

# Lecture 8: Plotting and Visualization

Alpar Sultan, PhD, Associate professor

# matplotlib

```
In [11]: import matplotlib.pyplot as plt
```

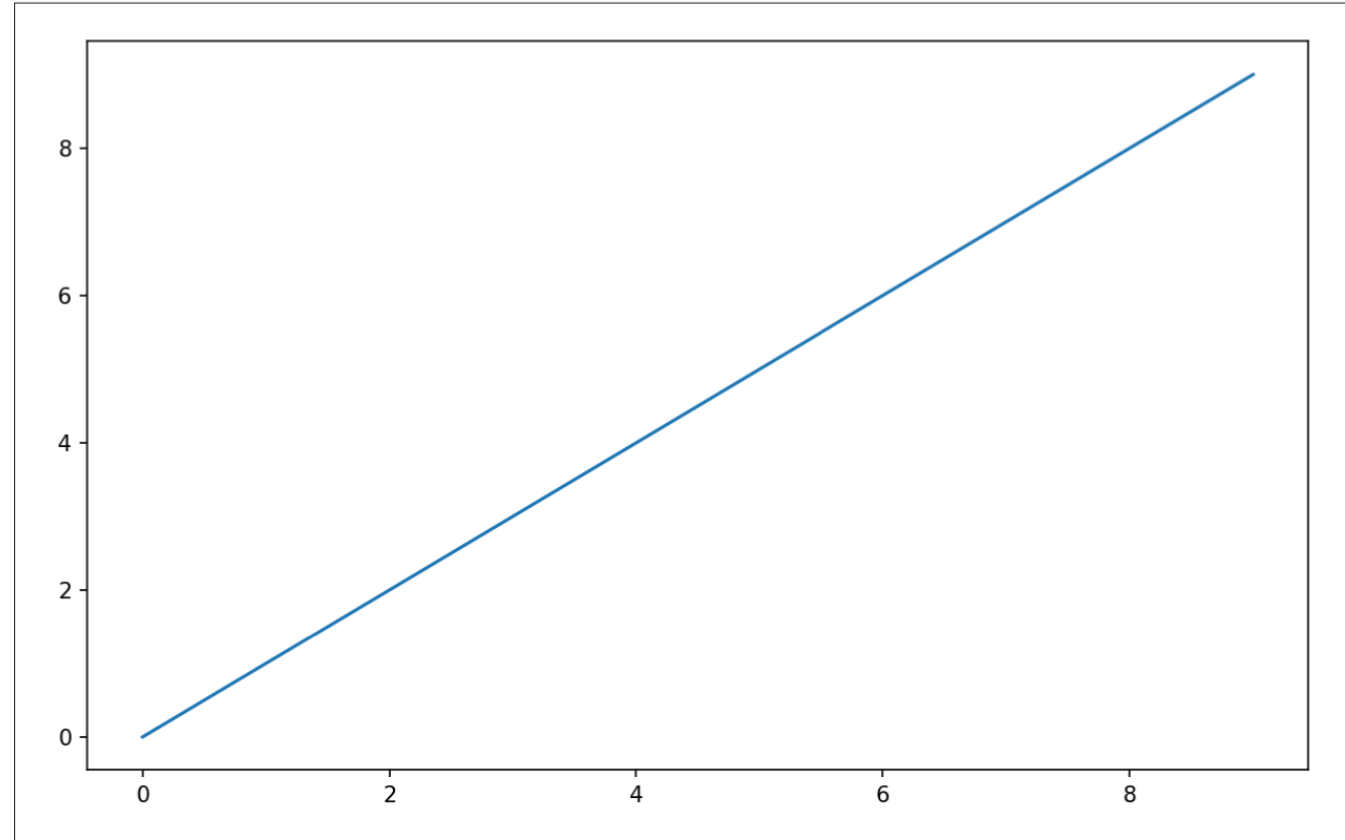
```
In [12]: import numpy as np
```

```
In [13]: data = np.arange(10)
```

```
In [14]: data
```

```
Out[14]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [15]: plt.plot(data)
```



*Figure 1: Simple line plot*

# Figures and Subplots

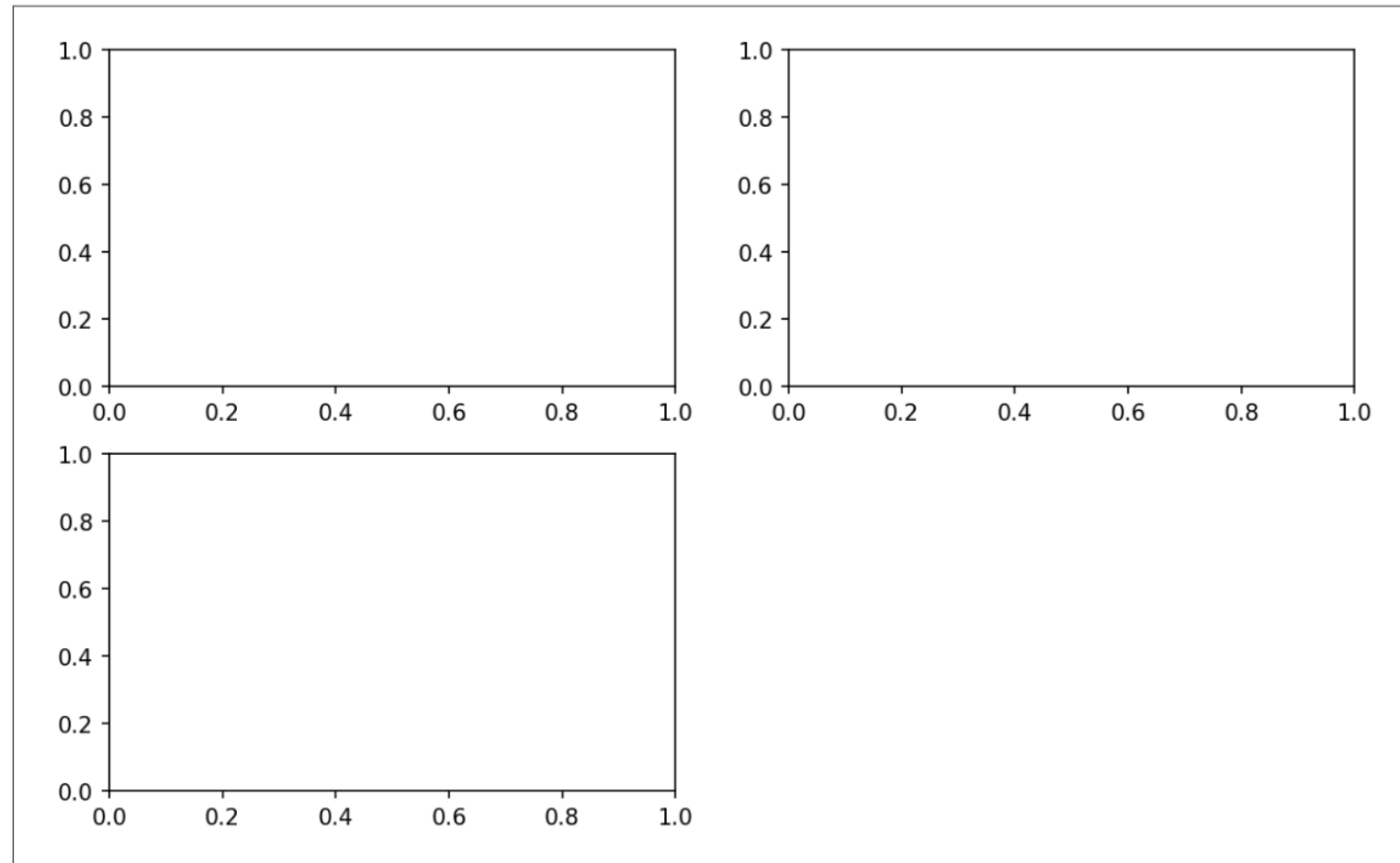
```
In [16]: fig = plt.figure()
```

```
In [17]: ax1 = fig.add_subplot(2, 2, 1)
```

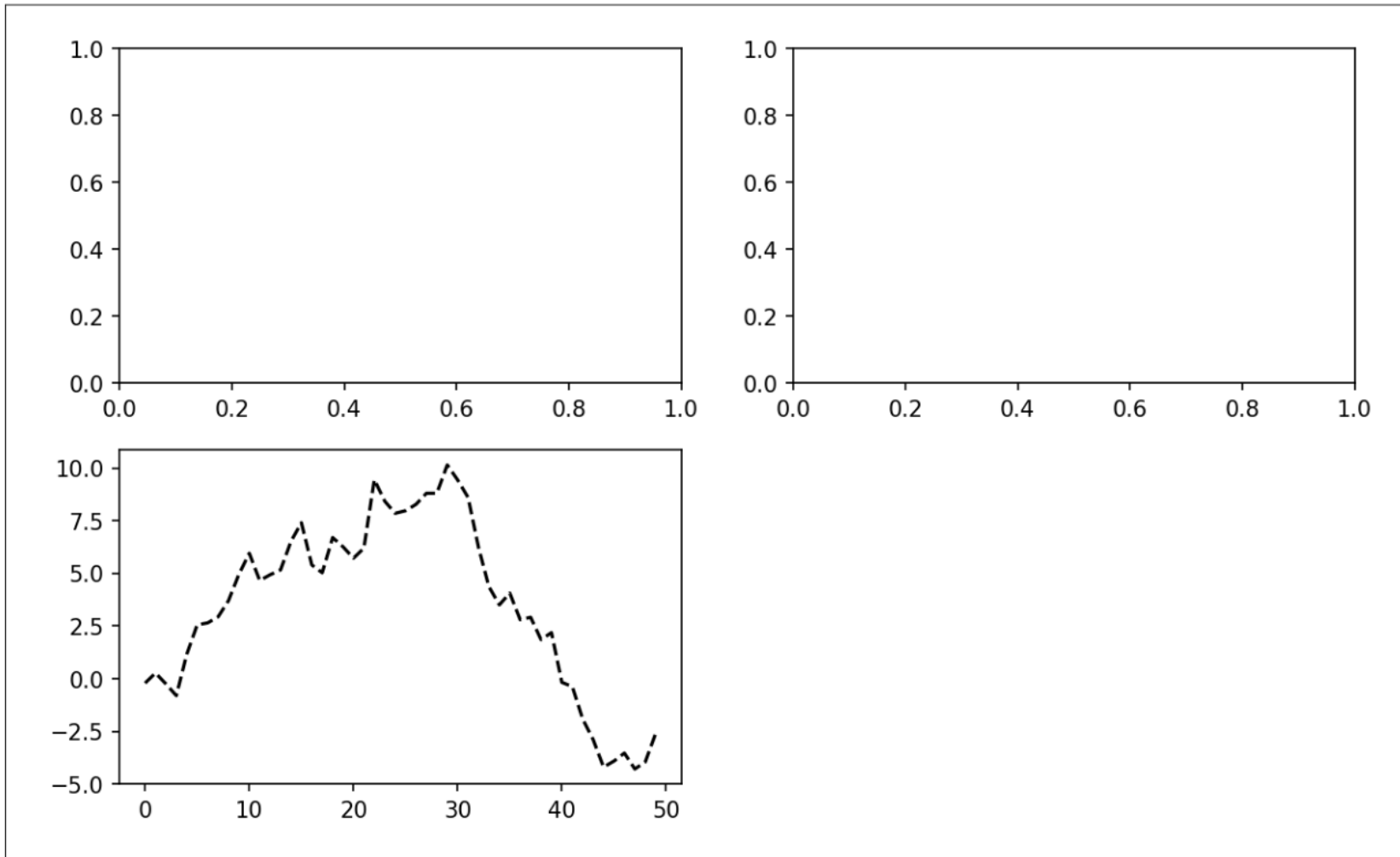
```
In [18]: ax2 = fig.add_subplot(2, 2, 2)
```

```
In [19]: ax3 = fig.add_subplot(2, 2, 3)
```

*Figure 2: An empty matplotlib figure with three subplots*



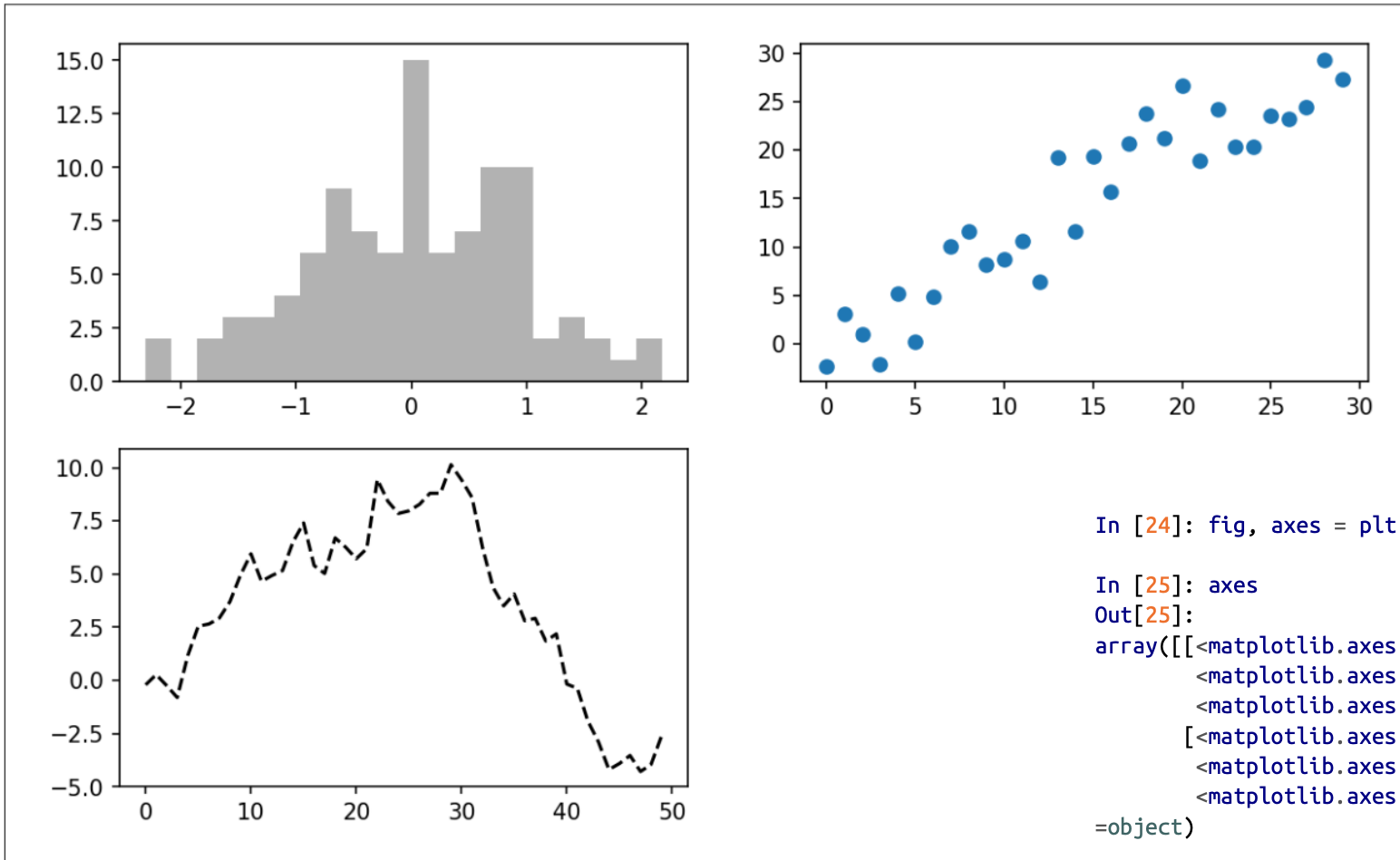
```
In [20]: plt.plot(np.random.randn(50).cumsum(), 'k--')
```



*Figure 3: Data visualization after single plot*

```
In [21]: _ = ax1.hist(np.random.randn(100), bins=20, color='k', alpha=0.3)
```

```
In [22]: ax2.scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))
```



```
In [24]: fig, axes = plt.subplots(2, 3)
```

```
In [25]: axes
```

```
Out[25]:
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fb626374048>,  
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fb62625db00>,  
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fb6262f6c88>],  
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fb6261a36a0>,  
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fb626181860>,  
       <matplotlib.axes._subplots.AxesSubplot object at 0x7fb6260fd4e0>]], dtype  
=object)
```

Figure 4: Data visualization after additional plots

# Pyplot.subplots options

| Argument                | Description                                                                                                                             |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <code>nrows</code>      | Number of rows of subplots                                                                                                              |
| <code>ncols</code>      | Number of columns of subplots                                                                                                           |
| <code>sharex</code>     | All subplots should use the same x-axis ticks (adjusting the <code>xlim</code> will affect all subplots)                                |
| <code>sharey</code>     | All subplots should use the same y-axis ticks (adjusting the <code>ylim</code> will affect all subplots)                                |
| <code>subplot_kw</code> | Dict of keywords passed to <code>add_subplot</code> call used to create each subplot                                                    |
| <code>**fig_kw</code>   | Additional keywords to <code>subplots</code> are used when creating the figure, such as <code>plt.subplots(2, 2, figsize=(8, 6))</code> |

---

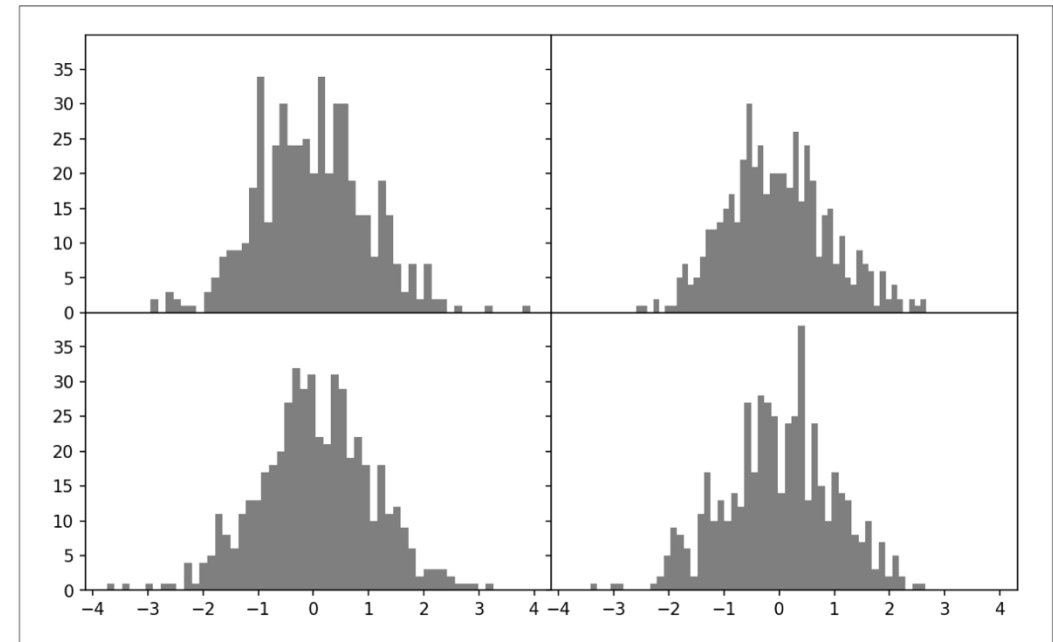
# Adjusting the spacing around subplots

```
subplots_adjust(left=None, bottom=None, right=None, top=None,  
               wspace=None, hspace=None)
```

wspace and hspace controls the percent of the figure width and figure height, respectively, to use as spacing between subplots

```
fig, axes = plt.subplots(2, 2, sharex=True, sharey=True)  
for i in range(2):  
    for j in range(2):  
        axes[i, j].hist(np.random.randn(500), bins=50, color='k', alpha=0.5)  
plt.subplots_adjust(wspace=0, hspace=0)
```

*Figure 5: Data visualization with no inter-subplot spacing*



# Colors, Markers and Line Styles

```
ax.plot(x, y, 'g--')
```

```
ax.plot(x, y, linestyle='--', color='g')
```

```
In [30]: from numpy.random import randn
```

```
In [31]: plt.plot(randn(30).cumsum(), 'ko--')
```

This could also have been written more explicitly as:

```
plot(randn(30).cumsum(), color='k', linestyle='dashed', marker='o')
```

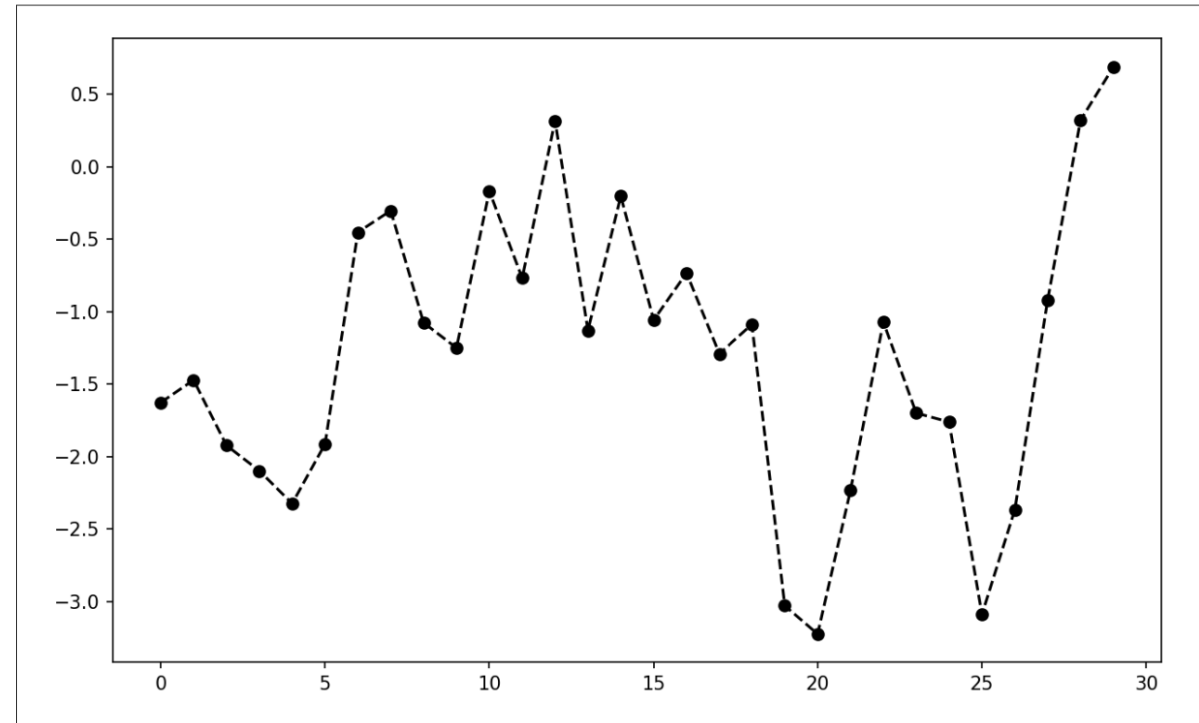


Figure 6: Line plot with markers



- For line plots, you will notice that subsequent points are linearly interpolated by default

```
In [33]: data = np.random.randn(30).cumsum()
```

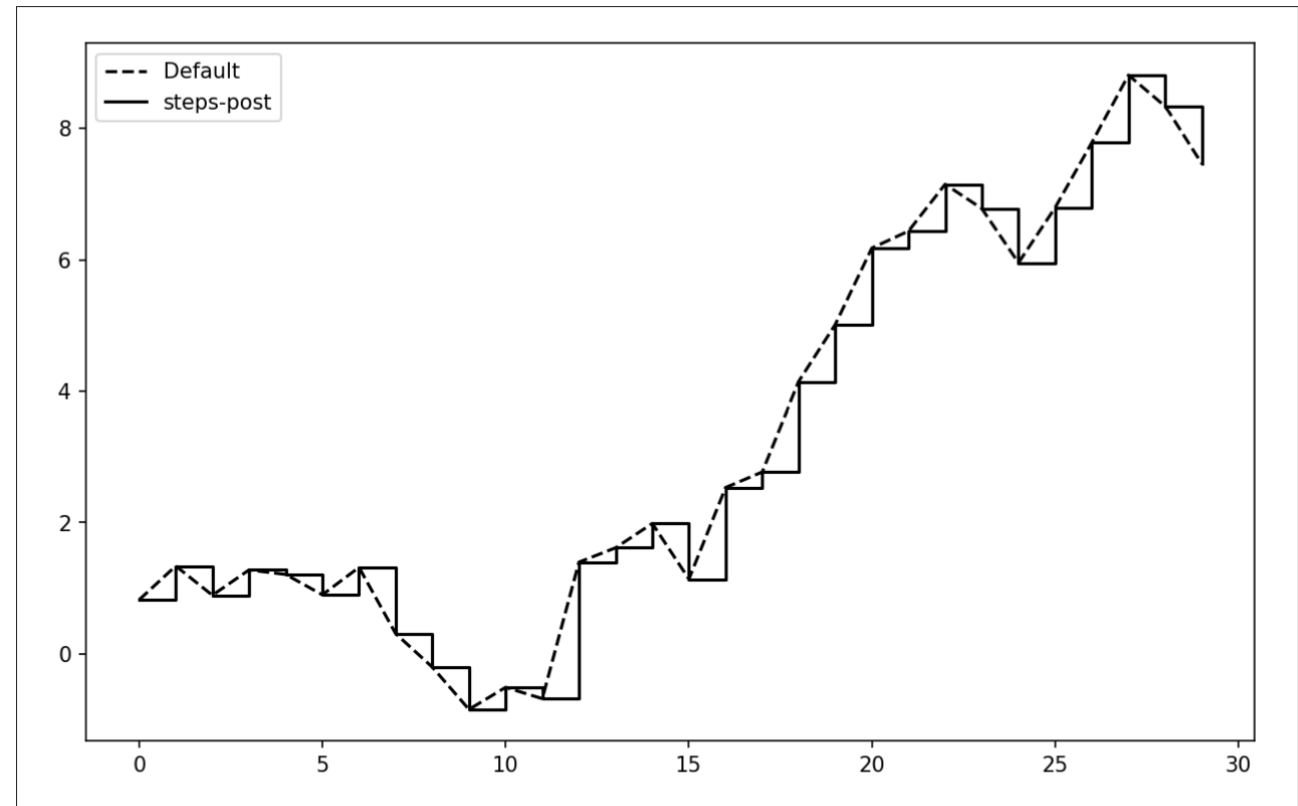
```
In [34]: plt.plot(data, 'k--', label='Default')
```

```
Out[34]: [<matplotlib.lines.Line2D at 0x7fb624d86160>]
```

```
In [35]: plt.plot(data, 'k-', drawstyle='steps-post', label='steps-post')
```

```
Out[35]: [<matplotlib.lines.Line2D at 0x7fb624d869e8>]
```

```
In [36]: plt.legend(loc='best')
```



*Figure 7: Line plot with different drawstyle options*

# Ticks, Labels, and Legends

- The pyplot interface, designed for interactive use, consists of methods like `xlim`, `xticks`, and `xticklabels`. These control the plot range, tick locations, and tick labels, respectively. They can be used in two ways:
  - Called with no arguments returns the current parameter value (e.g., `plt.xlim()` returns the current x-axis plotting range)
  - Called with parameters sets the parameter value (e.g., `plt.xlim([0, 10])`, sets the x-axis range to 0 to 10)

```
In [37]: fig = plt.figure()
```

```
In [38]: ax = fig.add_subplot(1, 1, 1)
```

```
In [39]: ax.plot(np.random.randn(1000).cumsum())
```

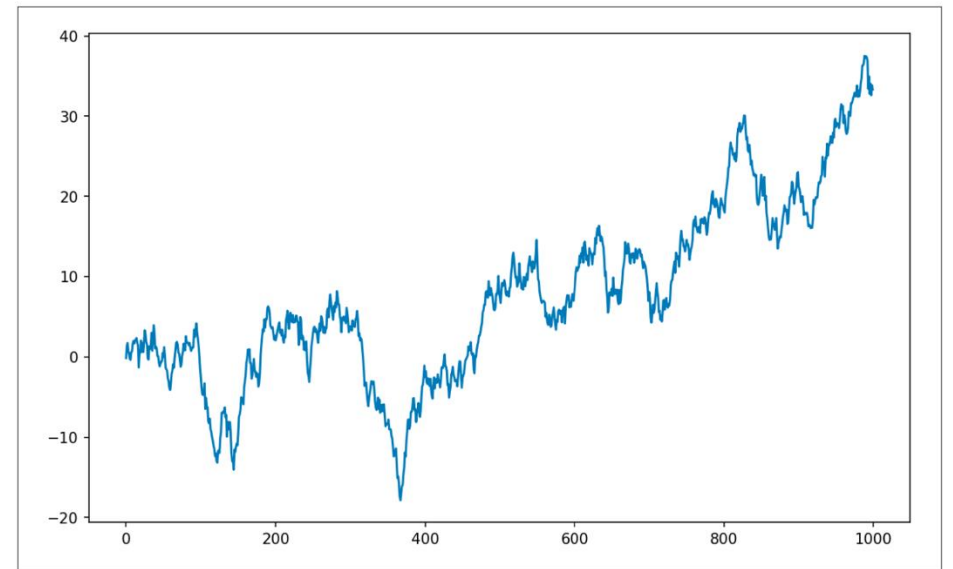


Figure 8: Simple plot for illustrating `xticks` (with label)

```
In [40]: ticks = ax.set_xticks([0, 250, 500, 750, 1000])
```

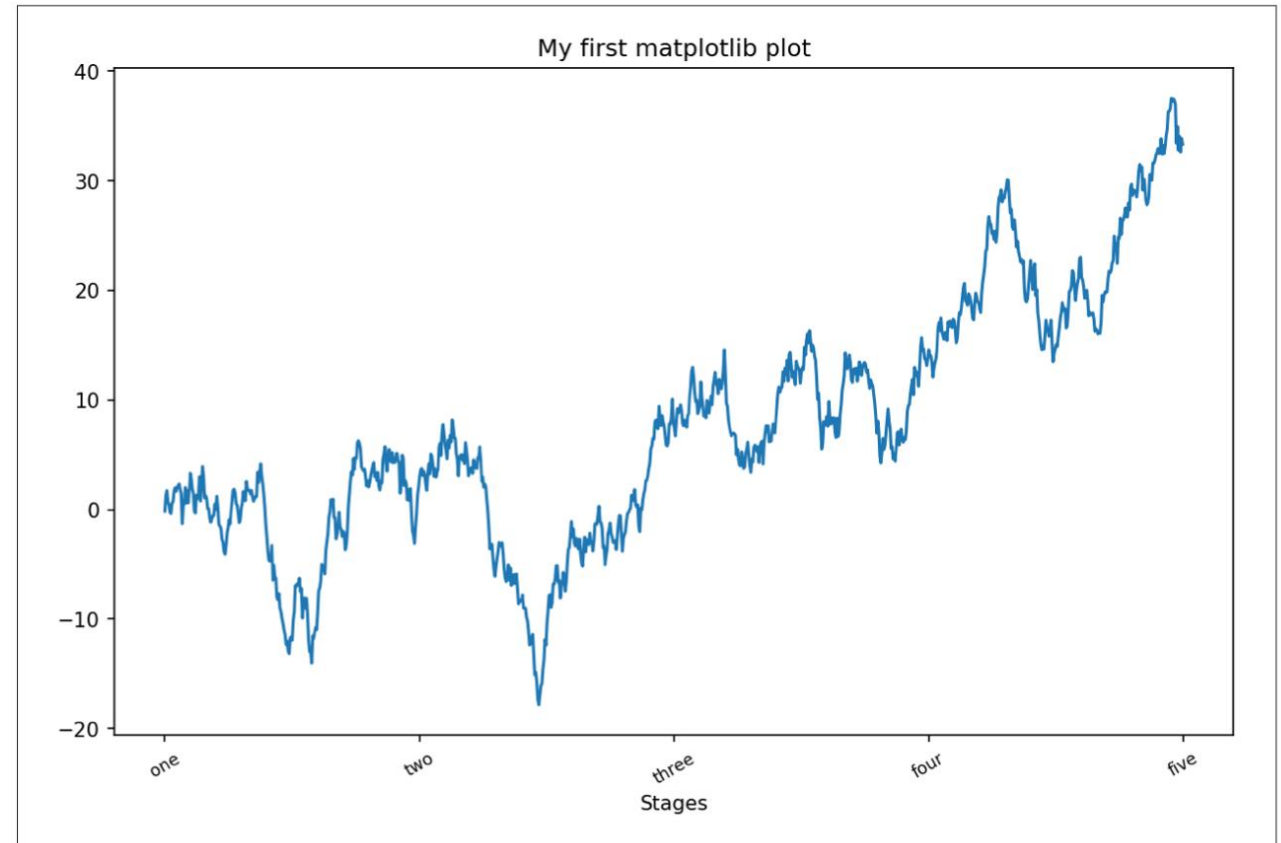
```
In [41]: labels = ax.set_xticklabels(['one', 'two', 'three', 'four', 'five'],  
.....:                             rotation=30, fontsize='small')
```

The rotation option sets the x tick labels at a 30-degree rotation. Lastly, `set_xlabel` gives a name to the x-axis and `set_title` the subplot title

```
In [42]: ax.set_title('My first matplotlib plot')
```

```
Out[42]: <matplotlib.text.Text at 0x7fb624d055f8>
```

```
In [43]: ax.set_xlabel('Stages')
```



*Figure 9: Simple plot for illustrating xticks*

# Adding legends

```
In [44]: from numpy.random import randn
```

```
In [45]: fig = plt.figure(); ax = fig.add_subplot(1, 1, 1)
```

```
In [46]: ax.plot(randn(1000).cumsum(), 'k', label='one')
```

```
Out[46]: [<matplotlib.lines.Line2D at 0x7fb624bdf860>]
```

```
In [47]: ax.plot(randn(1000).cumsum(), 'k--', label='two')
```

```
Out[47]: [<matplotlib.lines.Line2D at 0x7fb624be90f0>]
```

```
In [48]: ax.plot(randn(1000).cumsum(), 'k.', label='three')
```

```
Out[48]: [<matplotlib.lines.Line2D at 0x7fb624be9160>]
```

Once you've done this, you can either call `ax.legend()` or `plt.legend()` to automatically create a legend

```
In [49]: ax.legend(loc='best')
```

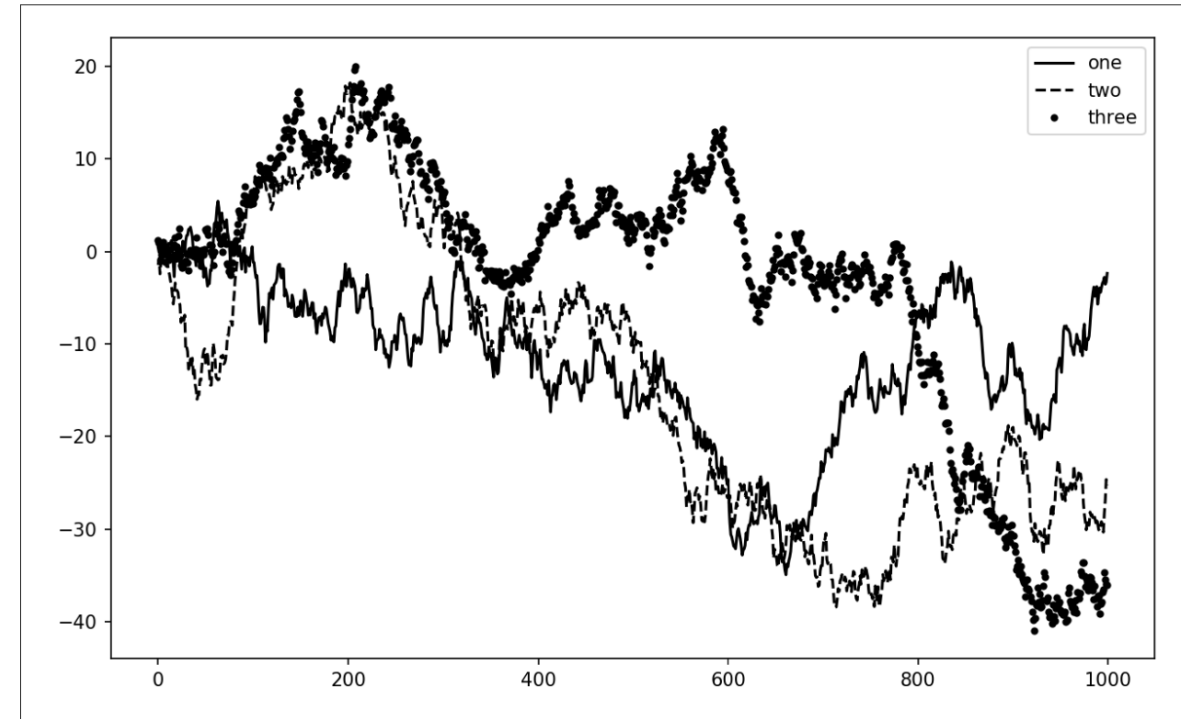


Figure 10: Simple plot with three lines and legend

# Saving Plots to a File

```
plt.savefig('figpath.svg')
```

| Argument                                          | Description                                                                                                                                                                     |
|---------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>fname</code>                                | String containing a filepath or a Python file-like object. The figure format is inferred from the file extension (e.g., <code>.pdf</code> for PDF or <code>.png</code> for PNG) |
| <code>dpi</code>                                  | The figure resolution in dots per inch; defaults to 100 out of the box but can be configured                                                                                    |
| <code>facecolor,</code><br><code>edgecolor</code> | The color of the figure background outside of the subplots; <code>'w'</code> (white), by default                                                                                |
| <code>format</code>                               | The explicit file format to use ( <code>'png'</code> , <code>'pdf'</code> , <code>'svg'</code> , <code>'ps'</code> , <code>'eps'</code> , ...)                                  |
| <code>bbox_inches</code>                          | The portion of the figure to save; if <code>'tight'</code> is passed, will attempt to trim the empty space around the figure                                                    |

---

# Plotting with pandas and seaborn

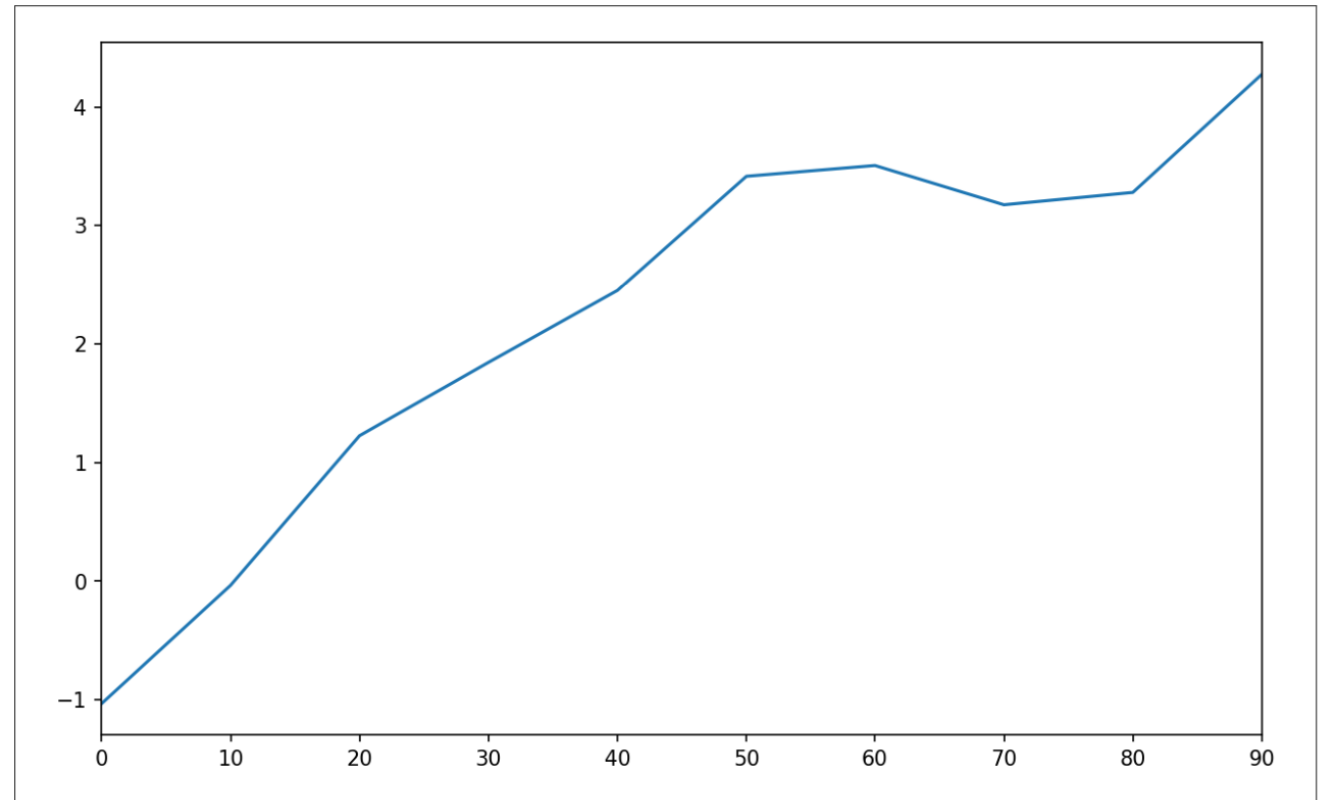
- **Line Plots**

- Series and DataFrame each have a plot attribute for making some basic plot types. By default, plot() makes line plots

```
In [60]: s = pd.Series(np.random.randn(10).cumsum(), index=np.arange(0, 100, 10))
```

```
In [61]: s.plot()
```

The Series object's index is passed to matplotlib for plotting on the x-axis, though you can disable this by passing `use_index=False`



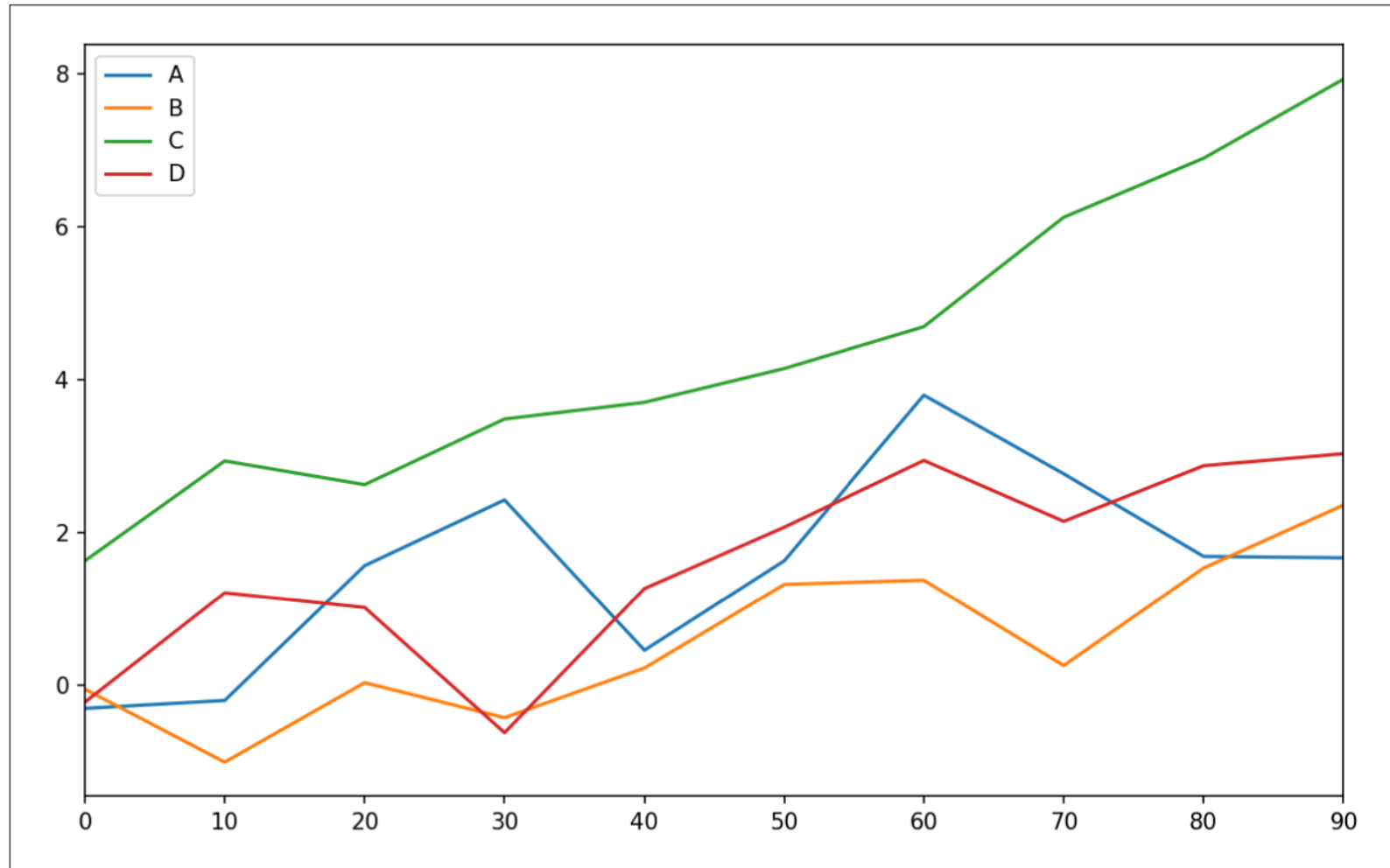
*Figure 11: Simple Series plot*

- DataFrame's plot method plots each of its columns as a different line on the same subplot, creating a legend automatically

```
In [62]: df = pd.DataFrame(np.random.randn(10, 4).cumsum(0),  
.....:                    columns=['A', 'B', 'C', 'D'],  
.....:                    index=np.arange(0, 100, 10))
```

```
In [63]: df.plot()
```

Figure 12: Simple DataFrame plot



# Series.plot method arguments

| Argument  | Description                                                                             |
|-----------|-----------------------------------------------------------------------------------------|
| label     | Label for plot legend                                                                   |
| ax        | matplotlib subplot object to plot on; if nothing passed, uses active matplotlib subplot |
| style     | Style string, like 'ko--', to be passed to matplotlib                                   |
| alpha     | The plot fill opacity (from 0 to 1)                                                     |
| kind      | Can be 'area', 'bar', 'barh', 'density', 'hist', 'kde', 'line', 'pie'                   |
| logy      | Use logarithmic scaling on the y-axis                                                   |
| use_index | Use the object index for tick labels                                                    |
| rot       | Rotation of tick labels (0 through 360)                                                 |
| xticks    | Values to use for x-axis ticks                                                          |
| yticks    | Values to use for y-axis ticks                                                          |
| xlim      | x-axis limits (e.g., [0, 10])                                                           |
| ylim      | y-axis limits                                                                           |
| grid      | Display axis grid (on by default)                                                       |

---



# DataFrame-specific plot arguments

| Argument                  | Description                                                                     |
|---------------------------|---------------------------------------------------------------------------------|
| <code>subplots</code>     | Plot each DataFrame column in a separate subplot                                |
| <code>sharex</code>       | If <code>subplots=True</code> , share the same x-axis, linking ticks and limits |
| <code>sharey</code>       | If <code>subplots=True</code> , share the same y-axis                           |
| <code>figsize</code>      | Size of figure to create as tuple                                               |
| <code>title</code>        | Plot title as string                                                            |
| <code>legend</code>       | Add a subplot legend (True by default)                                          |
| <code>sort_columns</code> | Plot columns in alphabetical order; by default uses existing column order       |

---

# Bar Plots

```
In [64]: fig, axes = plt.subplots(2, 1)
```

```
In [65]: data = pd.Series(np.random.rand(16), index=list('abcdefghijklmnop'))
```

```
In [66]: data.plot.bar(ax=axes[0], color='k', alpha=0.7)
```

```
Out[66]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb62493d470>
```

```
In [67]: data.plot.barh(ax=axes[1], color='k', alpha=0.7)
```

The options `color='k'` and `alpha=0.7` set the color of the plots to black and use partial transparency on the filling.

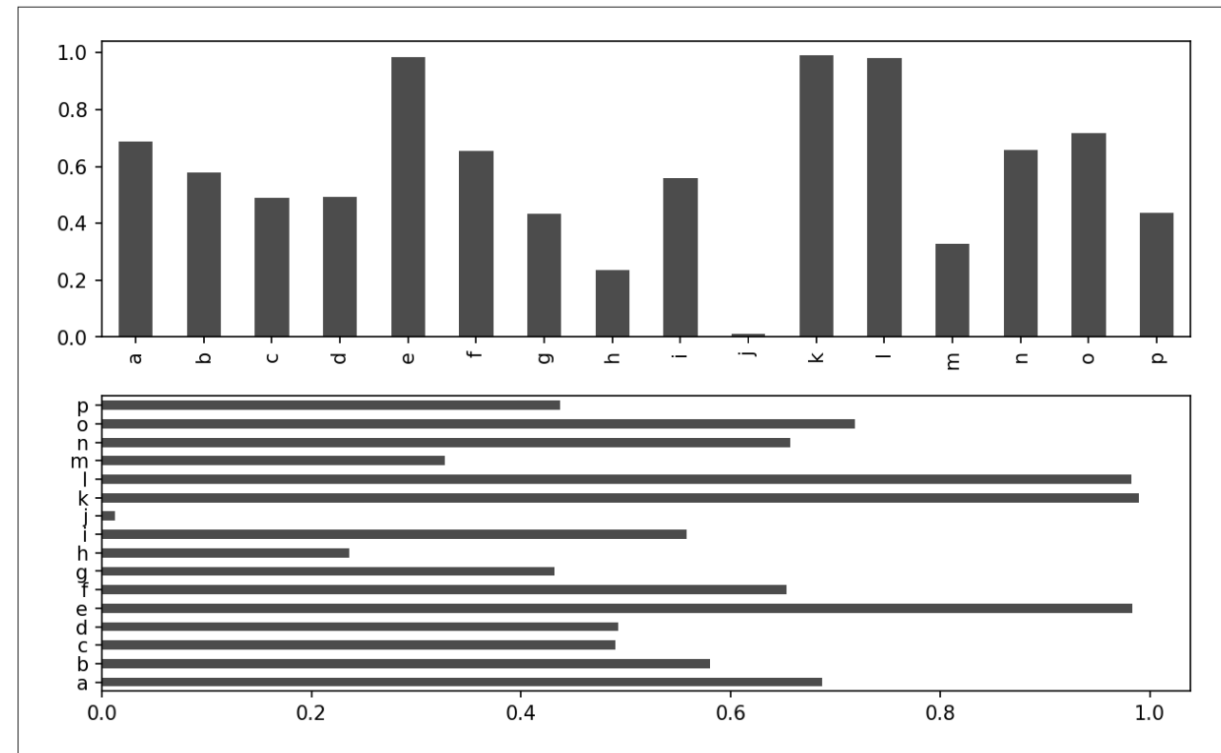


Figure 13: Horizontal and vertical bar plot

With a DataFrame, bar plots group the values in each row together in a group in bars, side by side, for each value

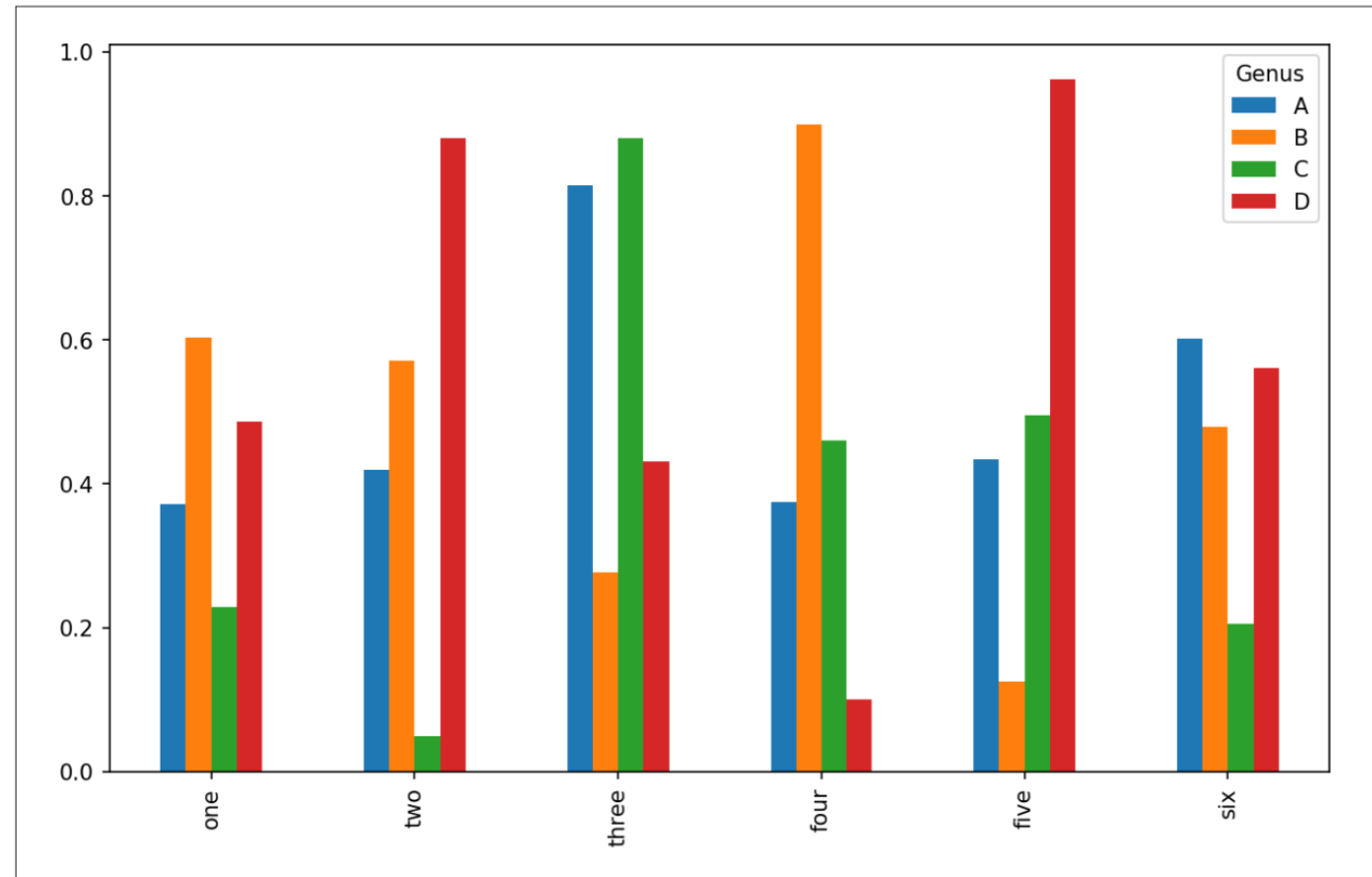
```
In [69]: df = pd.DataFrame(np.random.rand(6, 4),  
.....:                    index=['one', 'two', 'three', 'four', 'five', 'six'],  
.....:                    columns=pd.Index(['A', 'B', 'C', 'D'], name='Genus'))
```

```
In [70]: df
```

```
Out[70]:
```

| Genus | A        | B        | C        | D        |
|-------|----------|----------|----------|----------|
| one   | 0.370670 | 0.602792 | 0.229159 | 0.486744 |
| two   | 0.420082 | 0.571653 | 0.049024 | 0.880592 |
| three | 0.814568 | 0.277160 | 0.880316 | 0.431326 |
| four  | 0.374020 | 0.899420 | 0.460304 | 0.100843 |
| five  | 0.433270 | 0.125107 | 0.494675 | 0.961825 |
| six   | 0.601648 | 0.478576 | 0.205690 | 0.560547 |

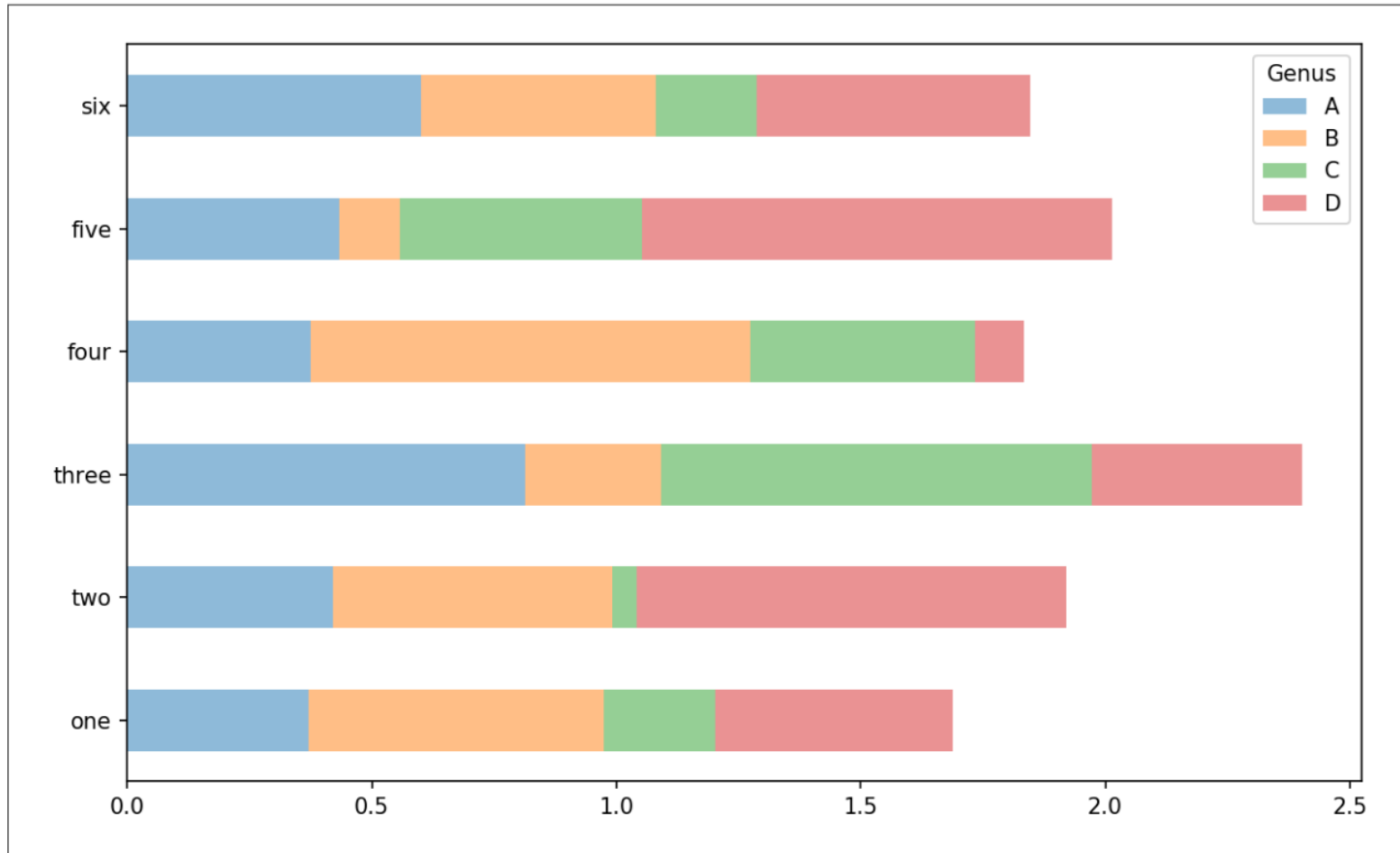
```
In [71]: df.plot.bar()
```



*Figure 14:  
DataFrame  
bar plot*

We create stacked bar plots from a DataFrame by passing `stacked=True`, resulting in the value in each row being stacked together

```
In [73]: df.plot.barh(stacked=True, alpha=0.5)
```



*Figure 15: DataFrame stacked bar plot*

# Histograms and Density Plots

```
In [92]: tips['tip_pct'].plot.hist(bins=50)
```

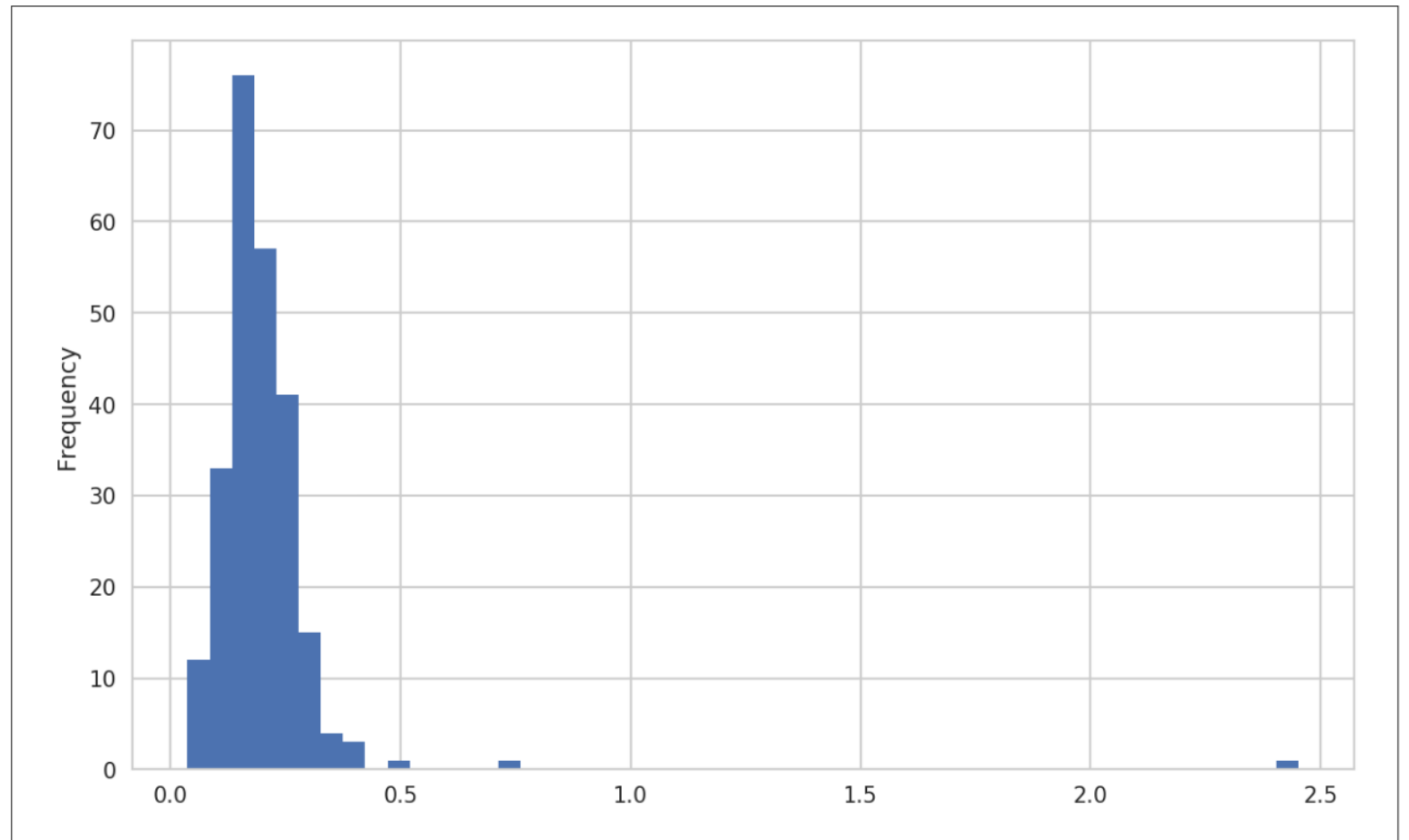
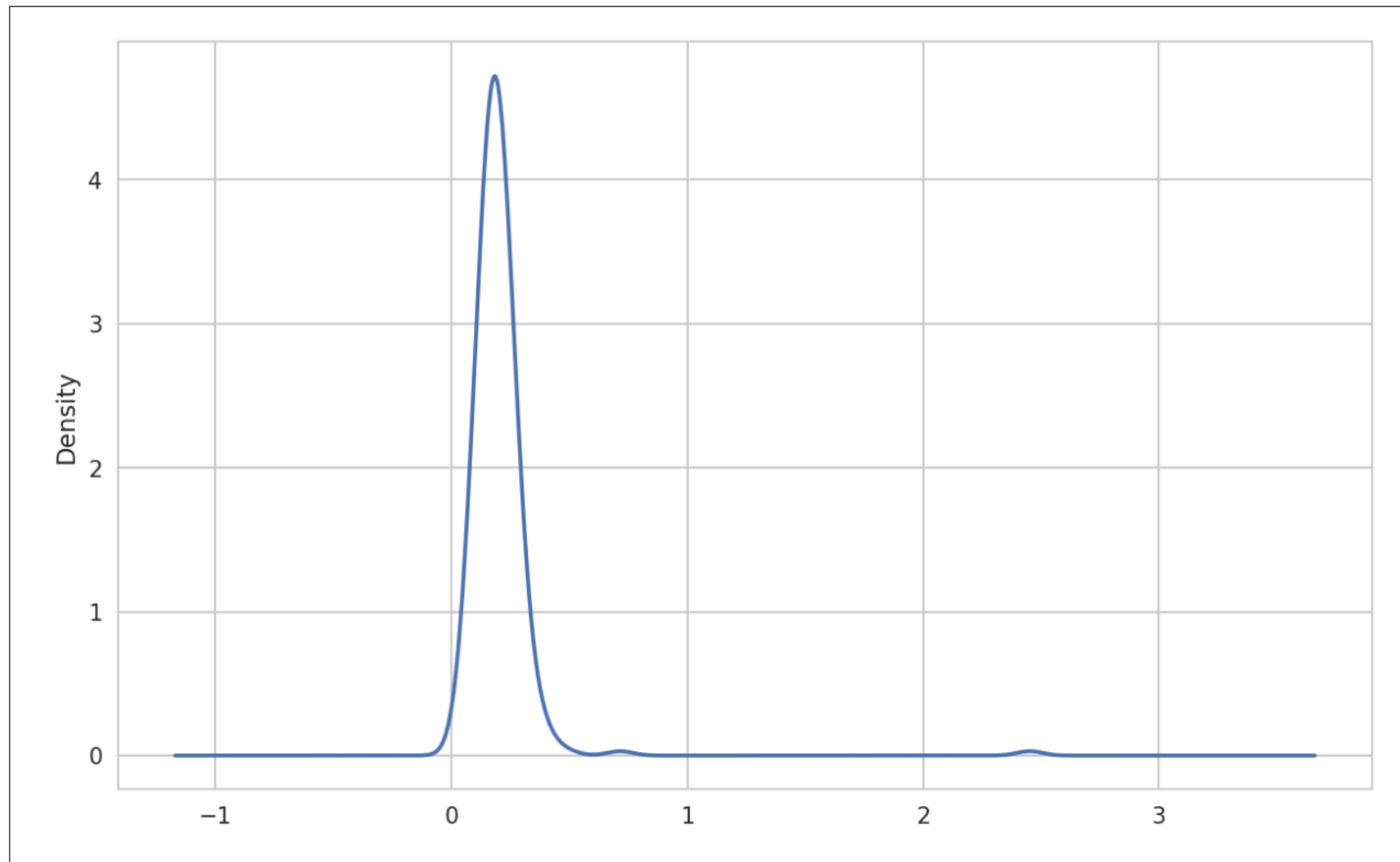


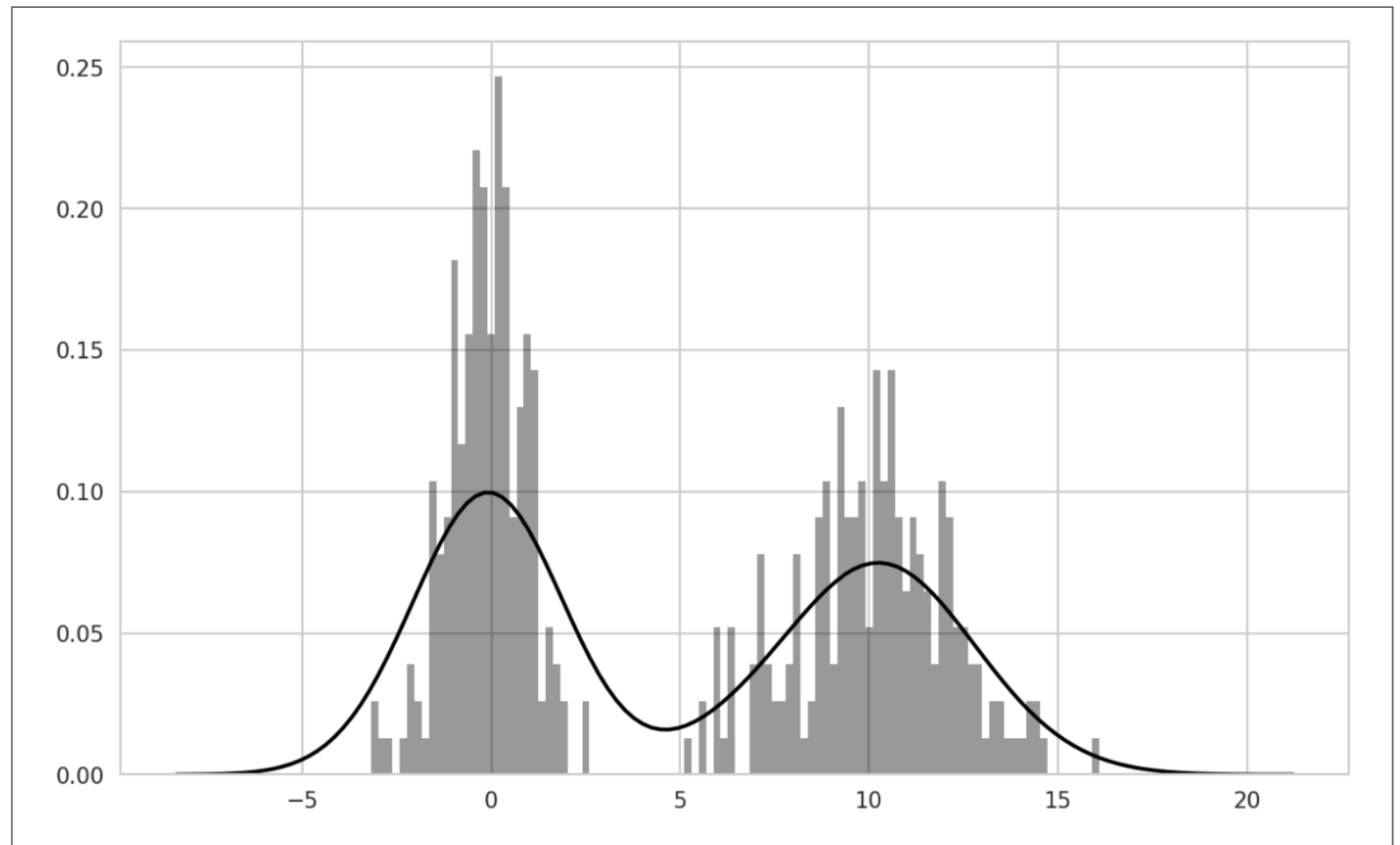
Figure 16: Histogram of tip percentages

```
In [94]: tips['tip_pct'].plot.density()
```



*Figure 17: Density plot of tip percentages*

```
In [96]: comp1 = np.random.normal(0, 1, size=200)
In [97]: comp2 = np.random.normal(10, 2, size=200)
In [98]: values = pd.Series(np.concatenate([comp1, comp2]))
In [99]: sns.distplot(values, bins=100, color='k')
```



*Figure 18: Normalized histogram of normal mixture with density estimate*

# Scatter or Point Plots

```
In [100]: macro = pd.read_csv('examples/macrodta.csv')
```

```
In [101]: data = macro[['cpi', 'm1', 'tbilrate', 'unemp']]
```

```
In [102]: trans_data = np.log(data).diff().dropna()
```

```
In [103]: trans_data[-5:]
```

```
Out[103]:
```

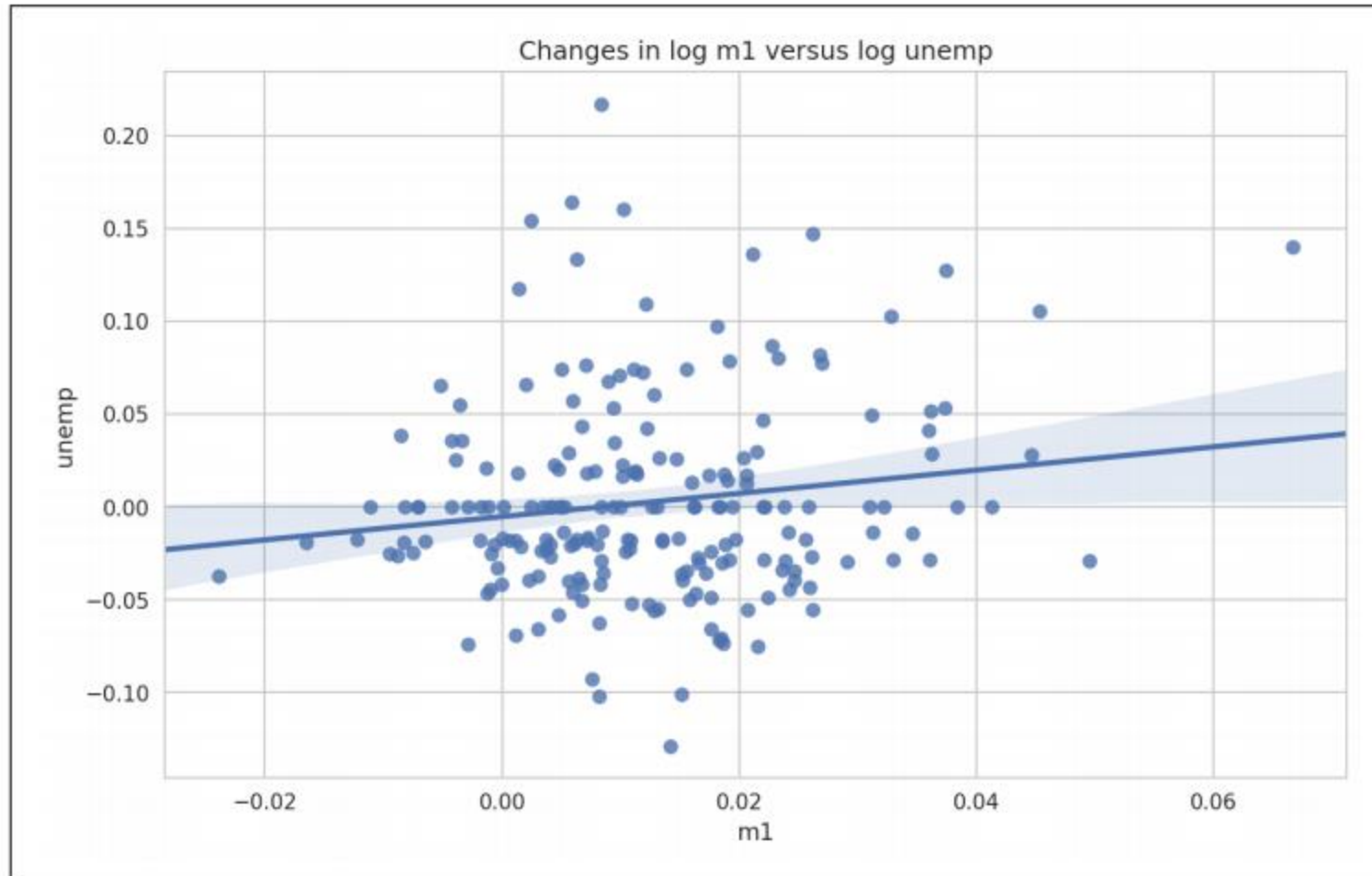
|     | cpi       | m1       | tbilrate  | unemp    |
|-----|-----------|----------|-----------|----------|
| 198 | -0.007904 | 0.045361 | -0.396881 | 0.105361 |
| 199 | -0.021979 | 0.066753 | -2.277267 | 0.139762 |
| 200 | 0.002340  | 0.010286 | 0.606136  | 0.160343 |
| 201 | 0.008419  | 0.037461 | -0.200671 | 0.127339 |
| 202 | 0.008894  | 0.012202 | -0.405465 | 0.042560 |



```
In [105]: sns.regplot('m1', 'unemp', data=trans_data)
```

```
Out[105]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb613720be0>
```

```
In [106]: plt.title('Changes in log %s versus log %s' % ('m1', 'unemp'))
```



*Figure 19: A seaborn regression/scatter plot*

```
In [107]: sns.pairplot(trans_data, diag_kind='kde', plot_kws={'alpha': 0.2})
```

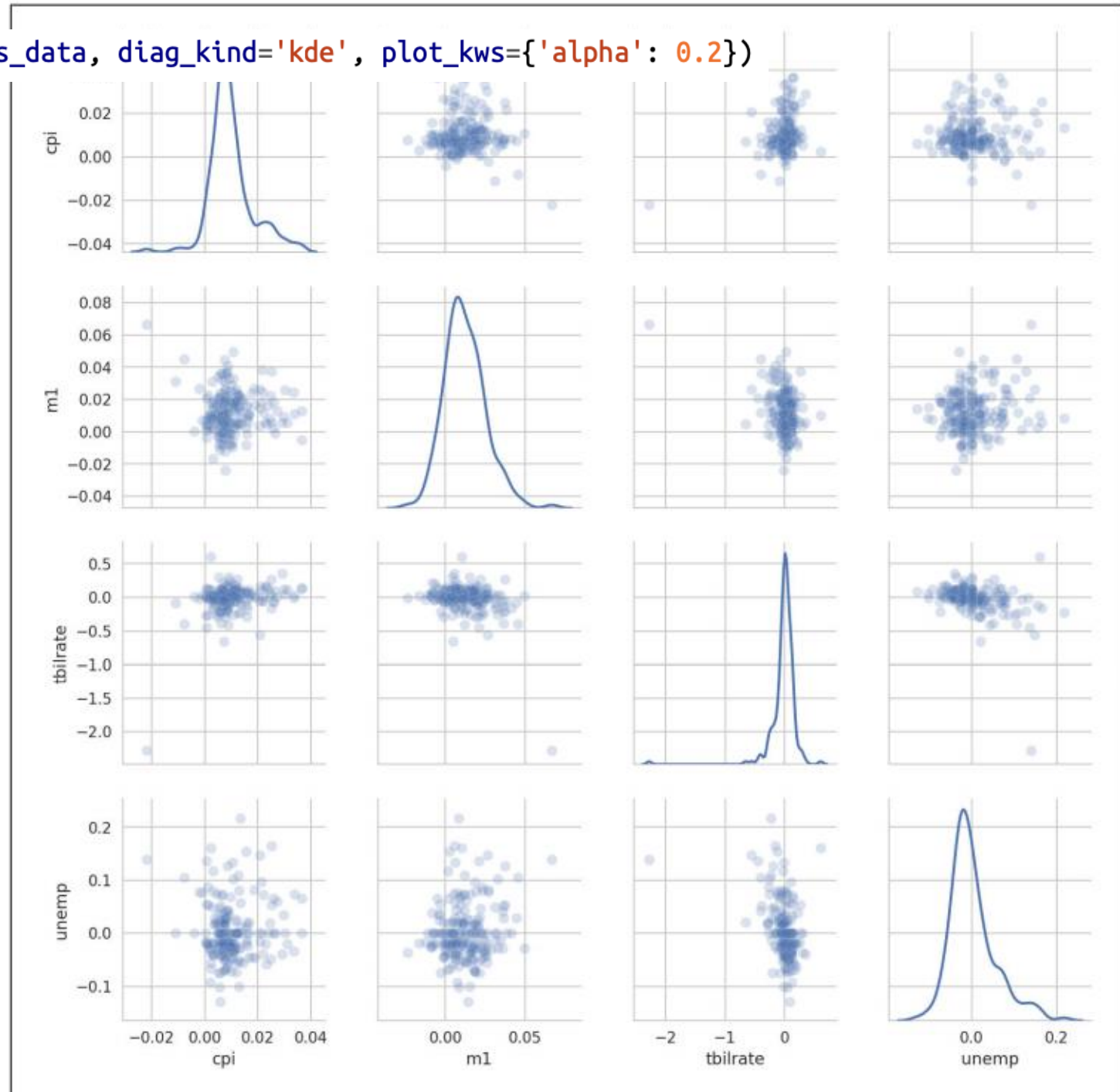


Figure 20: Pair plot matrix of statsmodels macro data

# Facet Grids and Categorical Data

```
In [108]: sns.factorplot(x='day', y='tip_pct', hue='time', col='smoker',  
.....:                  kind='bar', data=tips[tips.tip_pct < 1])
```

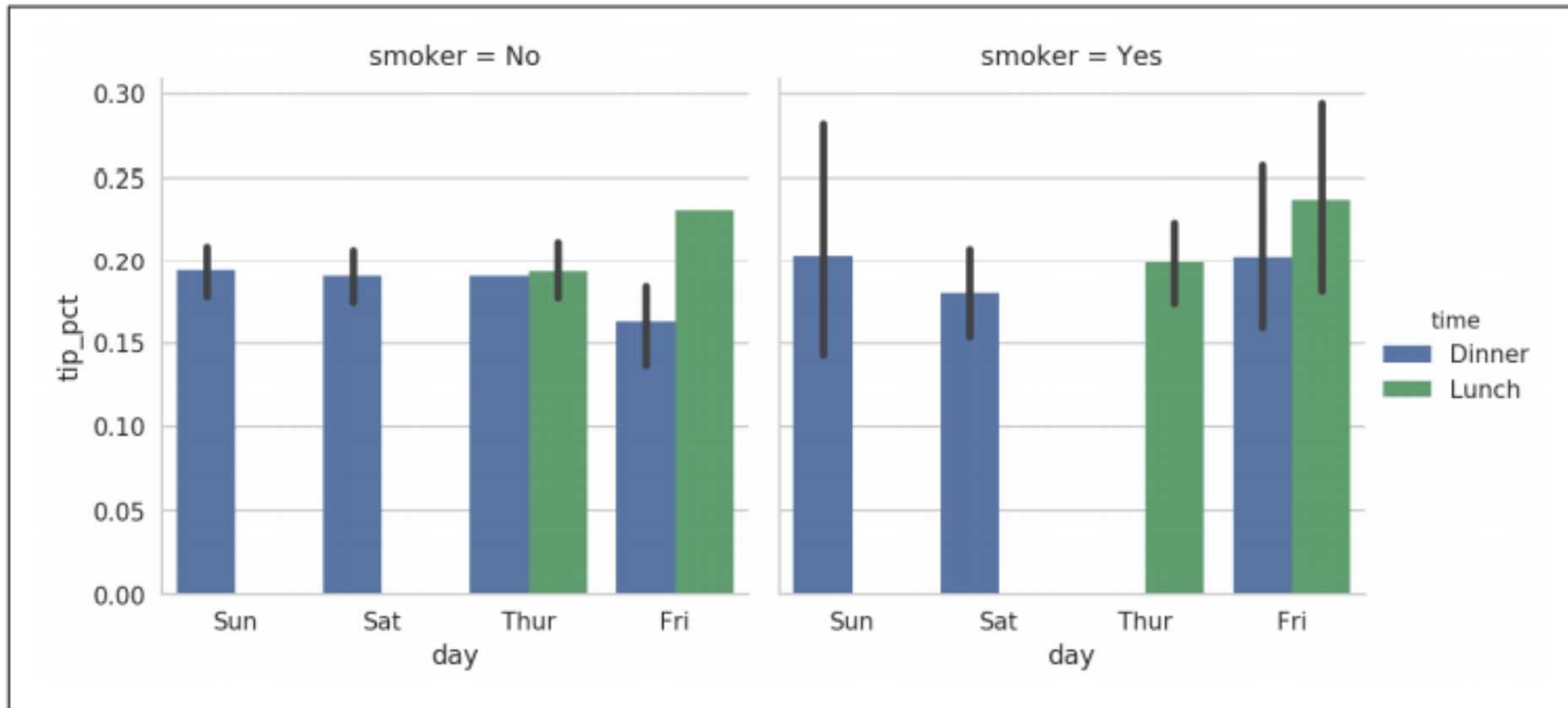


Figure 21: Tipping percentage by day/time/smoker

```
In [109]: sns.factorplot(x='day', y='tip_pct', row='time',  
.....:                 col='smoker',  
.....:                 kind='bar', data=tips[tips.tip_pct < 1])
```

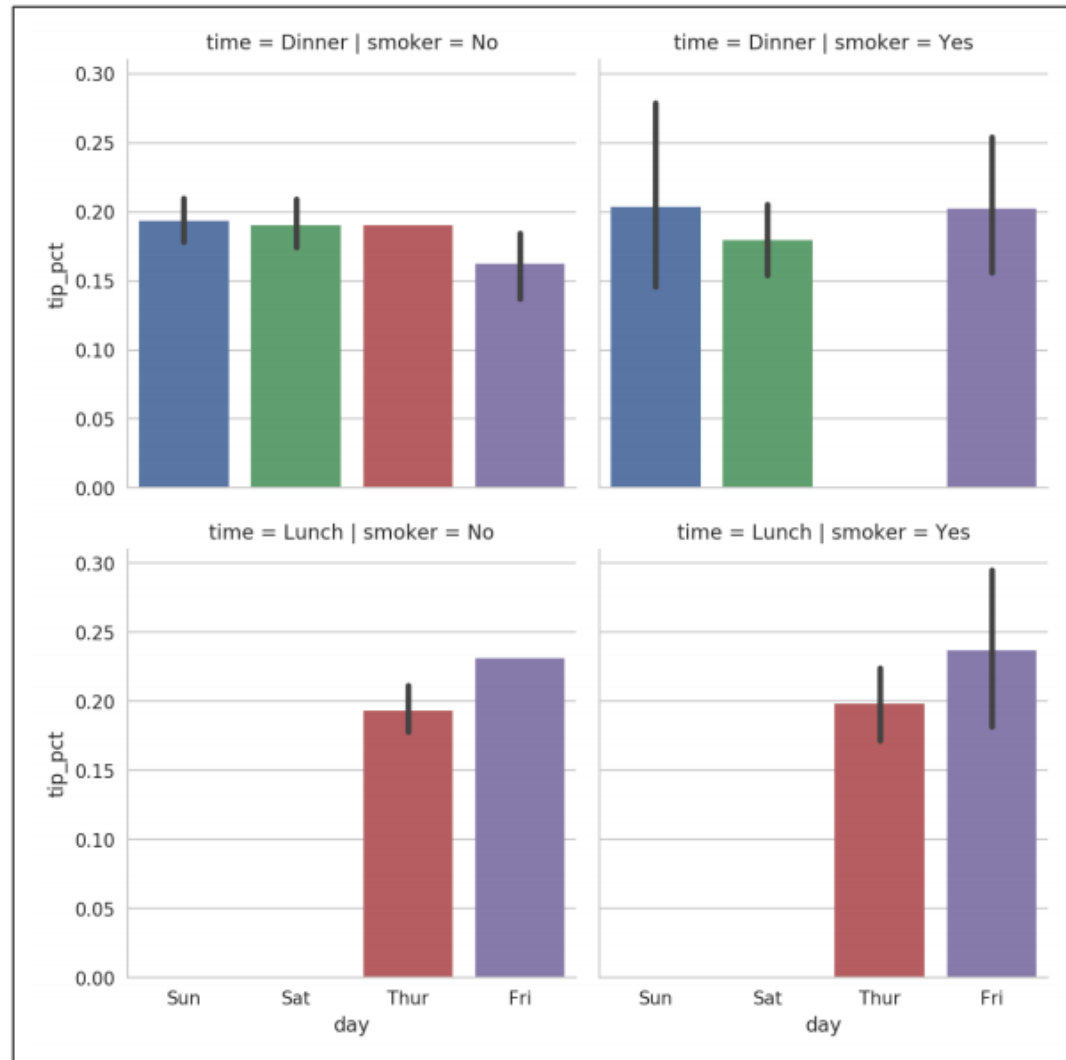


Figure 22: tip\_pct by day;  
facet by time/smoker

```
In [110]: sns.factorplot(x='tip_pct', y='day', kind='box',  
.....:                  data=tips[tips.tip_pct < 0.5])
```

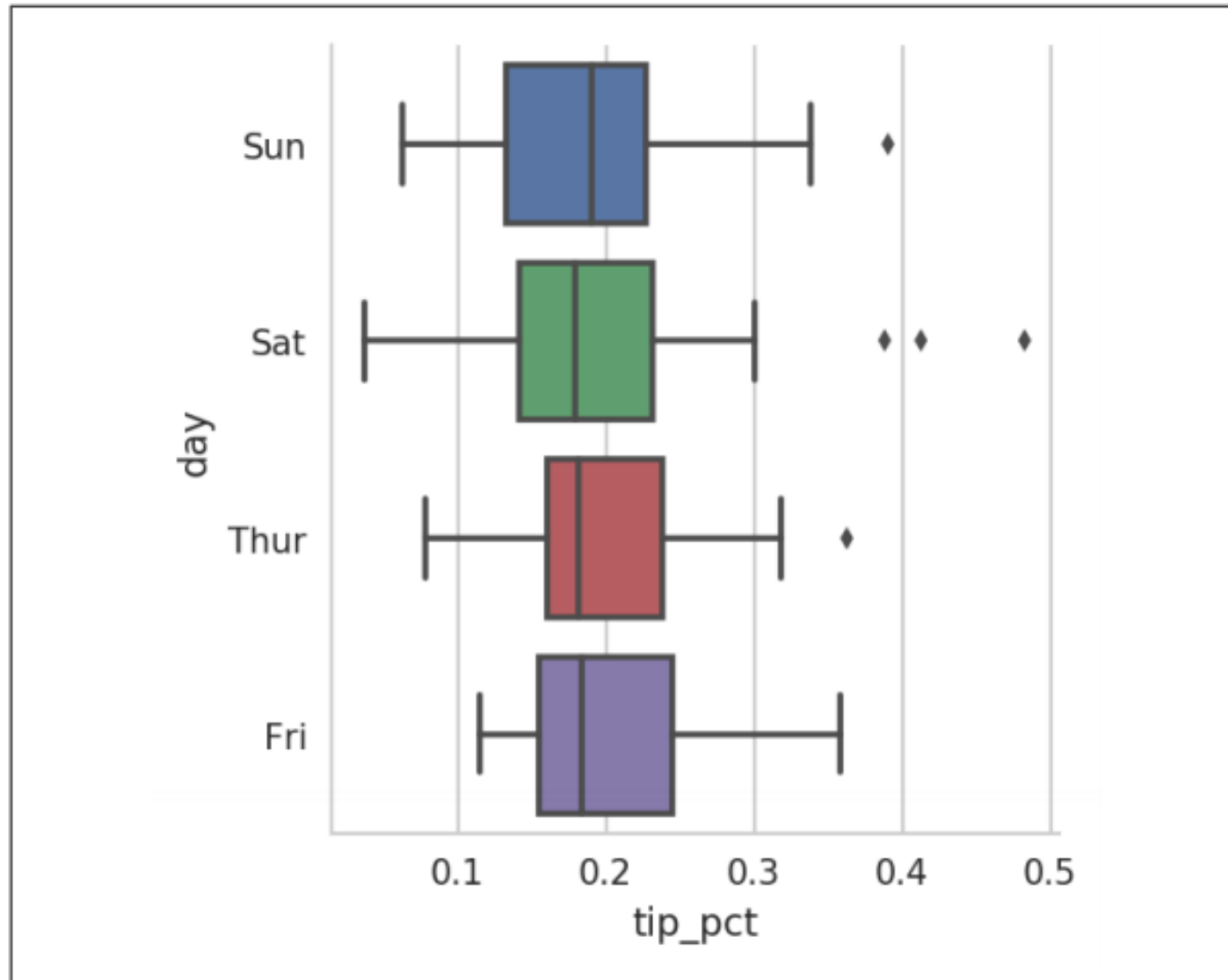


Figure 23: Box plot of tip\_pct by day

# Other Python Visualization Tools

- With tools like **Bokeh** and **Plotly**, it's now possible to specify dynamic, interactive graphics in Python that are destined for a web browser.