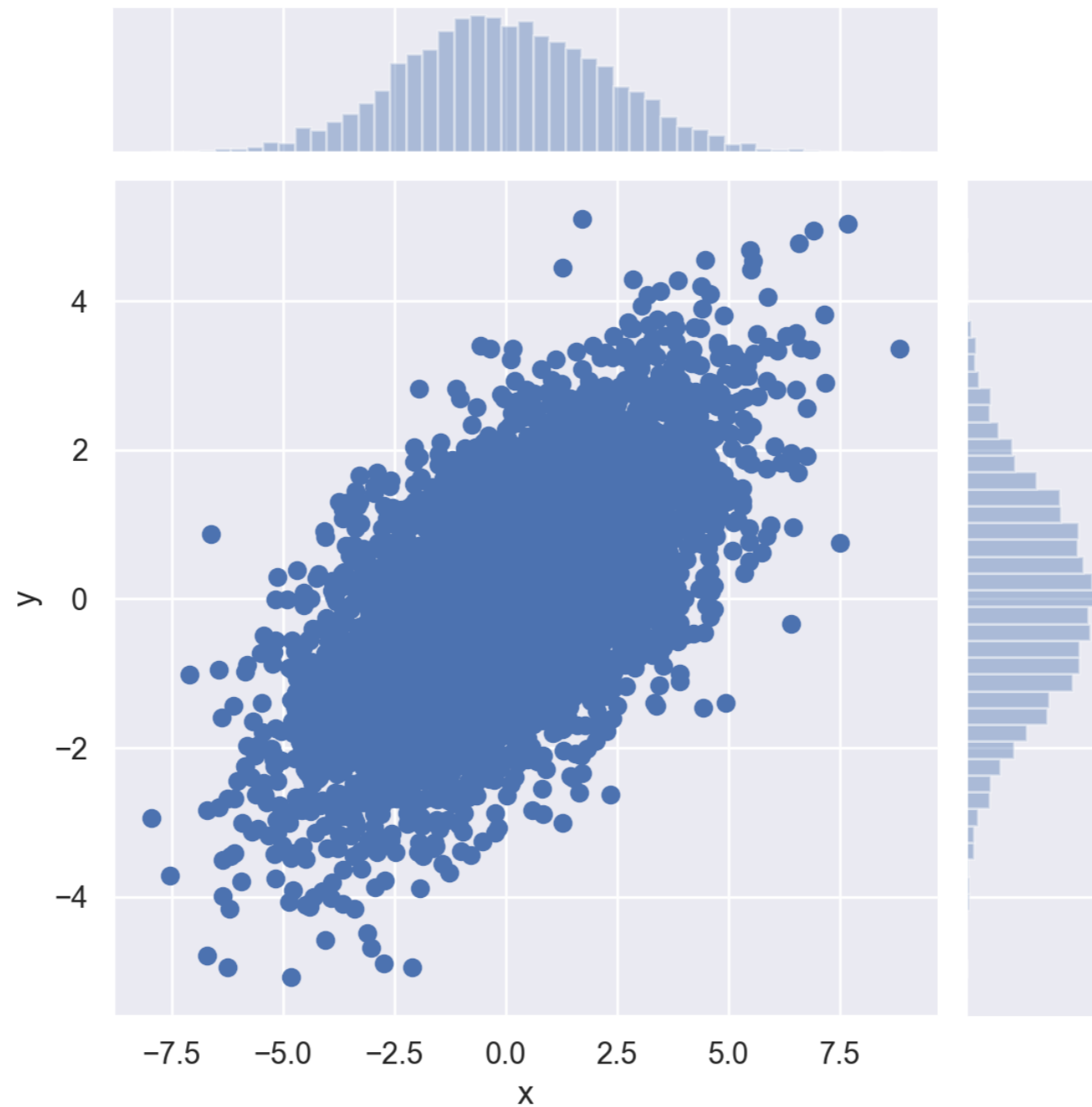
- Can also use a two dimensional kernel density estimation plot

```
sns.kdeplot(df.x, df.y)
```



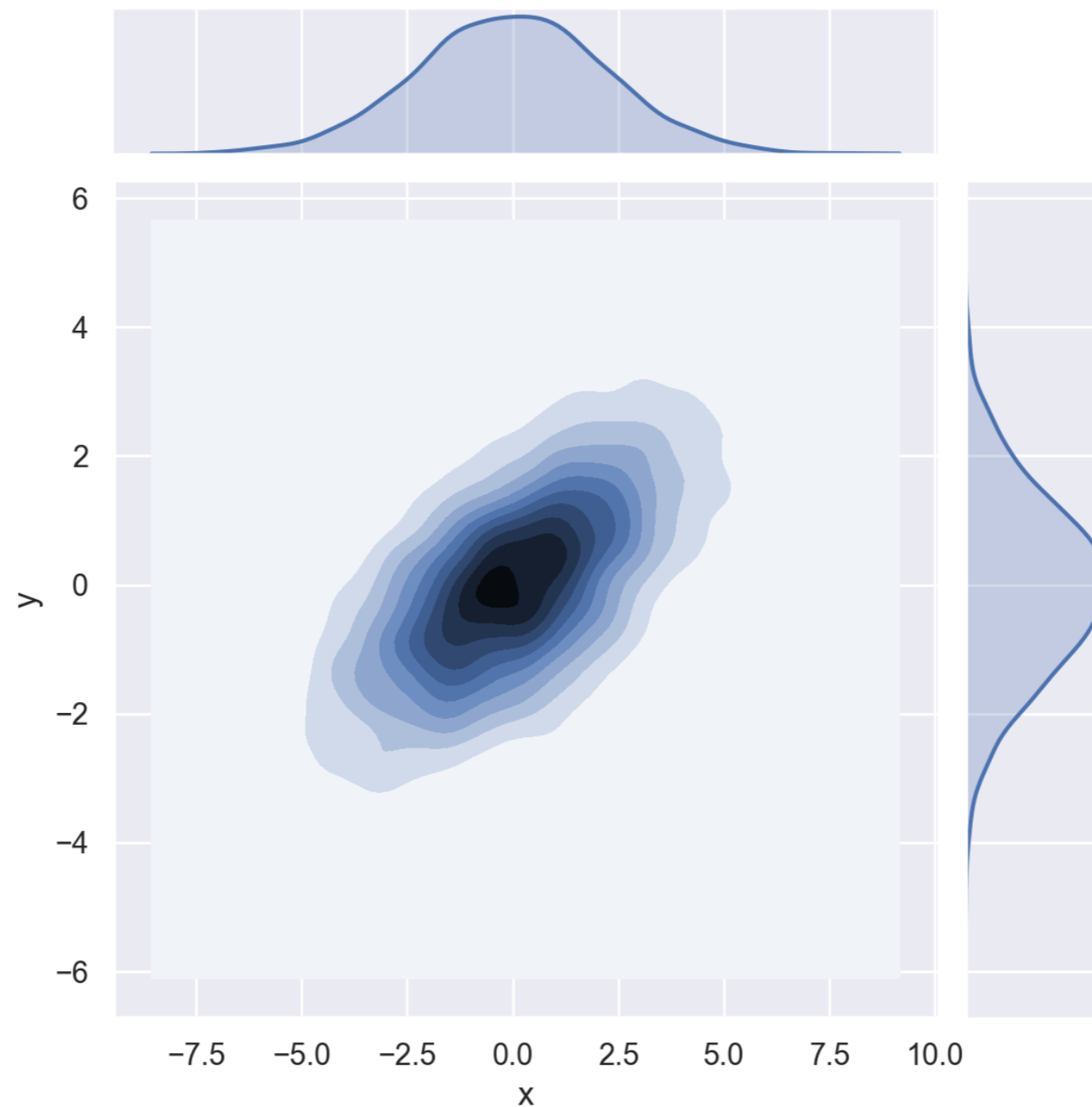
Seaborn

- Or use `sns.jointplot`



Seaborn

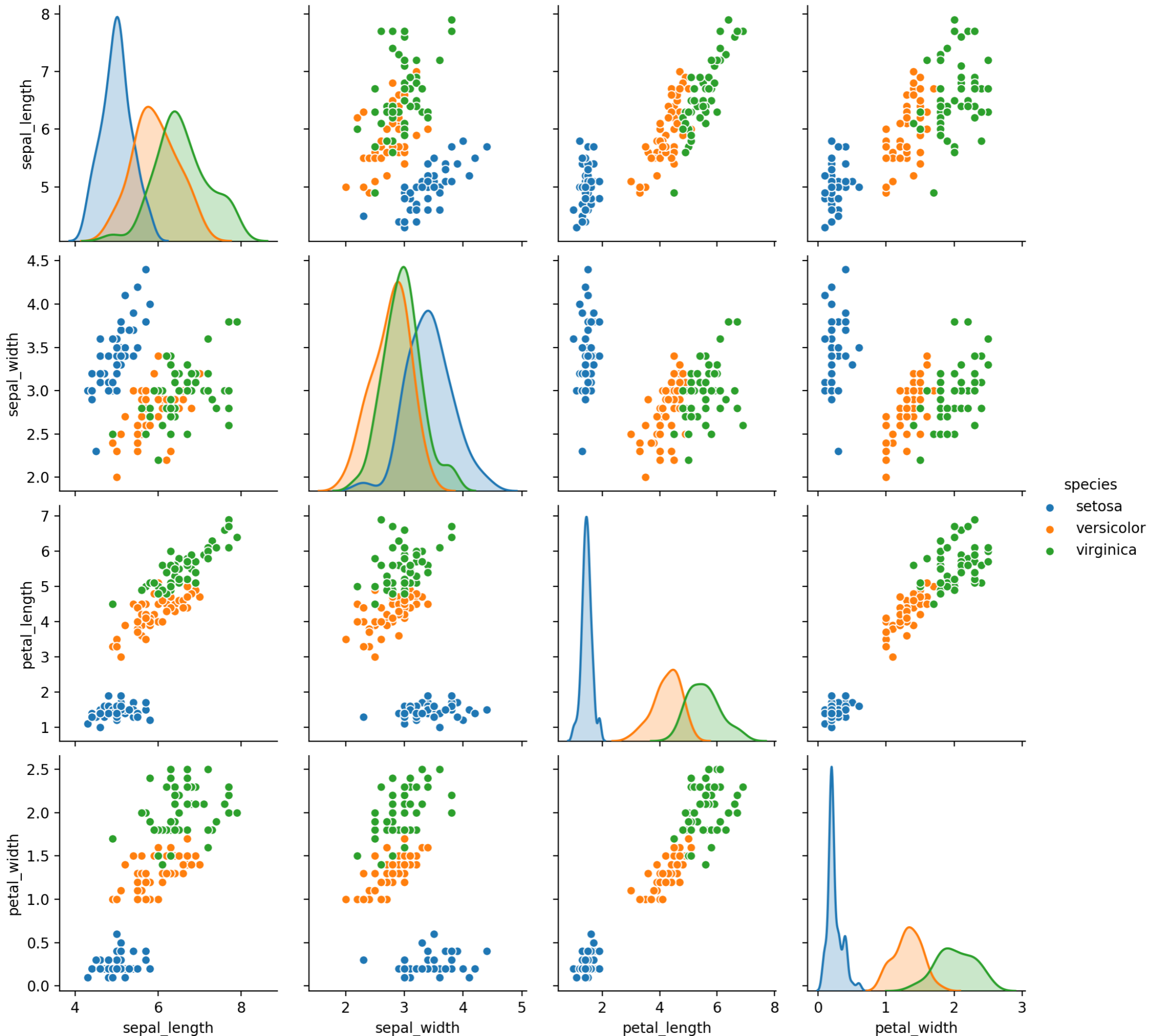
- `sns.jointplot(df.x, df.y, kind='kde')`



Seaborn

- Seaborn's jointplot even works for higher dimensions
 - When it is called `pairplot`
 - ```
iris = sns.load_dataset('iris')

sns.pairplot(iris, hue='species', height=2.5)
```



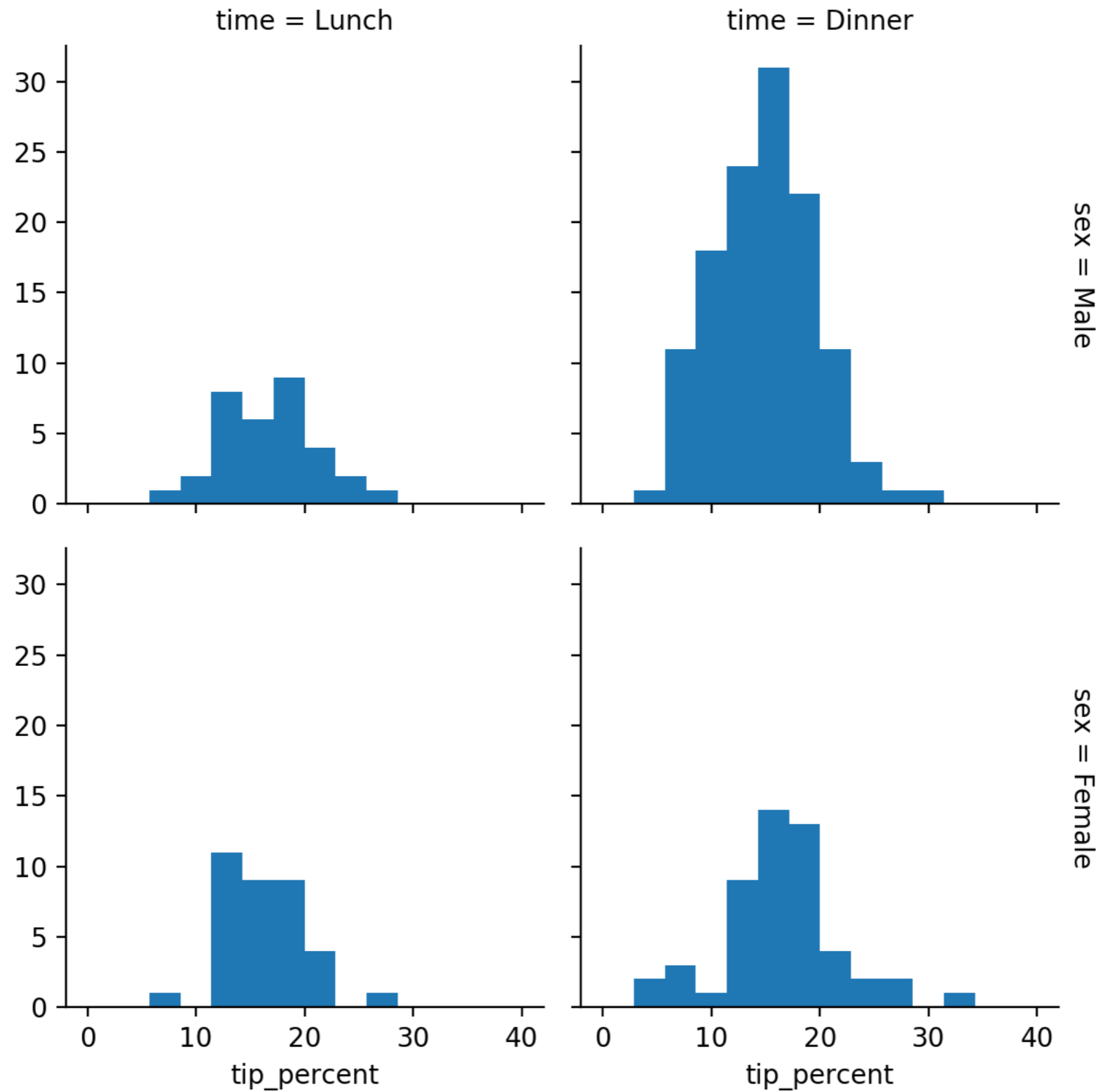


# Seaborn

- Seaborn allows us to show various factors

```
tips = sns.load_dataset('tips')
tips['tip_percent'] = 100*tips.tip/tips.total_bill
grid = sns.FacetGrid(tips,
 row='sex',
 col='time',
 margin_titles=True)
grid.map(plt.hist, 'tip_percent',
bins=np.linspace(0,40,15))
```

# Seaborn

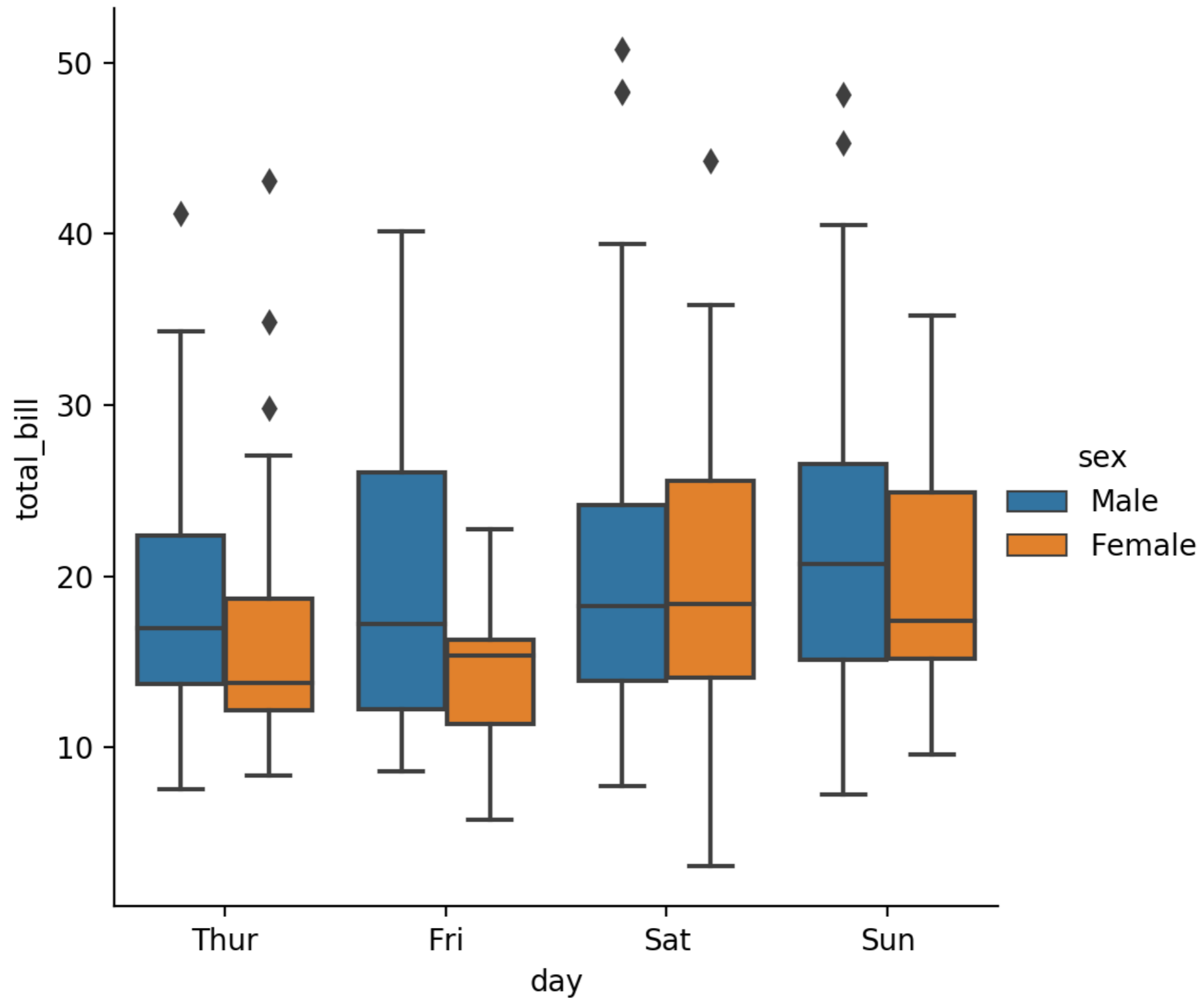


# Seaborn

- Can use category plot to show dependencies

```
tips['tip_percent'] = 100*tips.tip/tips.total_bill
g = sns.catplot('day', 'total_bill', 'sex', data=tips,
kind='box')
```

# Seaborn



# Tasks

- Import the penguin data set
  - Get rid of:
    - Rows with NA values
    - The row with the bad sex
- Use seaborn
- Plot body mass versus culmen-length for all penguins

# Tasks

- Use the seaborn pairplot on the penguins

# Tasks

- Use the seaborn category plot for body-mass based on sex and species

# Tasks

- Test whether the body mass for the penguins in dependence on sex and species are normally distributed



# Tasks

- Use scipy stats to check whether the means are different based on sex and species.

# Tasks

|        | Electric | Mechanical | Other |
|--------|----------|------------|-------|
| Line 1 | 13       | 33         | 15    |
| Line 2 | 7        | 4          | 11    |
| Line 3 | 18       | 10         | 14    |

- Can we say that failure behavior differs between lines.