Plotting and Visualization

Part 3

A Brief matplotlib API Primer

Part 3

Annotations and Drawing on a Subplot

- In addition to the standard plot types, you may wish to draw your own plot annotations, which could consist of text, arrows, or other shapes.
- You can add annotations and text using the text, arrow, and annotate functions.
- `text` draws text at given coordinates `(x, y)` on the plot with optional custom styling:

```
ax.text(x, y, 'Hello world!',
        family='monospace', fontsize=10)
```

- Annotations can draw both text and arrows arranged appropriately.
- As an example, let’s plot the closing S&P 500 index price since 2007 (obtained from Yahoo! Finance) and annotate it with some of the important dates from the 2008–2009 financial crisis.

In [24]:

```
from datetime import datetime

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

data = pd.read_csv('examples/spx.csv', index_col=0, parse_dates=True)
spx = data['SPX']

spx.plot(ax=ax, style='k-')

crisis_data = [
    (datetime(2007, 10, 11), 'Peak of bull market'),
    (datetime(2008, 3, 12), 'Bear Stearns Fails'),
    (datetime(2008, 9, 15), 'Lehman Bankruptcy')
]

for date, label in crisis_data:
    ax.annotate(label, xy=(date, spx.asof(date) + 75),
                xytext=(date, spx.asof(date) + 225),
                arrowprops=dict(facecolor='black', headwidth=4, width=2,
                                headlength=4),
                horizontalalignment='left', verticalalignment='top')

# Zoom in on 2007-2010
ax.set_xlim(['1/1/2007', '1/1/2011'])
ax.set_ylim([600, 1800])

ax.set_title('Important dates in the 2008-2009 financial crisis')
```

Figure 9


Important dates in the 2008-2009 financial crisis
Peak of bull market
Bear Stearns Fails
Lehman Bankruptcy
1800
1600
1400
1200
1000
800
600
2007-01
2007-07
2008-01
2008-07
2009-01
2009-07
2010-01
2010-07
2011-01

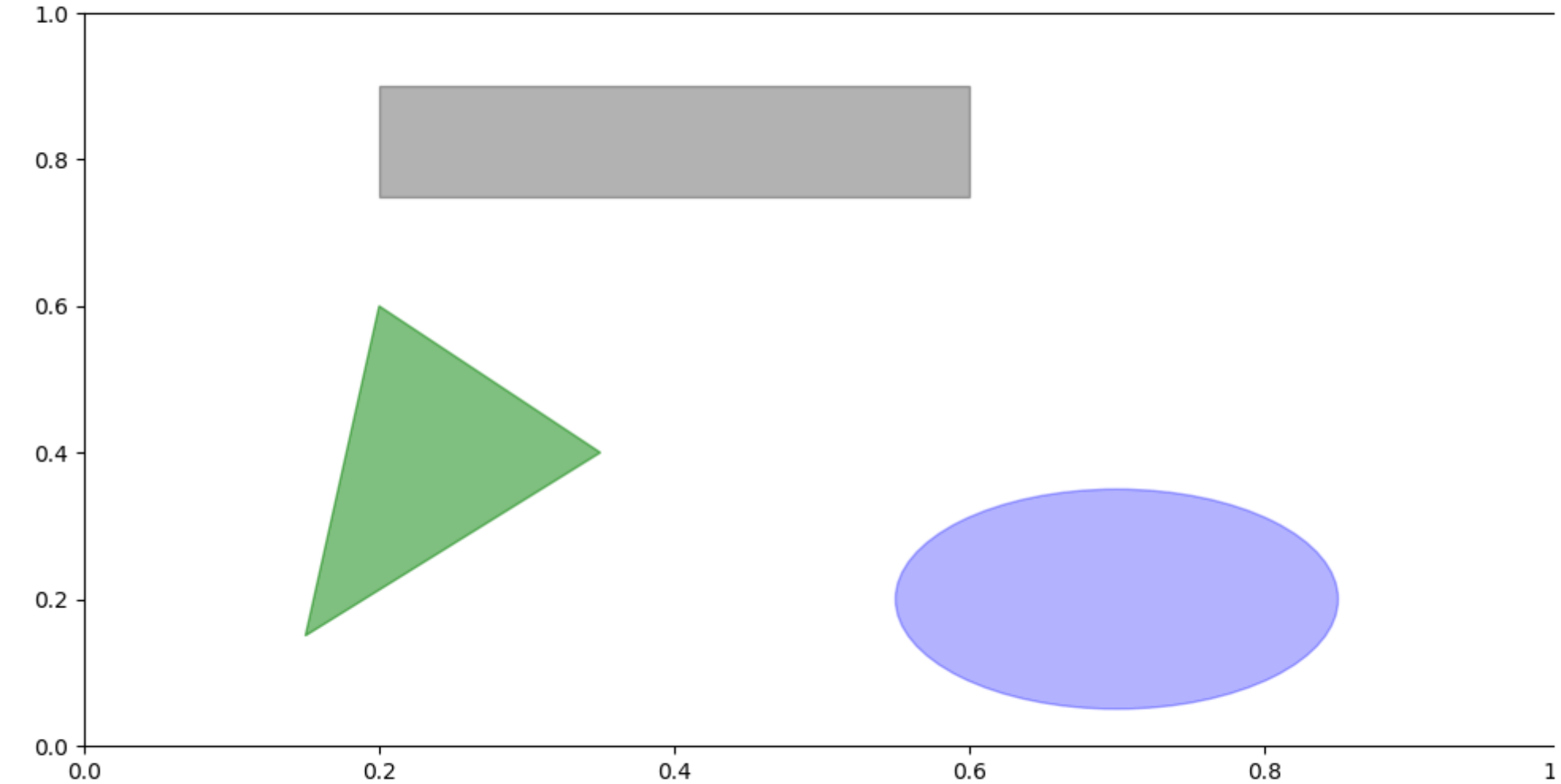
- See the online matplotlib gallery for many more annotation examples to learn from.

- Drawing shapes requires some more care.
- matplotlib has objects that represent many common shapes, referred to as patches.
- Some of these, like `Rectangle` and `Circle`, are found in `matplotlib.pyplot`, but the full set is located in `matplotlib.patches`.

- To add a shape to a plot, you create the patch object `shp` and add it to a subplot by calling `ax.add_patch(shp)`:

In [25]:
```
fig = plt.figure(figsize=(12, 6)); ax = fig.add_subplot(1, 1, 1)
rect = plt.Rectangle((0.2, 0.75), 0.4, 0.15, color='k', alpha=0.3)
circ = plt.Circle((0.7, 0.2), 0.15, color='b', alpha=0.3)
pgon = plt.Polygon([[0.15, 0.15], [0.35, 0.4], [0.2, 0.6]],
                   color='g', alpha=0.5)
ax.add_patch(rect)
ax.add_patch(circ)
ax.add_patch(pgon)
```

Figure 10



x=0.2

Saving Plots to File

- You can save the active figure to file using `plt.savefig`.
- This method is equivalent to the figure object's `savefig` instance method.
- For example, to save an SVG version of a figure, you need only type:

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

- The file type is inferred from the file extension.
- So if you used `.pdf` instead, you would get a PDF.

- There are a couple of important options that are used frequently for publishing graphics: `dpi`, which controls the dots-per-inch resolution, and `bbox_inches`, which can trim the whitespace around the actual figure.
- To get the same plot as a PNG with minimal whitespace around the plot and at 400 DPI, you would do:

```
plt.savefig('figpath.png', dpi=400, bbox_inches='tight')
```

- `savefig` doesn't have to write to disk; it can also write to any file-like object, such as a `BytesIO`:

```
from io import BytesIO
buffer = BytesIO()
plt.savefig(buffer)
plot_data = buffer.getvalue()
```

matplotlib Configuration

- matplotlib comes configured with color schemes and defaults that are geared primarily toward preparing figures for publication.
- Fortunately, nearly all of the default behavior can be customized via an extensive set of global parameters governing figure size, subplot spacing, colors, font sizes, grid styles, and so on.
- One way to modify the configuration programmatically from Python is to use the `rc` method; for example, to set the global default figure size to be 10 × 10, you could enter:

```
plt.rc('figure', figsize=(10, 10))
```

- An easy way to write down the options in your program is as a dict:

```
font_options = {'family' : 'monospace',
                'weight' : 'bold',
                'size'   : 'small'}
plt.rc('font', **font_options)
```

- For more extensive customization and to see a list of all the options, matplotlib comes with a configuration file *matplotlibrc* in the *matplotlib/mpl-data* directory.
- If you customize this file and place it in your home directory titled *.matplotlibrc*, it will be loaded each time you use matplotlib.