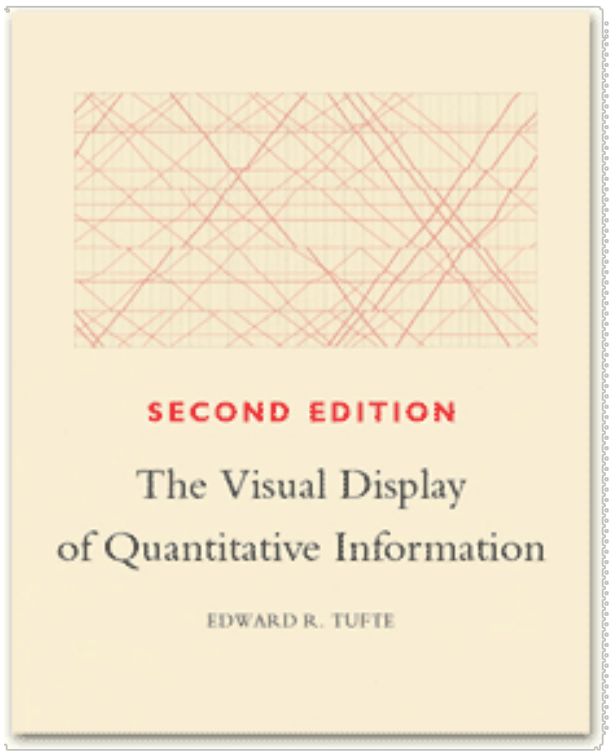
Data Visualization in Python

Georgia Tech

Data Visualization



> *Data graphics visually display measured quantities by means of the combined use of points, lines, a coordinate system, numbers, words, shading, and color.*

– Edward Tufte, *The Visual Display of Quantitative Information*

Georgia Tech

Data Visualization in Python

You already know basic concepts of visualization, and there are many courses that go in depth. Here we'll learn how to manipulate the data and parameters of the visualizations available in the SciPy stack.

- Matplotlib
  - Matlab-like plotting interface
  - The granddaddy of all scientific plotting in Python
  - Powerful, low-level
  - Built on NumPy arrays
- Seaborn
  - Higher-level API on top of Matplotlib
  - Integrates with Pandas DataFrames
- Bokeh or Plotly/Dash
  - Interactive visualizations like D3

Georgia Tech

Matplotlib

Standard import:

```
import matplotlib.pyplot as plt
```

Three contexts:

- Python script: (example)

```
xs = np.linspace(0, 10, 100) # 100 evenly spaced points in [0,10]
plt.plot(xs, np.sin(xs))
plt.plot(xs, np.cos(xs))
plt.show()
```

  After constructing plot, call `plt.show()` only once, typically at end of script. Result of calling `plt.show()` multiple times is undefined.

- iPython: (I've had better luck with the qt5 backend.)

```
In [4]: %matplotlib qt5
```

  Now any plot command opens a figure window. Force redraw with `plt.draw()`.

- Jupyter Notebook, two options:

```
%matplotlib notebook
```

  Embed interactive plots in notebook.

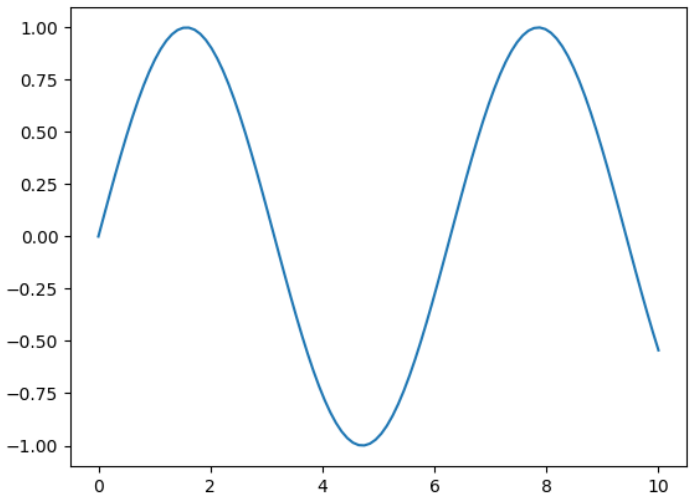
```
%matplotlib inline
```

  Embed static plot images in notebook. We'll usually use this option.

Matlab-style Interface

It's easy to throw up a simple plot using the stateful Matlab-style interface:

```
xs = np.linspace(0, 10, 100)
plt.figure()
plt.plot(xs, np.sin(xs))
```



However, every plot resides in a figure, which can have a number of subplots. Managing these figures is clunky using the Matlab-style interface. For that reason we'll use the object-oriented interface on explicit references to the figures and axes.

Figures and Axes

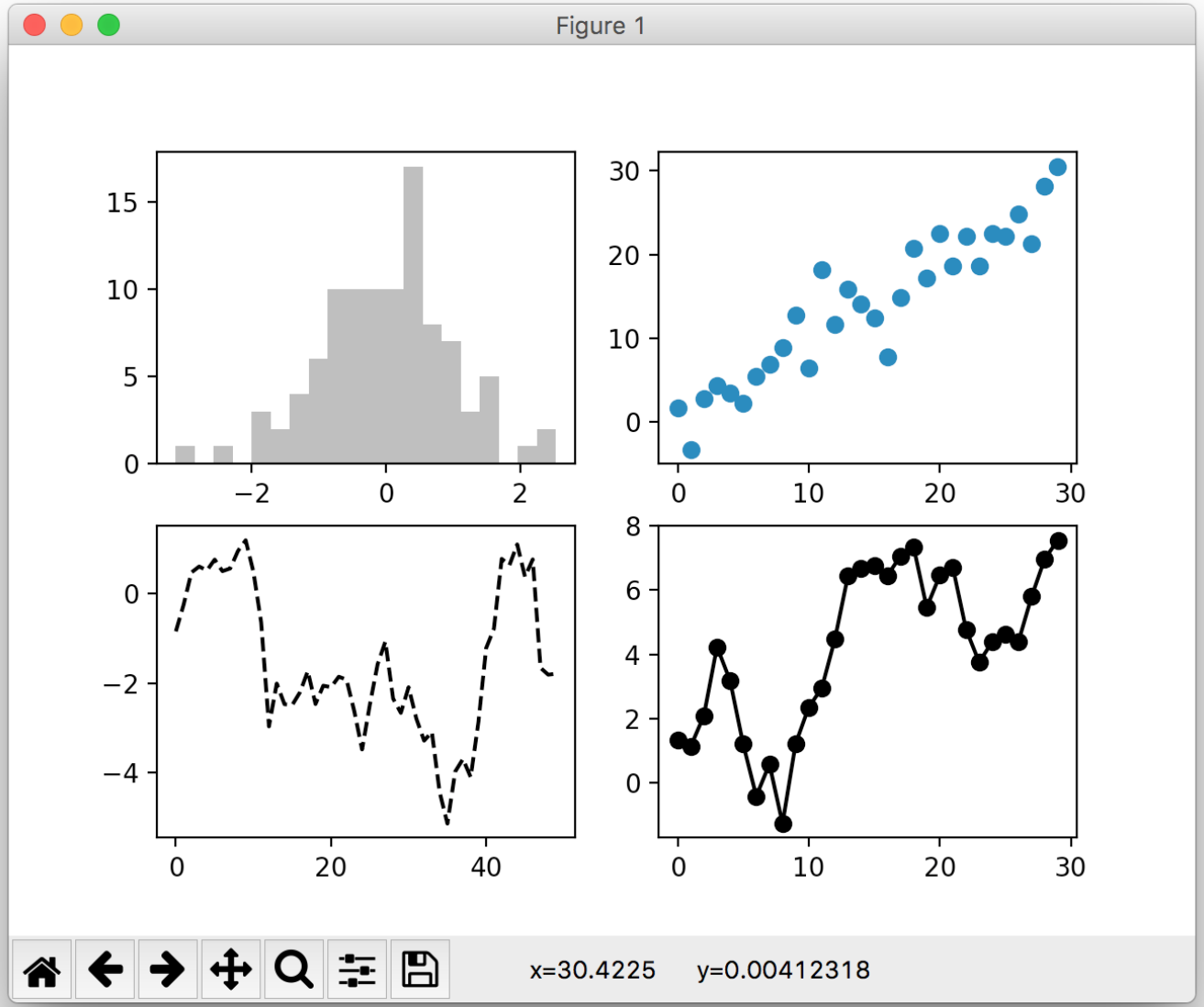
Here we make four subplots in a 2x2 layout and put a different kind of plot in each one. Notice 1-based indexing third argument – top left to bottom right.

```
In [5]: fig = plt.figure()

In [6]: ax1 = fig.add_subplot(2, 2, 1)

In [7]: ax2 = fig.add_subplot(2, 2, 2)

In [9]: ax3 = fig.add_subplot(2, 2, 3)

In [10]: ax4 = fig.add_subplot(2, 2, 4)

In [13]: ax1.hist(np.random.randn(100), bins=20, color='k', alpha=0.3)
Out[13]: ... elided for brevity
 <a list of 20 Patch objects>)

In [14]: ax2.scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))
Out[14]: <matplotlib.collections.PathCollection at 0x11477c1d0>

In [15]: ax3.plot(np.random.randn(50).cumsum(), 'k--')
Out[15]: [<matplotlib.lines.Line2D at 0x114411fd0>]

In [18]: ax4.plot(np.random.randn(30).cumsum(), 'ko-')
Out[18]: [<matplotlib.lines.Line2D at 0x1146ce0b8>]
```

Georgia Tech

Figures

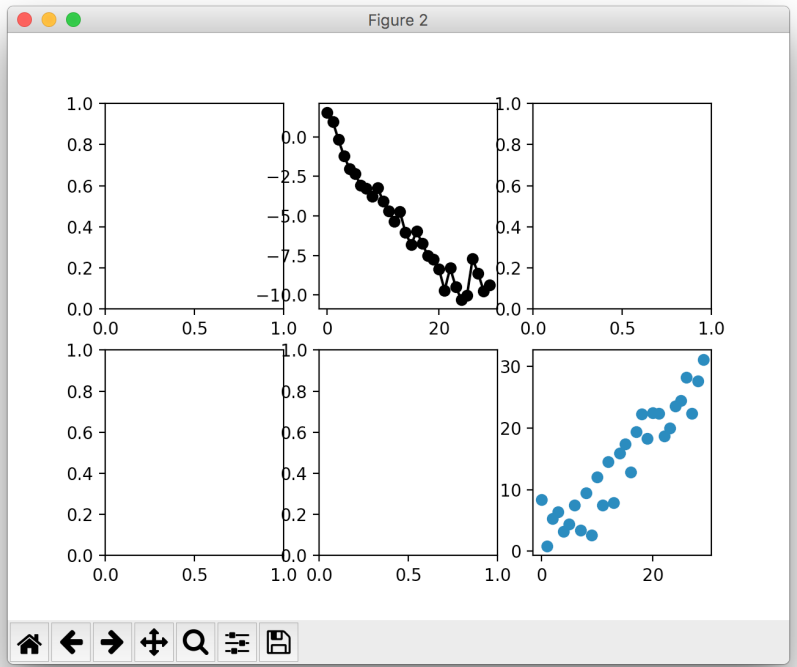
The commands on the previous slide would produce this:



plt.subplots

Matplotlib includes a convenience method for making subplots.

```
In [20]: fig, axes = plt.subplots(2, 3)

In [22]: axes[0,1].plot(np.random.randn(30).cumsum(), 'ko-')
Out[22]: [<matplotlib.lines.Line2D at 0x1204e4470>]

In [23]: axes[1,2].scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))
Out[23]: <matplotlib.collections.PathCollection at 0x1204f8940>
```

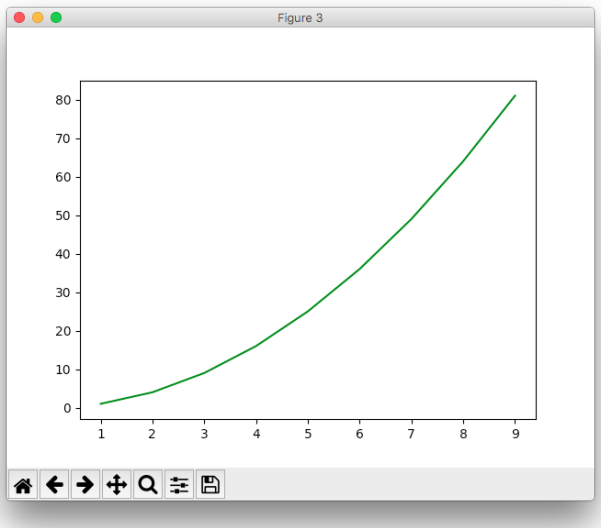


Note the 0-based indexing for axes.

Single Plot Shortcut

```
In [35]: xs, ys = np.arange(1, 11), np.arange(1, 11) ** 2

In [37]: fig, axis = plt.subplots(1, 1)

In [38]: axis.plot(xs, ys, linestyle='-', color='g')
Out[38]: [<matplotlib.lines.Line2D at 0x120c60518>]
```

- Notice that if you create a figure with one subplot `plt.subplots` returns a single axis instead of an array of axes.
- Notice also the explicit linestyle and color.



This is how I usually do a simple one-plot figure.

Colors, Markers, and Line Styles

Notice the 'ko-' in `plot(np.random.randn(30).cumsum(), 'ko-')`

- 'k' is a color for the marker and line used in the plot. A few examples:
  - 'b' - blue
  - 'g' - green
  - 'r' - red
  - 'k' - black
  - 'w' - white
- 'o' is a marker. A few examples:
  - '.' - point marker
  - ',' - pixel marker
  - 'o' - circle marker
  - 'v' - triangle_down marker
  - '^' - triangle_up marker
  - '<' - triangle_left marker
  - '>' - triangle_right marker
- '-' is a line style. A few examples:
  - '-' - solid line style
  - '–' - dashed line style
  - '-. - dash-dot line style
  - ':' - dotted line style

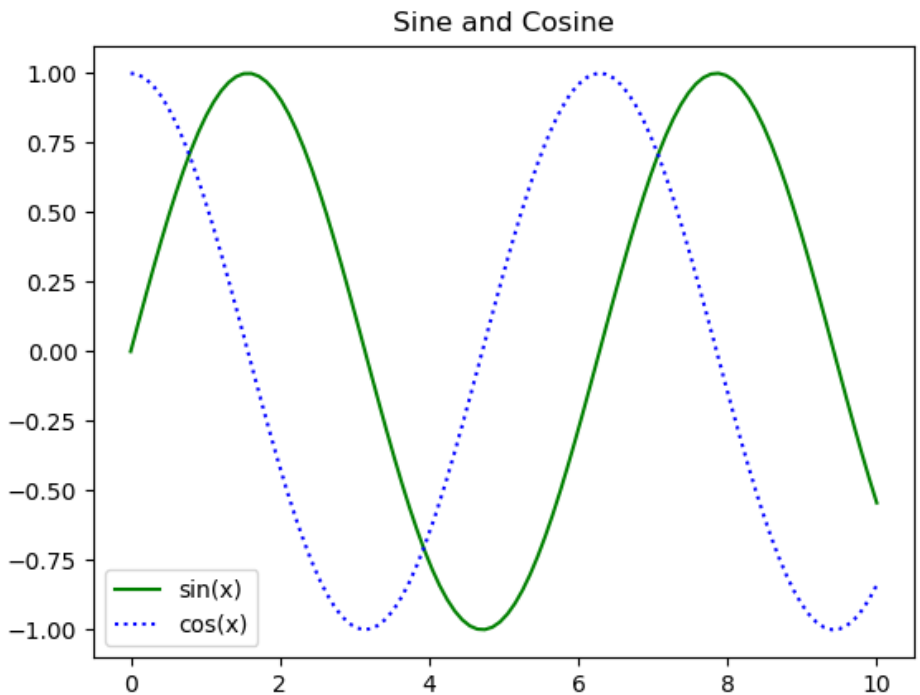
For complete details see documentation for `plt.plot`

Georgia Tech

Legends

A legend is a label for one of the series of data on an axis. Here's an axis with two plots, each having a different label which appears in a legend.

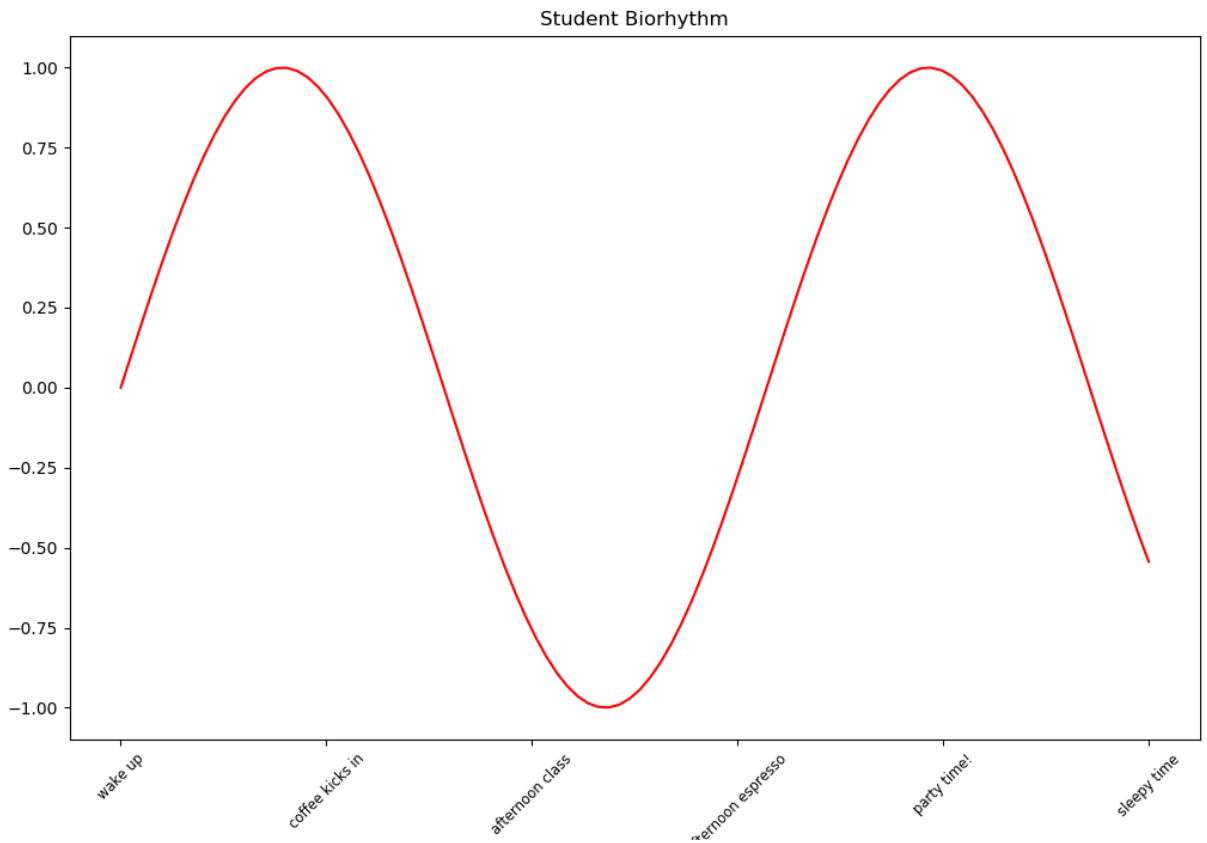
```
xs = np.linspace(0, 10, 100)
fig, ax = plt.subplots(1, 1)
ax.plot(xs, np.sin(xs), "-g", label="sin(x)")
ax.plot(xs, np.cos(xs), ":b", label="cos(x)")
ax.legend() # Causes the legend to be displayed
ax.set_title("Sine and Cosine")
```



Ticks and Labels

Ticks and labels are set automatically but can be customized.

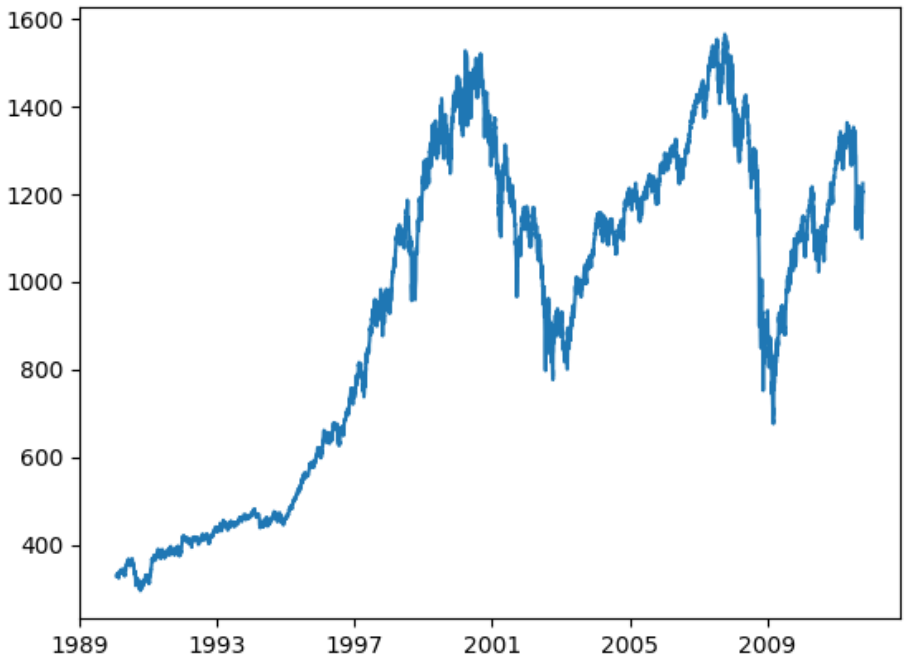
```
fig, ax = plt.subplots(1, 1)
ax.plot(xs, np.sin(xs), "-r")
ax.set_xticklabels(["wake up", "coffee kicks in", "afternoon class",
                    "afternoon espresso",   "party time!", "sleepy time"],
                   rotation=45, fontsize="small")
ax.set_title("Student Biorhythm")
```



SPX Example

To start, download historical S&P 500 index data: spx.csv. [1]

```
spx = pd.read_csv('spx.csv', index_col=0, parse_dates=True)
fig, ax = plt.subplots(1,1)
ax.plot(spx.index, spx['SPX'])
```



[1] Example and data courtesy of Wes McKinney
https://github.com/wesm/pydata-book

Annotations

Define annotation data (note use of collections.namedtuple).

```
import collections
Annotation = collections.namedtuple('Annotation', ['label', 'date'])
events = [Annotation(label="Peak bull market", date=dt.datetime(2007, 10, 11)),
          Annotation(label="Bear Stearns fails", date=dt.datetime(2008, 3, 12)),
          Annotation(label="Lehman bankruptcy", date=dt.datetime(2008, 9, 15))]
```

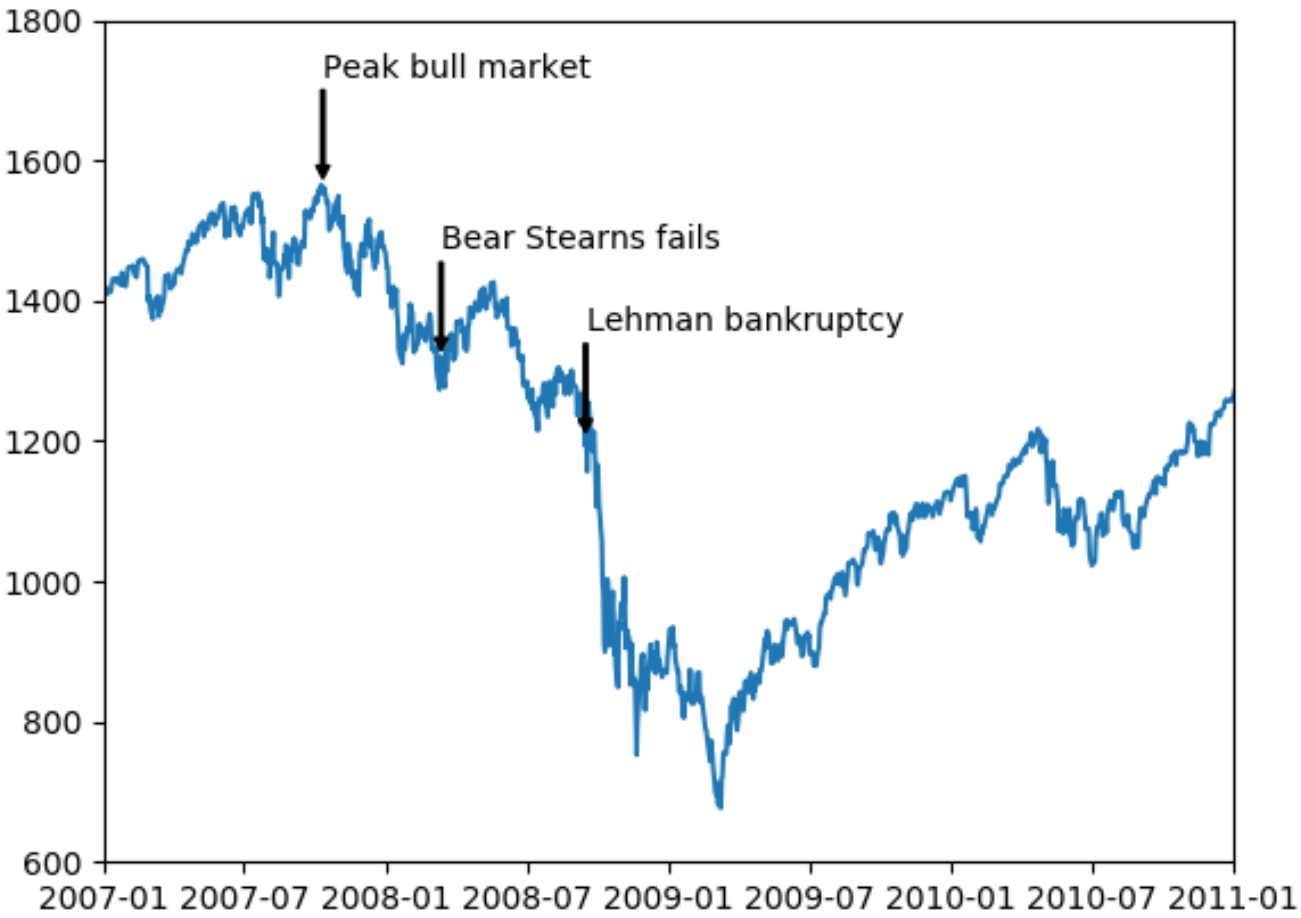
Zoom in on period of interest and add annotations. [2]

```
ax.set(xlim=['1/1/2007', '1/1/2011'], ylim=[600, 1800])
for event in events:
    ax.annotate(event.label,
                xy=(event.date, spx.asof(event.date) + 20),
                xytext=(event.date, spx.asof(event.date) + 200),
                arrowprops=dict(facecolor="black",
                                headwidth=4, width=1, headlength=4),
                horizontalalignment="left", verticalalignment="top")
```

- `xy` is the x, y position of the arrowhead
- `xytext` is the x, y position of the label
- `arrowprops` defines the shape of the arrow and its head. Note that the length is determined by the distance between `xy` and `xytext`.
- See `matplotlib.text.Text` for `horizontalalignment` and `verticalalignment`

[2] See docs for pandas.DataFrame.asof

Annotated SPX Plot



Saving Plots to Files

Saving figures is usually as easy as

```
fig.savefig("figure.png")
```

The graphics encoding is inferred by the file extension (after the last "."). You can find the supported file types and associated file name extensions on your system by:

```
In [77]: fig.canvas.get_supported_filetypes()
Out[77]:
{'eps': 'Encapsulated Postscript',
 'pdf': 'Portable Document Format',
 'pgf': 'PGF code for LaTeX',
 'png': 'Portable Network Graphics',
 'ps': 'Postscript',
 'raw': 'Raw RGBA bitmap',
 'rgba': 'Raw RGBA bitmap',
 'svg': 'Scalable Vector Graphics',
 'svgz': 'Scalable Vector Graphics'}
```

Note that there's no need to show the figure before saving it.

Georgia Tech