04. Matplotlib - Basic

September 23, 2020

아래 강의 노트는 Python Data Science Handbook 4장을 기반으로 번역 및 편집하여 페이지 구성함

무단 배포를 금지 합니다. ***

This notebook contains an excerpt from the Python Data Science Handbook by Jake VanderPlas; the content is available on GitHub.

The text is released under the CC-BY-NC-ND license, and code is released under the MIT license. If you find this content useful, please consider supporting the work by buying the book!

1 일반적인 matplotlib 사용법

- **matpotlib** 를 참고하여 더 많은 예를 참고 바랍니다.

2 Matplotlib 임포트 하기

NumPy 약어로 `np` 사용 Pandas 약어로 `pd` 사용

[1]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

전반적으로 `plt` 으로 많이 사용

2.1 스타일 설정하기

`plt.style` 사용해 그림에 적합한 스타일 선택

`classic` 을 사용 할 수 있고 다른 스타일을 원할 경우 여기 참고

[2]:
```
plt.style.use('classic')
```

스타일에 대한 자세항 사항은 327페이지에 조금더 자세히 배울 예정

2.2 `show()` 또는 No `show()`? 플롯 표현의 방법

Matplotlib는 a script, IPython 터미널, 또는 IPython notebook사용가능.

스크립트에서 matplot을 이용하여 그림 그리기

- script에서 Matplotlib를 사용한다면 `plt.show()`를 이용하여 그림 그리기 가능.
- `plt.show()` 는 이벤트 루프를 시작해 현재 활성화된 모든 그림 객체를 찾아서 그림을 표시
- 하나 이상의 대화창 열기 가능.

예를 들어 *myplot.py* 다음과 같이 구성될 경우:

```
# ------- file: myplot.py ------
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))

plt.show()
```

커맨드 창에 다음과 같이 실행

```
$ python myplot.py
```

- `plt.show()`파이썬 세션당 한버만 사용 가능
- 스크립트의 맨 마지막에 사용
- 여러번의 `show()` 할 경우, 빽엔드 종속적인 동작을 일으킬 수 있으 므로 피해야

IPython shell에서 플로팅 하기 IPython은 Matplotlib 모드 지정 하면 Matplotlib 사용 가능. `ipython` 시작후 `%matplotlib` 모드 활성화

```
In [1]: %matplotlib
Using matplotlib backend: TkAgg
```

```
In [2]: import matplotlib.pyplot as plt
```

plt 그림창을 열고 선의 속성을 수정하는 등의 변경사항은 자동적으로 그려지지 않음 plt.draw()를 이용해 업데이트 가능. Matplotlib mode 에서는 plt.show() 사용하지 않아도 괜찮음.

[3]:
```
%matplotlib
```

```
Using matplotlib backend: MacOSX
```

[4]:
```
import matplotlib.pyplot as plt
```

IPython notebook 에서 플로팅하기 IPython notebook은 IPython shell과 유사하게 %matplotlib 동작.

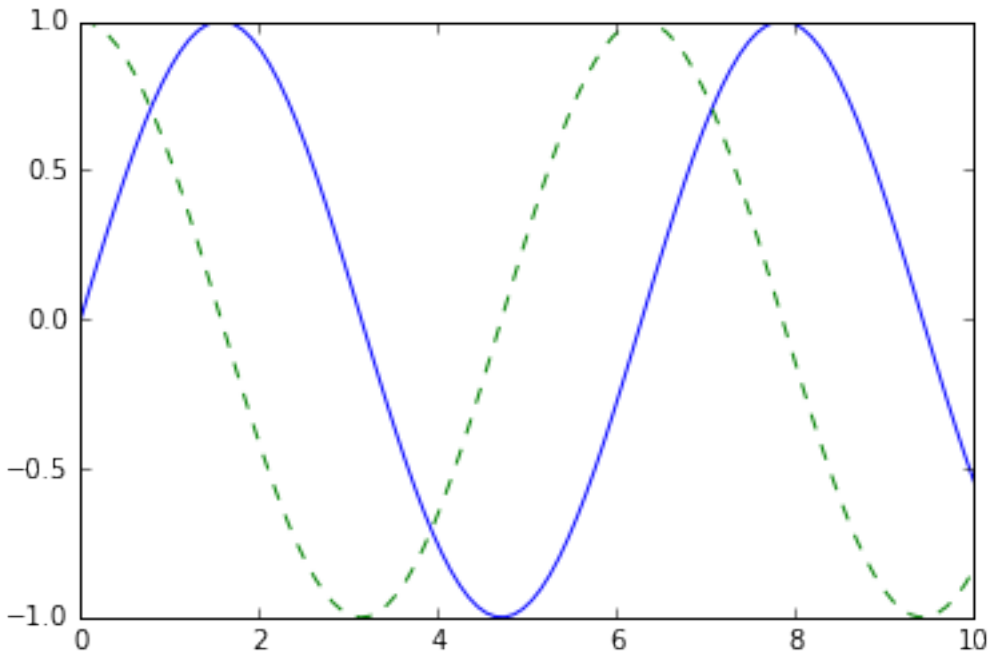
- %matplotlib notebook 노트북 내의 대화형 플롯을 삽입
- %matplotlib inline 노트북의 플롯의 정적 이미지 삽입

For this book, we will generally opt for %matplotlib inline:

[5]:
```
%matplotlib inline
```

이 명령어를 실행하면 플롯을 생성하는 노트북 내의 셀이 결과 그래픽의 PNG 이미지 삽입

[6]:
```
import numpy as np
x = np.linspace(0, 10, 100)

fig = plt.figure()
plt.plot(x, np.sin(x), '-')
plt.plot(x, np.cos(x), '--');
```



2.3 그림을 파일로 저장하기

`savefig()` 사용하여 그림 저장 가능

[7]:
```
fig.savefig('my_figure.png')
```

현재 작업하고 있는 디렉토리에 `my_figure.png` 저장

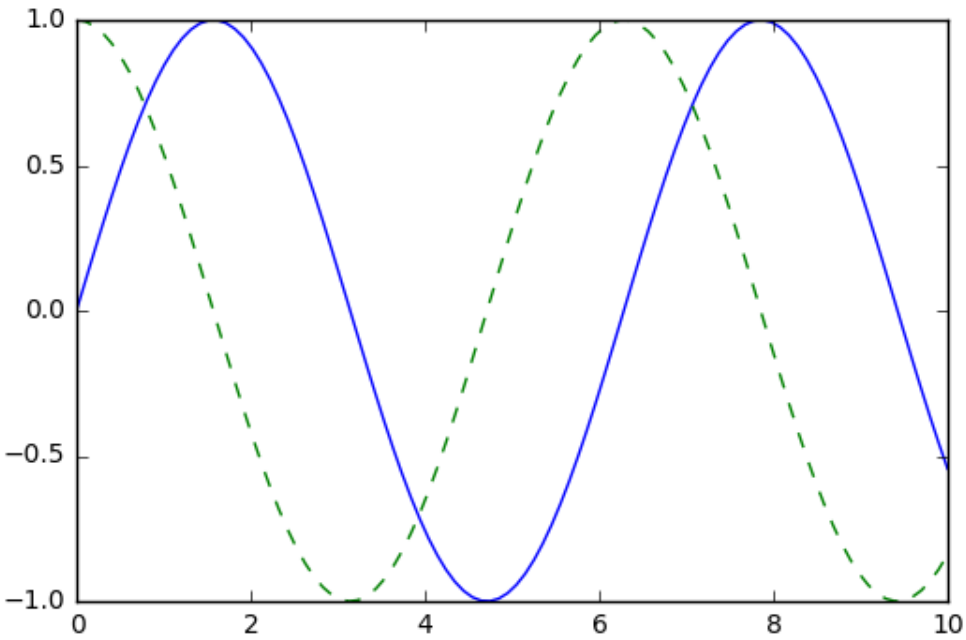
[8]:
```
!ls -lh my_figure.png
```

```
-rw-r--r--  1 Jaehee  staff    26K  9 10 14:37 my_figure.png
```

IPython `Image` 객체를 이용하여 파일의 내용 표시

[9]:
```
from IPython.display import Image
Image('my_figure.png')
```

[9]:



`savefig()`에서 형식 지정 가능 패키지를 어느 백엔드에 설치 했는지에 따라 여러 가지 파일 형식 가능

[10]:
```
fig.canvas.get_supported_filetypes()
```

[10]:
```
{'eps': 'Encapsulated Postscript',
 'jpg': 'Joint Photographic Experts Group',
 'jpeg': 'Joint Photographic Experts Group',
 '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',
 'tif': 'Tagged Image File Format',
 'tiff': 'Tagged Image File Format'}
```

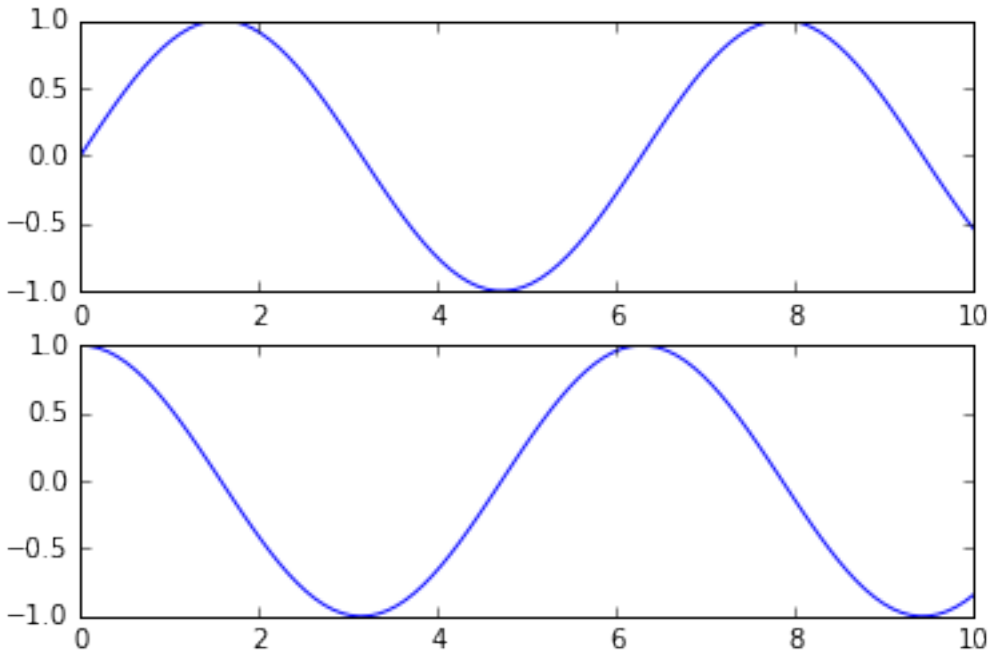
그림 저장시에는 `plt.show()`와 같은 명령어 사용할 필요 없음.

3 인터페이스 두개

- MATLAB-style 스타일의 상태 기반 인터페이스
- 객체지향 인터페이스

MATLAB스타일 인터페이스 Matplotlib은 MATLAB 사용자의 파이썬 대안으로 제작 The MATLAB 스타일 도구는 matplotlib.pyplot (`plt`) 인터페이스에 포함. MATLAB 스타일

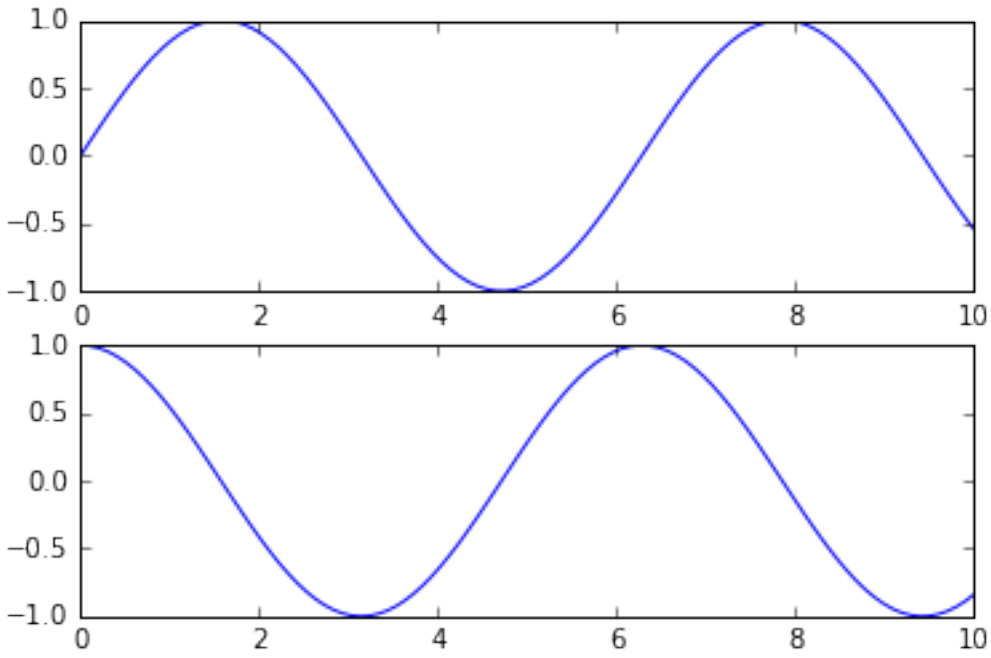
[11]:
```
plt.figure()  # create a plot figure

# create the first of two panels and set current axis
plt.subplot(2, 1, 1) # (rows, columns, panel number)
plt.plot(x, np.sin(x))

# create the second panel and set current axis
plt.subplot(2, 1, 2)
plt.plot(x, np.cos(x));
```



`plt.gcf()` (get current figure : 현재 그림을 읽임) 와 `plt.gca()` (get current axes: 현재 축을 읽음) 사용하여 참조 가능

객체 지향 인터페이스 객체지향 인터페이스에서 플로팅 함수는 "활성화" 그림이나 축의 개념에 의존하지 않는 명시적인 `Figure` 와 `Axes` 객체.

[12]:
```
# First create a grid of plots
# ax will be an array of two Axes objects
fig, ax = plt.subplots(2)

# Call plot() method on the appropriate object
ax[0].plot(x, np.sin(x))
ax[1].plot(x, np.cos(x));
```



플롯이 복잡해 질 수록 객체 지향 방식 사용 둘의 차이는 `plt.plot()` 와 `ax.plot()`바꿔쓰는 사소한 차이가 있음.

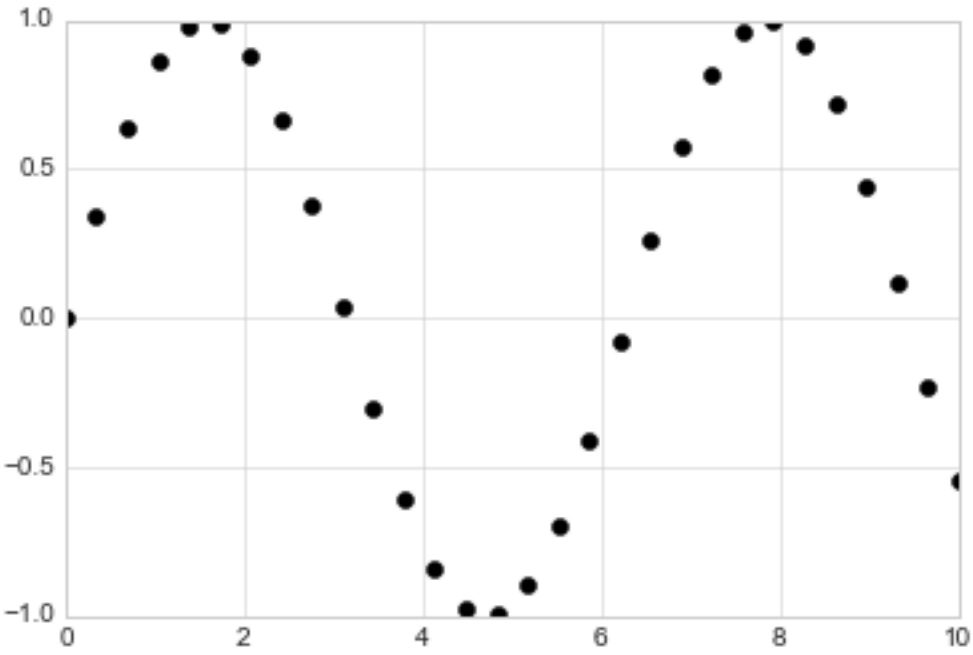
4 간단한 산점도

[13]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
```

4.1 `plt.plot` 사용한 산점도

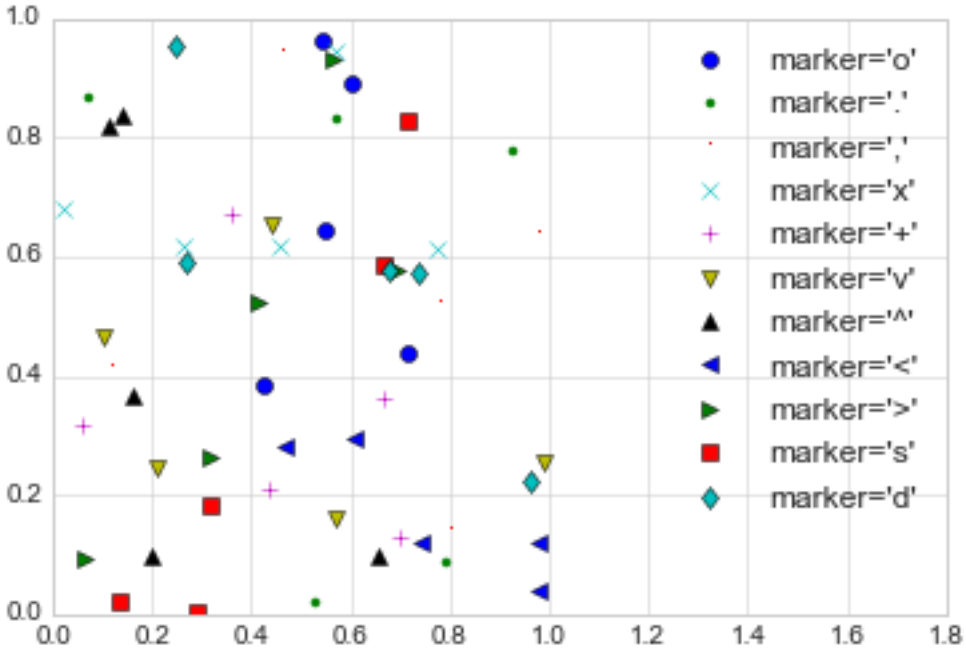
`plt.plot/ax.plot`와 유사하게 산점도 생성 가능.

[14]:
```
x = np.linspace(0, 10, 30)
y = np.sin(x)

plt.plot(x, y, 'o', color='black');
```



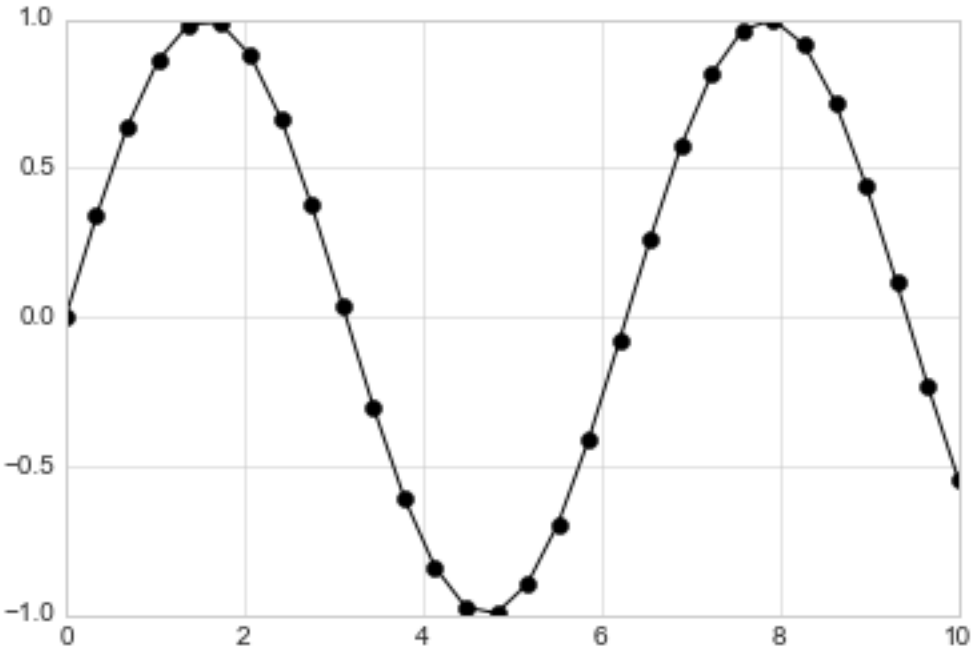
세번째 인수는 '-', '--'플로팅에 사용될 기호유형

[15]:
```
rng = np.random.RandomState(0)
for marker in ['o', '.', ',', 'x', '+', 'v', '^', '<', '>', 's', 'd']:
    plt.plot(rng.rand(5), rng.rand(5), marker,
             label="marker='{0}'".format(marker))
plt.legend(numpoints=1)
plt.xlim(0, 1.8);
```



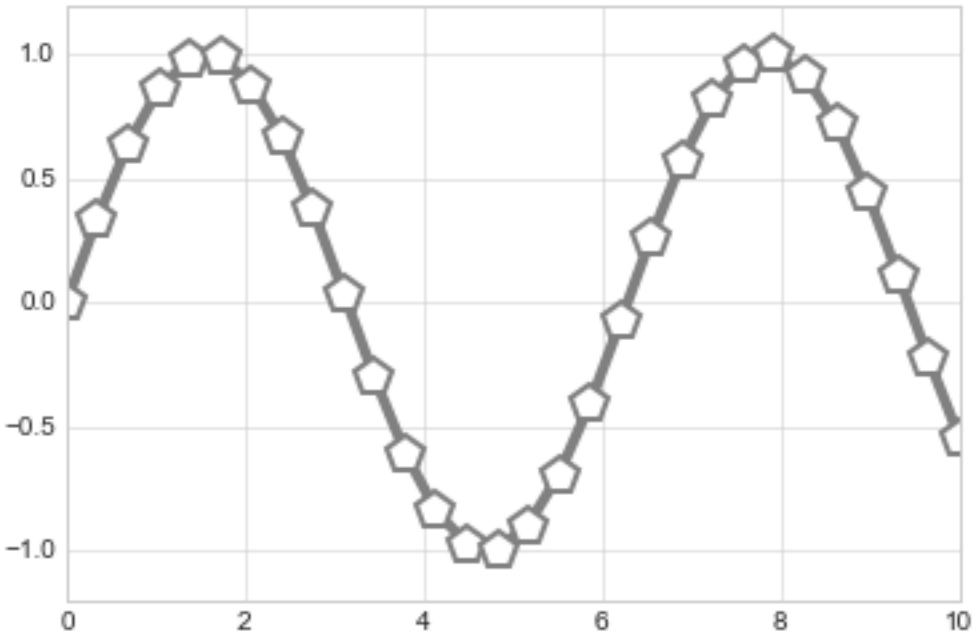
문자코드와 선과 색상코드를 함께 사용해 점들을 연결해 사용

[16]:
```
plt.plot(x, y, '-ok');
```



plt.plot 키워드는 다양한 선과 표시 속성 지정:

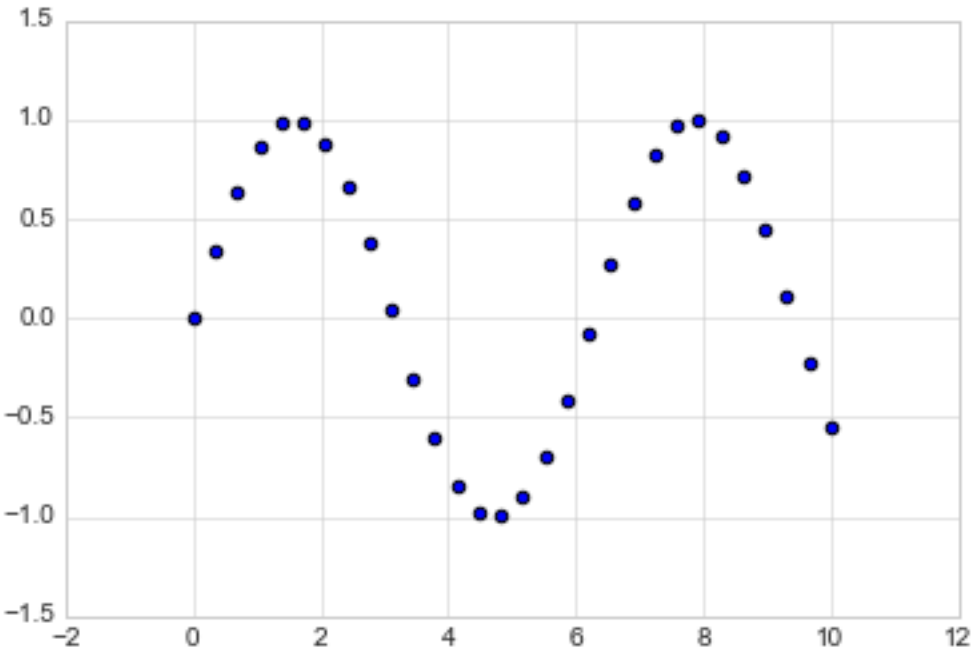
[17]:
```
plt.plot(x, y, '-p', color='gray',
         markersize=15, linewidth=4,
         markerfacecolor='white',
         markeredgecolor='gray',
         markeredgewidth=2)
plt.ylim(-1.2, 1.2);
```



4.2 plt.scatter를 활용한 산점도

plt.plot 와 유사하게 plt.scatter를 활용한 산점도 가능:

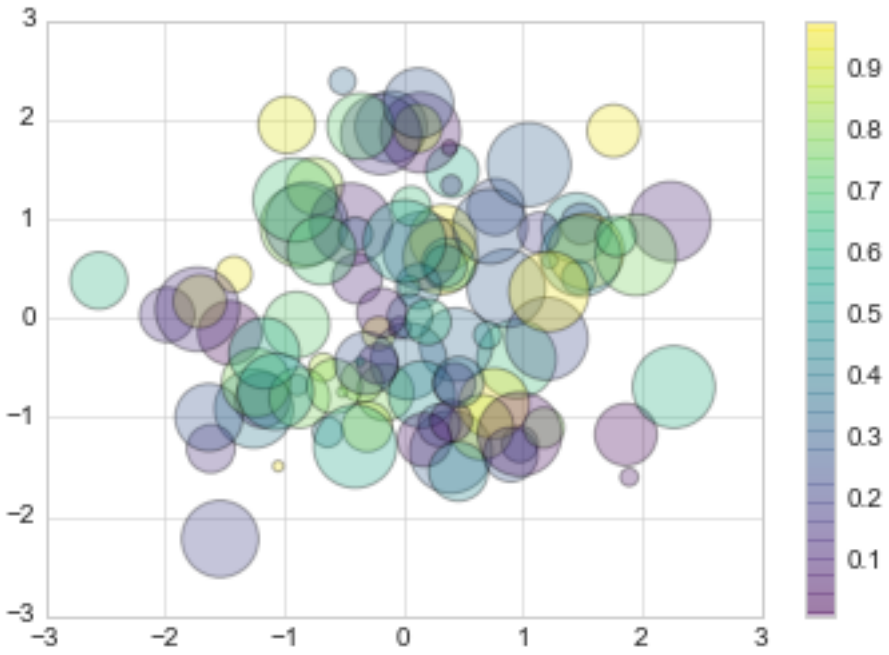
[18]:
```
plt.scatter(x, y, marker='o');
```



`plt.scatter` 와 `plt.plot`의 차이는 `plt.scatter`의 경우 각 점의 속성 (크기, 표면 색상, 테두리, 색상) 등을 개별적으로 제어하거나 데이터에 매핑 할 수 있는 산점도 생성 가능

아래 예는 산점도에서`alpha` 를 이용하여 투명도 조절가능:

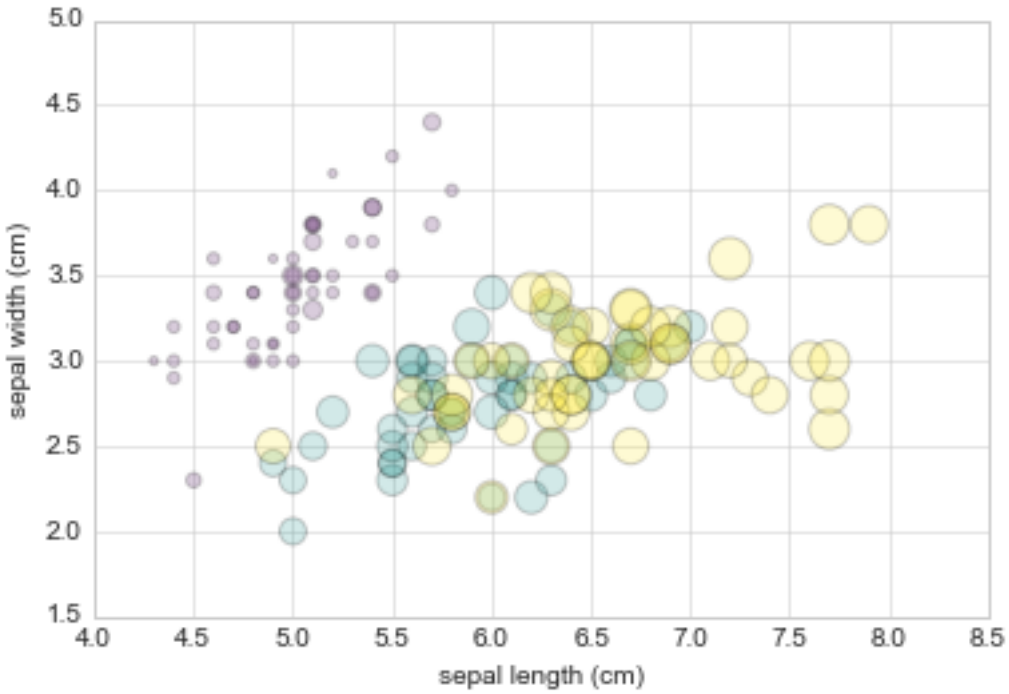
[19]:
```
rng = np.random.RandomState(0)
x = rng.randn(100)
y = rng.randn(100)
colors = rng.rand(100)
sizes = 1000 * rng.rand(100)

plt.scatter(x, y, c=colors, s=sizes, alpha=0.3,
            cmap='viridis')
plt.colorbar();  # show color scale
```



색상 인수는 색상 척도에 자동으로 매핑되는 `colorbar()`를 이용하여 픽셀단위로 보여줌. 점의 색상과 크기를 사용해 정보 전달 가능.

예를 들어 Scikit-Learn에서 붓꽃 데이터를 이용하여 데이터의 표본은 세가지 유형의 꽃중 하나로 꽃잎과 꽃받침의 크기를 세밀하게 측정하 값을 가지고 있음:

[20]:
```
from sklearn.datasets import load_iris
iris = load_iris()
features = iris.data.T

plt.scatter(features[0], features[1], alpha=0.2,
            s=100*features[3], c=iris.target, cmap='viridis')
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1]);
```



산점도로 네가지 차원을 동시 탐색 * 각 점의 (x,y) 위치는 꽃 받침의 길이와 폭 * 점의 크기 : 꽃잎의 폭 * 색상 : 꽃의 종

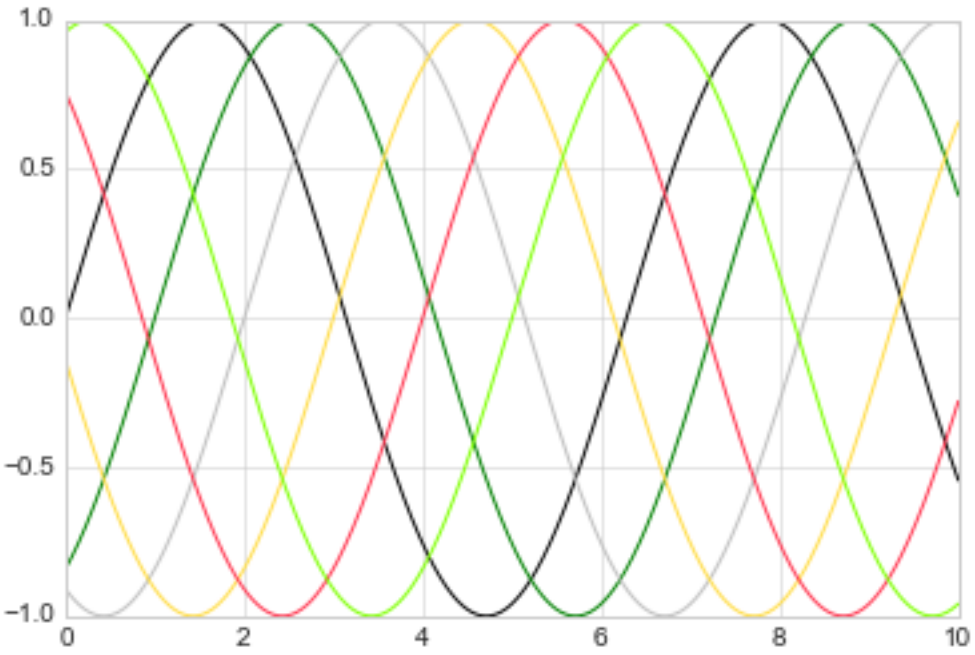
4.3 `plot` vs `scatter`: 효율성 측면 유의점

데이터의 수가 많을 경우 `plt.plot` 이 `plt.scatter`보다 효율적 `plt.scatter`는 각 점에 대한 다양한 크기와 색상을 나타내는 능력이 있어 각 점을 개별적으로 구성하는 추가 작업을 해야 함 `plt.plot`점이 기본적으로 항상 서로 복제 되므로 점의 모양을 결정하는 작업이 전체 데이터 집합에 대해 한번만 수행

4.4 플롯 수정하기: 선 색상과 스타일

선의 색상과 스타일을 변경하기 위해 `plt.plot()` 함수에 추가 인수 받기로 가능 색깔 조정 `color` keyword 사용

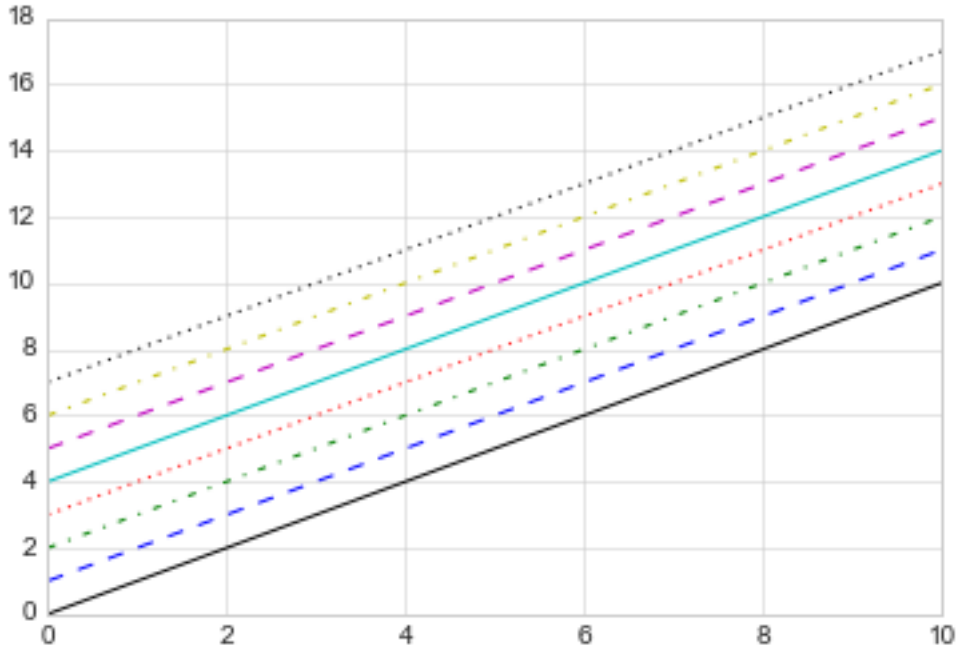
[21]:
```
x = np.linspace(0, 10, 1000)
plt.plot(x, np.sin(x - 0), color='black')         # 색상 이름으로 지정
plt.plot(x, np.sin(x - 1), color='g')            # 짧은 색상 코드로 지정 (rgbcmyk)
plt.plot(x, np.sin(x - 2), color='0.75')         # 0 과 1 사이의 회색조로 이용
plt.plot(x, np.sin(x - 3), color='#FFDD44')      # Hex code (RRGGBB from 00 to FF)
plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3)) # 0 과 1 사이의 RGB 튜플값
plt.plot(x, np.sin(x - 5), color='chartreuse'); # HTML 색상 이름 지원
```



색상에 관한 자세한 정보는 를 여기 참고 `linestyle` 키워드를 이용하여 라인 스타일 변경:

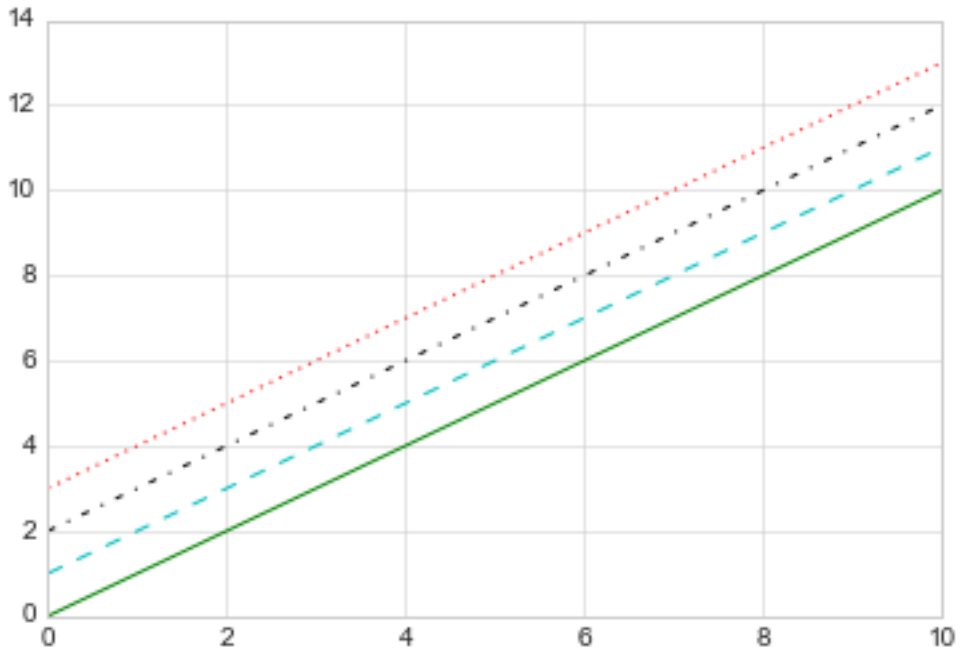
[22]:
```
plt.plot(x, x + 0, linestyle='solid', color='black')
plt.plot(x, x + 1, linestyle='dashed')
plt.plot(x, x + 2, linestyle='dashdot')
```

```
plt.plot(x, x + 3, linestyle='dotted');

# For short, you can use the following codes:
plt.plot(x, x + 4, linestyle='-')  # solid
plt.plot(x, x + 5, linestyle='--') # dashed
plt.plot(x, x + 6, linestyle='-.') # dashdot
plt.plot(x, x + 7, linestyle=':');  # dotted
```



`linestyle`과 `color` 를 `plt.plot()`함수에서 함께 사용:

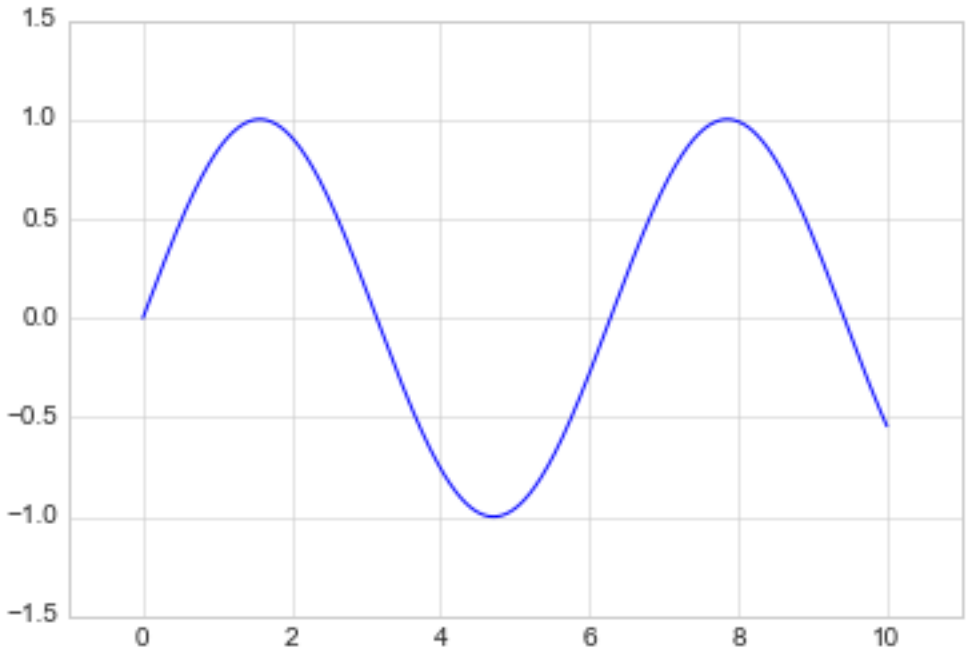
[23]:
```
plt.plot(x, x + 0, '-g')  # solid green
plt.plot(x, x + 1, '--c') # dashed cyan
plt.plot(x, x + 2, '-.k') # dashdot black
plt.plot(x, x + 3, ':r');  # dotted red
```



4.5 Plot 조정하기 : 축 경계

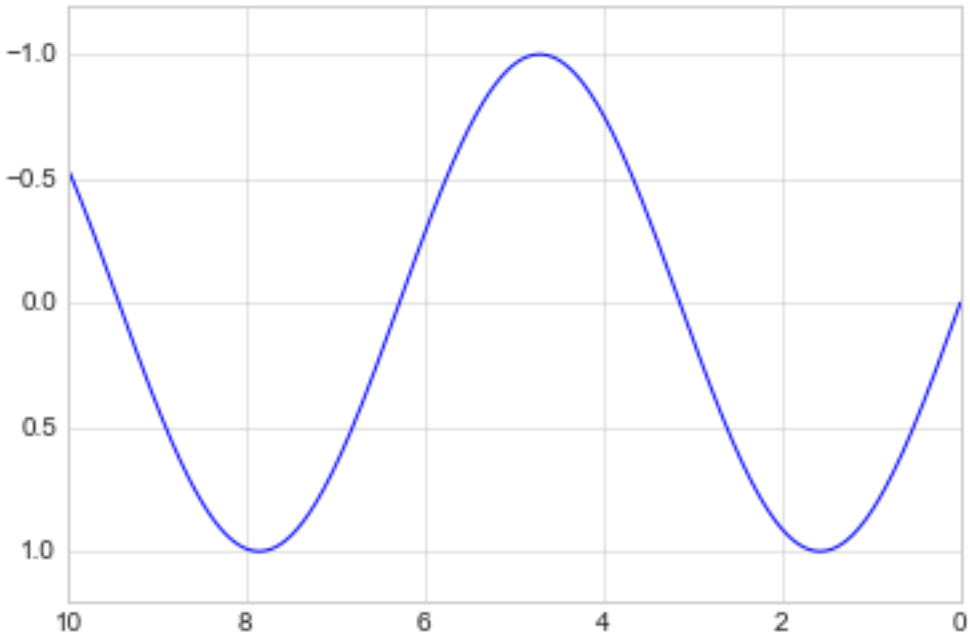
Matplotlib은 기본적으로 축 경계를 적절하게 선택 `plt.xlim()` and `plt.ylim()` methods:

[24]:
```
plt.plot(x, np.sin(x))

plt.xlim(-1, 11)
plt.ylim(-1.5, 1.5);
```



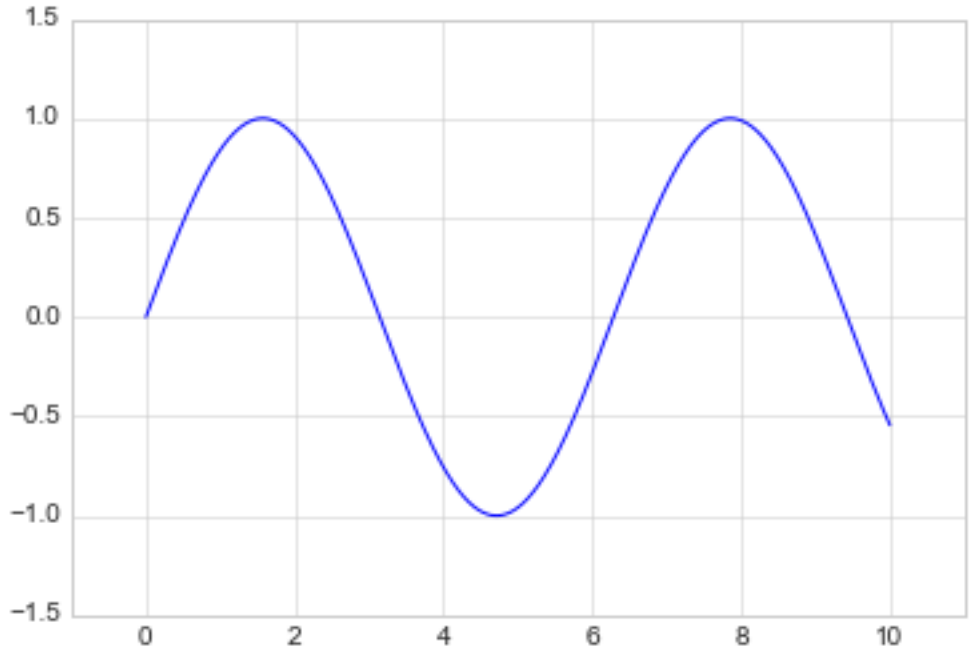
축의 범위를 거꾸로 표현할 경우 첫번째 값에 더 큰 숫자를 넣고 두번째 값에 더 작은 숫자를 넣을 경우 축을 거꾸로 표현 가능

[25]:
```
plt.plot(x, np.sin(x))

plt.xlim(10, 0)
plt.ylim(1.2, -1.2);
```



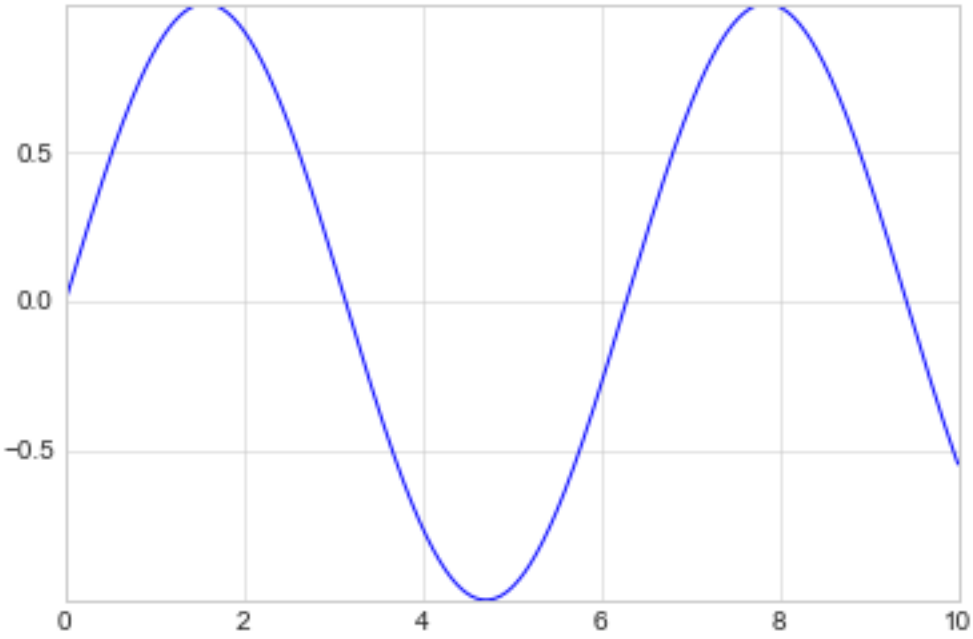
plt.axis() 는[xmin, xmax, ymin, ymax] 와 같이 네개의 값을 넣어 동시에 축 변경 가능

[26]:
```
plt.plot(x, np.sin(x))
plt.axis([-1, 11, -1.5, 1.5]);
```



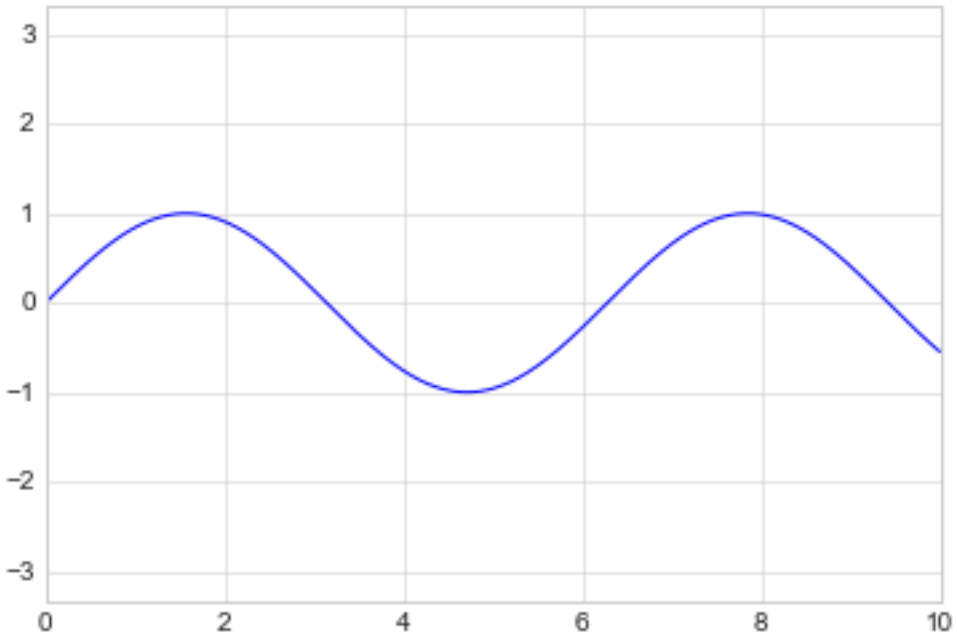
축에 자동으로 맞게 하기 위해 다음과 같이 사용 가능 `plt.axis('tight')`

[27]:
```
plt.plot(x, np.sin(x))
plt.axis('tight');
```



x 와 y 축의 비율을 맞추기 위해서 다음과 같이 사용 가능 `plt.axis('equal');`:

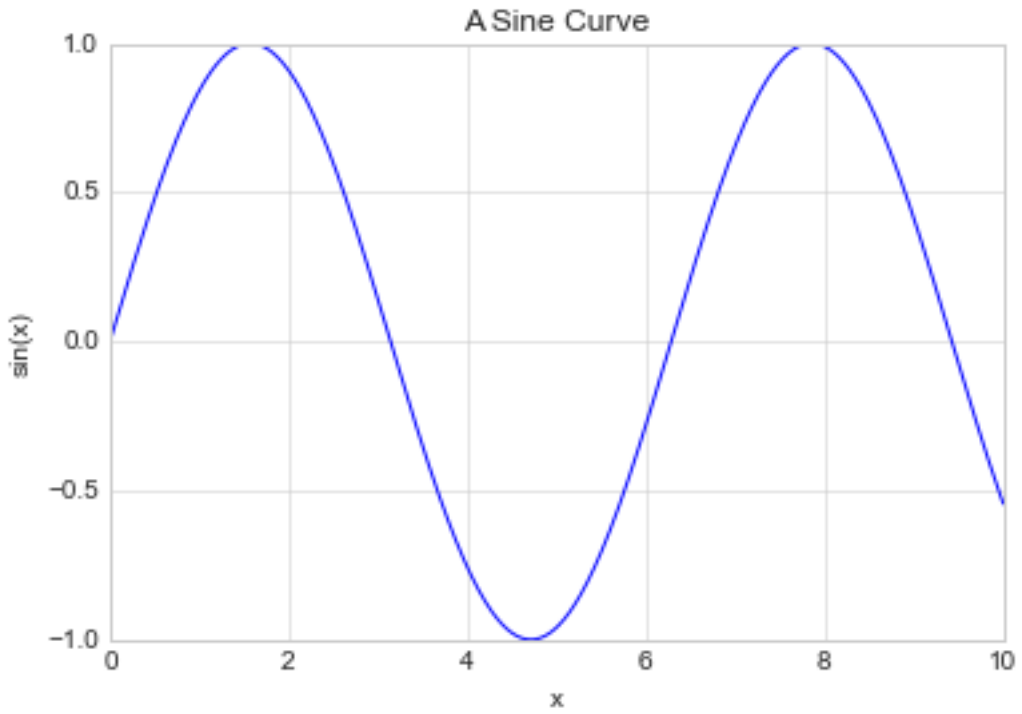
[28]:
```
plt.plot(x, np.sin(x))
plt.axis('equal');
```



4.6 그림에 라벨링

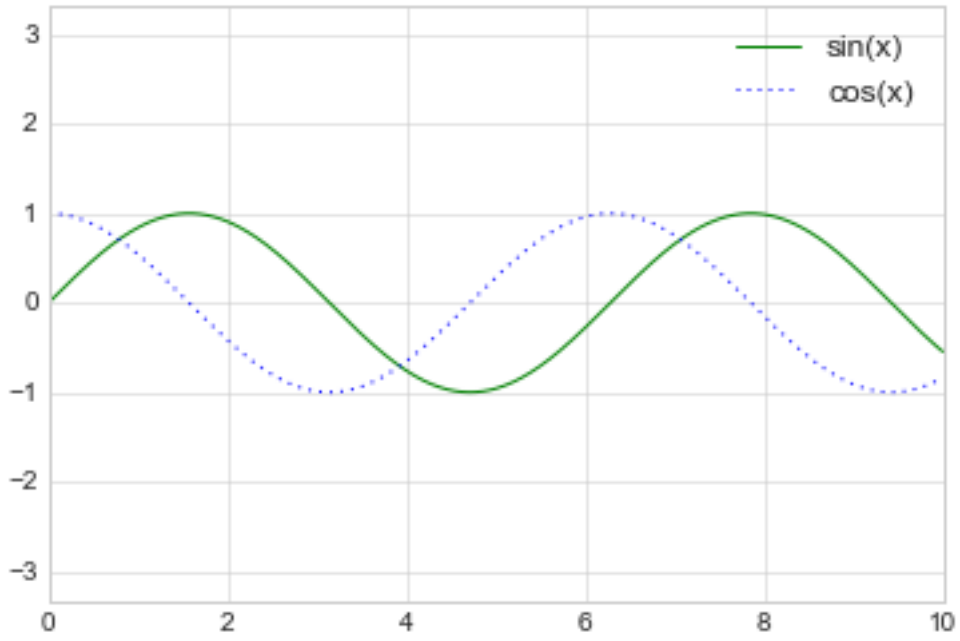
제목과 축 레이블은 가증 간단란 레이블

[29]:
```
plt.plot(x, np.sin(x))
plt.title("A Sine Curve")
plt.xlabel("x")
plt.ylabel("sin(x)");
```



축안에 여러 선을 표시하는 경우 각 선의 유형에 레이블을 붙이는 범례를 생성 `plt.legend()` 메소드 이용.

[30]:
```
plt.plot(x, np.sin(x), '-g', label='sin(x)')
plt.plot(x, np.cos(x), ':b', label='cos(x)')
plt.axis('equal')

plt.legend();
```



`plt.legend()` 함수가 라인 스타일과 색상을 기록하고 이를 정확한 레이블과 매칭

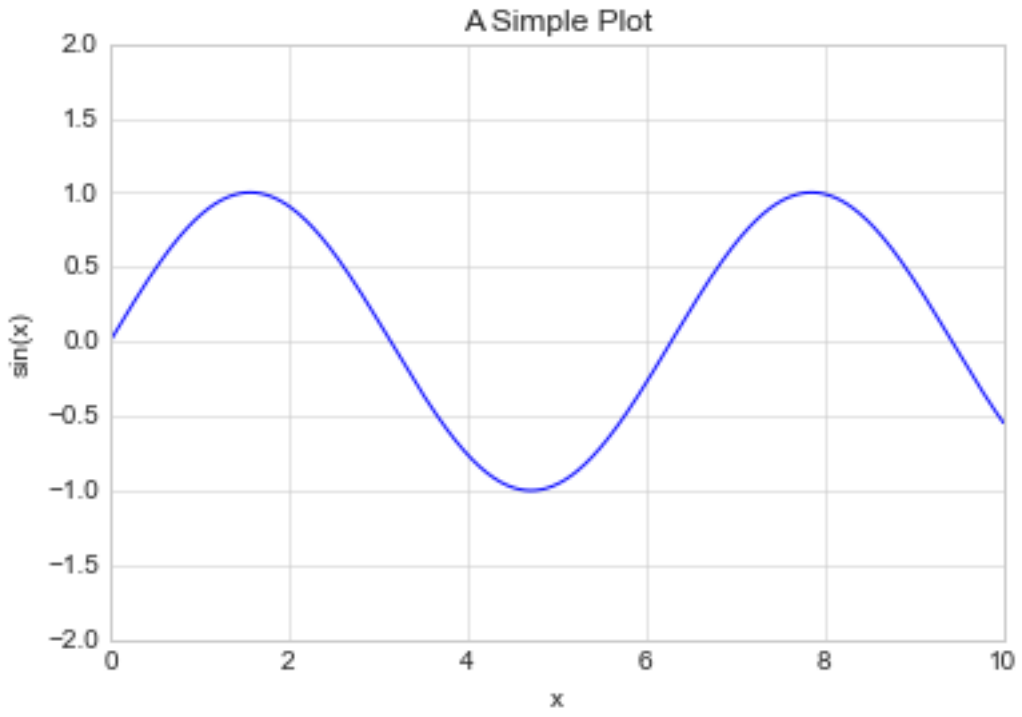
4.7 Aside: Matplotlib Gotchas

대부분의 `plt` 함수는 (`plt.plot()` → `ax.plot()`, `plt.legend()` → `ax.legend()`, etc.) 처럼 바로 `ax` 메소드로 변경 가능 하지만 모두 그런것은 아님 매트랩 스타일 함수와 객체 지향 메스드 사에어서 전환시 다음과 같이 변경 해야 함:

- `plt.xlabel()` → `ax.set_xlabel()`
- `plt.ylabel()` → `ax.set_ylabel()`
- `plt.xlim()` → `ax.set_xlim()`
- `plt.ylim()` → `ax.set_ylim()`
- `plt.title()` → `ax.set_title()`

플로팅을 위한 객체 지향 인터페이스에서는 이 함수들을 개별적으로 호출 하는 대신`ax.set()` 메서드를 사용하여 한꺼 번에 속성을 지정

[31]:
```
ax = plt.axes()
ax.plot(x, np.sin(x))
ax.set(xlim=(0, 10), ylim=(-2, 2),
       xlabel='x', ylabel='sin(x)',
       title='A Simple Plot');
```



5 다중 서브 플롯

때로는 서로 다른 데이터 뷰를 나란히 비교하는 것이 유용 할 경우가 있음 Matplotlib 에서는 *subplots*: 개념을 이용하여 하나의 그림내에 공존 할 수 있는 더 작은 축의 그림을 그릴 수 있음

[32]:
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')
import numpy as np
```

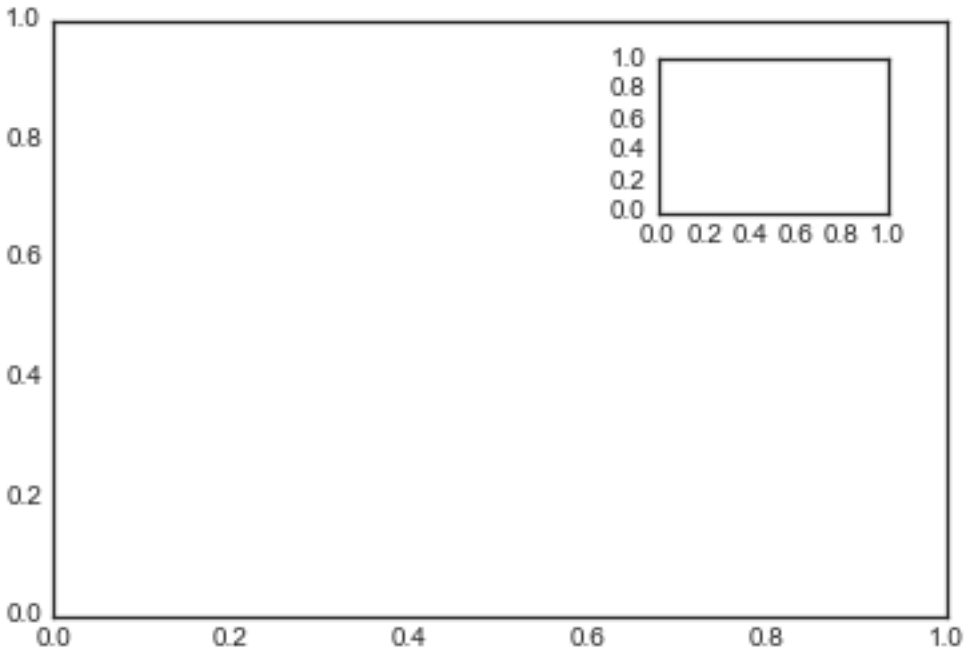
5.1 `plt.axes`: 직접 만든 서브 플롯

가장 기본 적인 방법은 `plt.axes` 함수 사용. 전체 그림을 채우는 표준 축 객체를 생성. `plt.axes` 는 선택적으로 네 개의 숫자 목록을 인수로 취할 수 있음 . 숫자는 좌표계의 `[left, bottom, width, height]` 를 나타내는 것으로 그림 왼족 하단의 0 부터 오른쪽 상단의 1까지의 범위를 나타냄.

예를 들어 x 와 y의 위치를 0.65 설정(그림 너비의 65% 와 높이의 65%에서 시작)

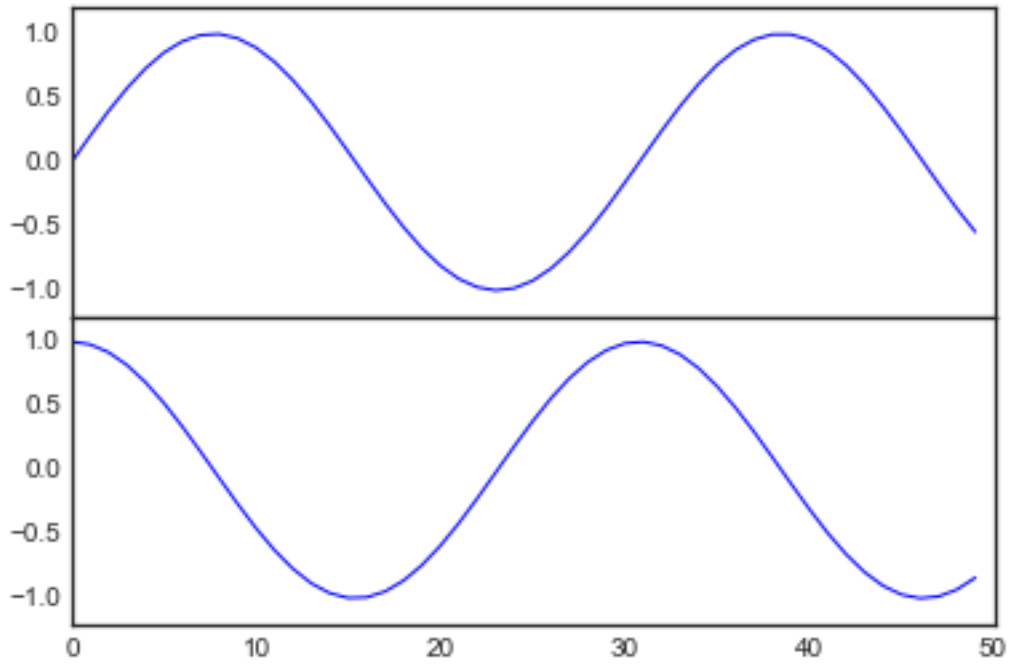
x 와 y 의 범위를 0.2로 설정 (축의 크기는 그림 너비와 높이의 각 20%가 됨):

[33]:
```
ax1 = plt.axes()  # standard axes
ax2 = plt.axes([0.65, 0.65, 0.2, 0.2])
```



객체 지향에 인터페이스 에서는 `fig.add_axes()`로 사용 두개의 세로로 배치된 축 만들기 예:

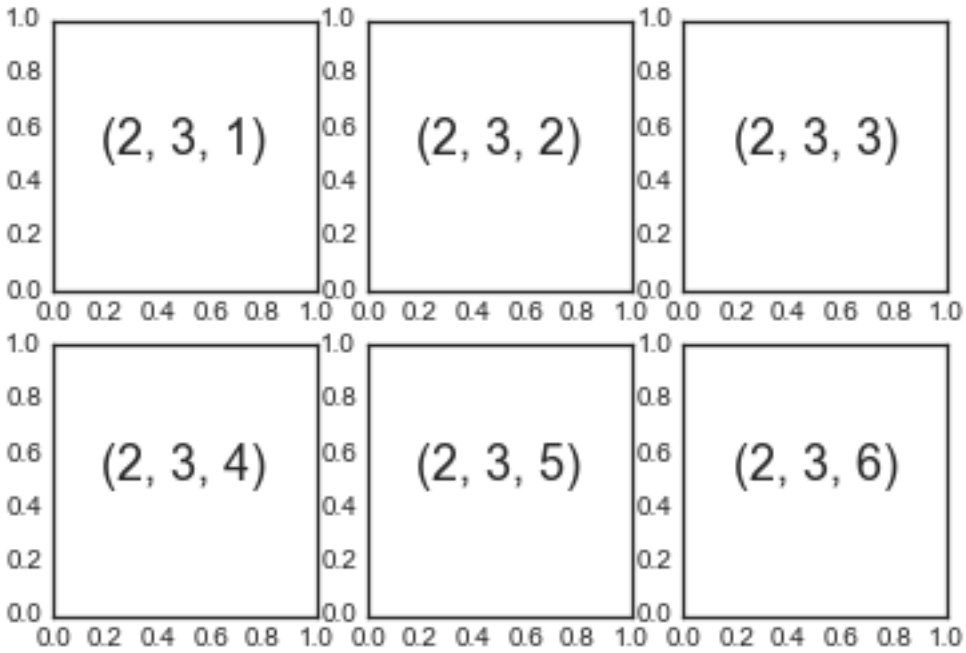
[54]:
```
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax1 = fig.add_axes([0.1, 0.5, 0.8, 0.4],
                   xticklabels=[], ylim=(-1.2, 1.2))
ax2 = fig.add_axes([0.1, 0.1, 0.8, 0.4],
                   ylim=(-1.2, 1.2))

x = np.linspace(0, 10)
ax1.plot(np.sin(x))
ax2.plot(np.cos(x));
```



5.2 `plt.subplot`: 간단한 서브 플롯의 그리드

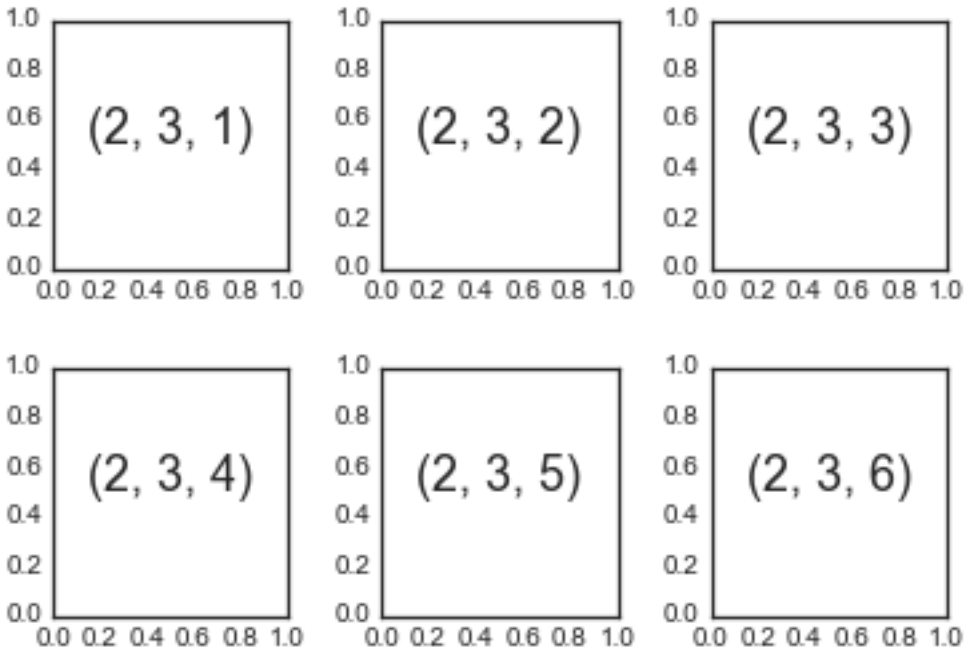
`plt.subplot()`그리드 안에 하나의 서브 플롯 생성 세개의 정수 인수를 취함 왼쪽 상단에서 오른쪽 하단으로 이어지는 행과 열의 갯수 플롯의 인덱스 의미:

[35]:
```
for i in range(1, 7):
    plt.subplot(2, 3, i)
    plt.text(0.5, 0.5, str((2, 3, i)),
             fontsize=18, ha='center')
```



- plt.subplots_adjust는 플롯들 사이의 간격을 조정하는데 사용. 객체 지향 명령어의 경우 fig.add_subplot()사용:
- 그림의 높이와 너비를 따라 서브 플롯의 크기 단위로 간격을 지정하기 위해 plt.subplots_adjust의hspace 와 wspace 사용

[36]:
```
fig = plt.figure()
fig.subplots_adjust(hspace=0.4, wspace=0.4)
for i in range(1, 7):
    ax = fig.add_subplot(2, 3, i)
    ax.text(0.5, 0.5, str((2, 3, i)),
           fontsize=18, ha='center')
```



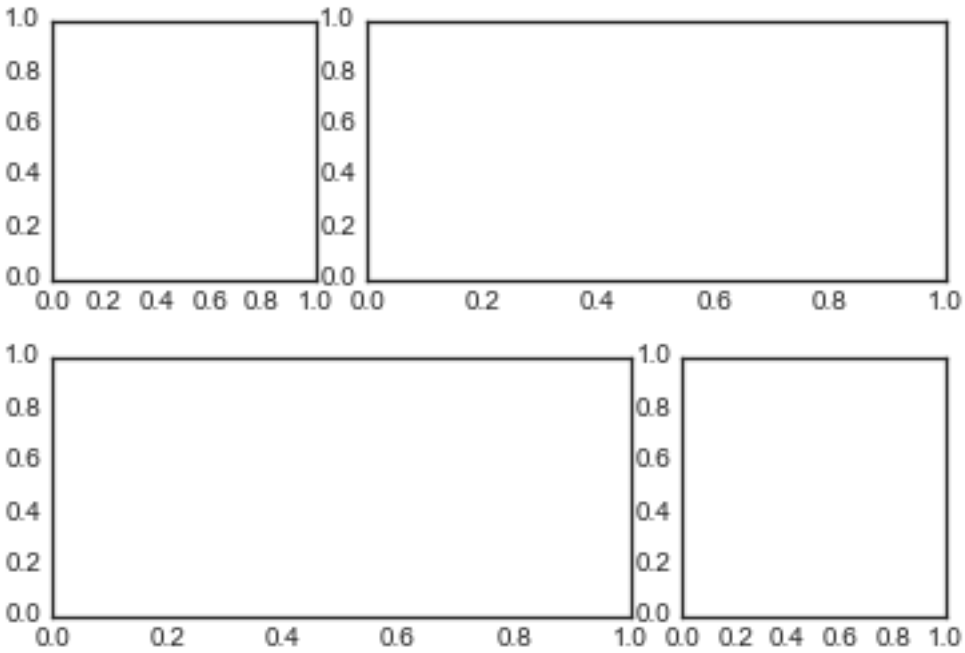
5.3 plt.GridSpec: 복잡한 배치

서브 플롯의 규칙적으로 배치하는 그리드를 넘어 여러 행과 열로 확장 할 경우 plt.GridSpec()사용 plt.GridSpec()자체가 객체 플롯을 만들지는 않고,plt.subplot() 명령어가 인식하는 편리한 이터페이스

[59]:
```
grid = plt.GridSpec(2, 3, wspace=0.2, hspace=0.3)
```

이를 토대로 익숙한 파이썬 슬라이싱 구문을 사용해 각 서브 플롯의 위치와 범위를 지정 가능

[60]:
```
plt.subplot(grid[0, 0])
plt.subplot(grid[0, 1:])
plt.subplot(grid[1, :2])
plt.subplot(grid[1, 2]);
```



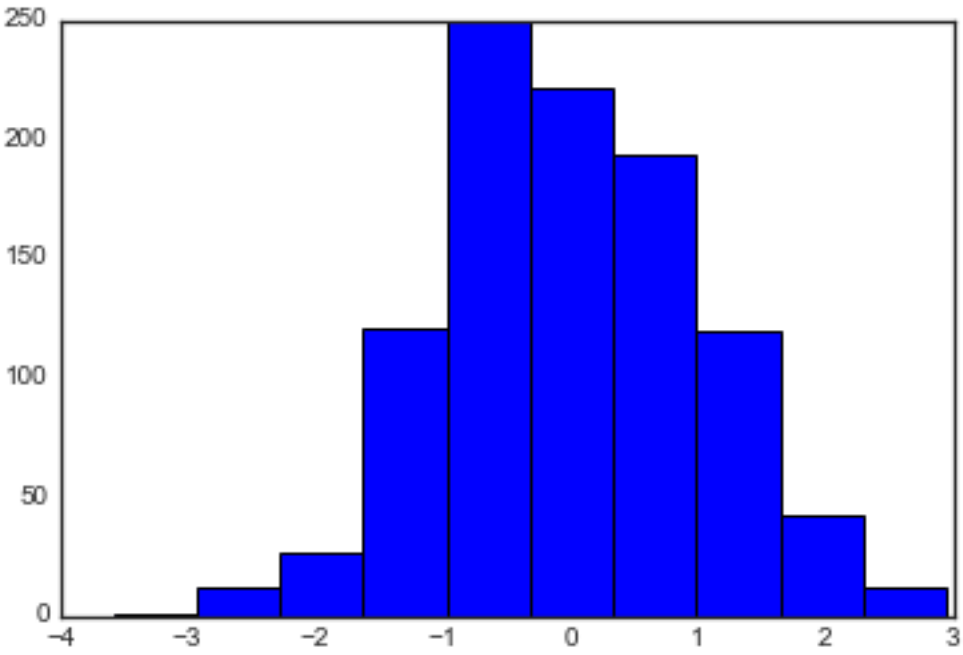
이러한 유형의 유연한 그리드 정렬은 광범위한 용도로 사용 가능 :

6 히스토그램, 구간화, 밀도

데이터세트를 이해하는 가장 좋은 방법은 간단한 히스토 그램을 그려 보는 것이다.

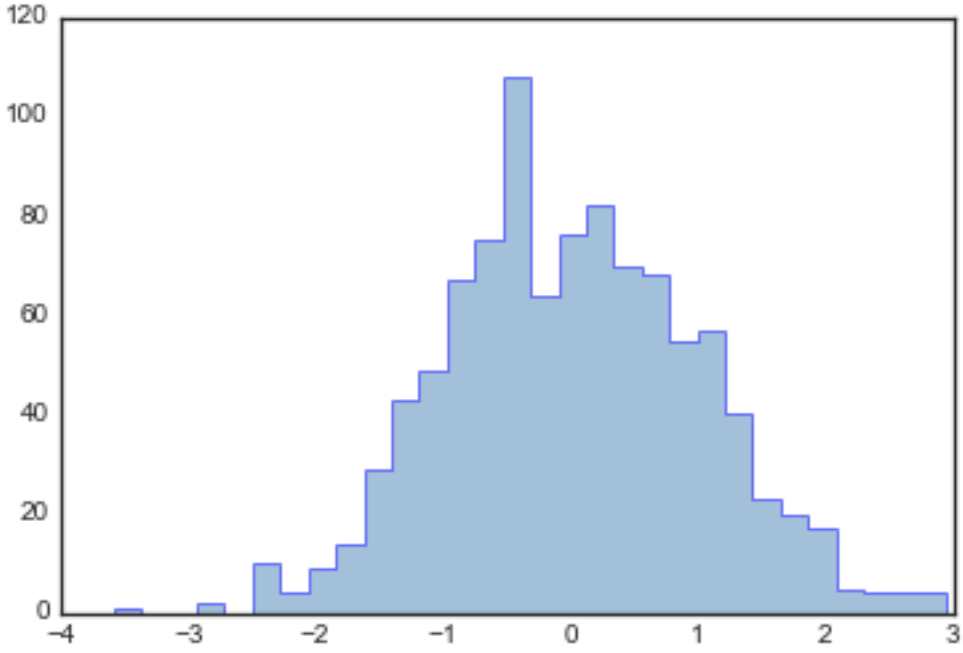
[40]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')

data = np.random.randn(1000)
```

[41]:
```
plt.hist(data);
```



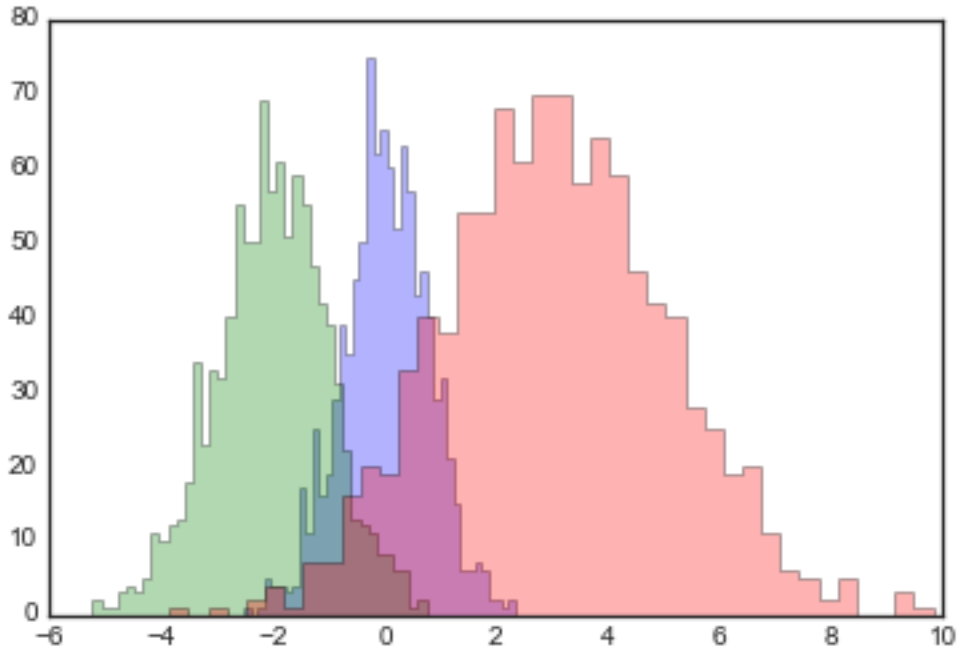
hist() 함수는 계산과 표현 모두 조정할 수 있는 많은 옵션 값을 제공한다. 아래 그림은 위 히스토 그램을 좀더 맞춤 변경 한 예이다.

[42]:
```
plt.hist(data, bins=30, alpha=0.5,
         histtype='stepfilled', color='steelblue',edgecolor='blue');
```



plt.hist독스트링에 그밖에 사용 할 수 있는 맞춤 설정 옵션에 관한 정보가 더 많이 있다. 다양한 분포에 대한 히스토그램을 비교 할때는 histtype='stepfilled' 와 몇몇 투명도alpha 를 결합 하는 것이 때로는 유용하다.

[43]:
```
x1 = np.random.normal(0, 0.8, 1000)
x2 = np.random.normal(-2, 1, 1000)
x3 = np.random.normal(3, 2, 1000)

kwargs = dict(histtype='stepfilled', alpha=0.3,  bins=40)

plt.hist(x1, **kwargs)
plt.hist(x2, **kwargs)
plt.hist(x3, **kwargs);
```



만약 히스토 그램을 계산만 하고 표시할 생각이 없다면 np.histogram() 함수 사용이 가능하다.

[44]:
```
counts, bin_edges = np.histogram(data, bins=5)
print(counts)
```

```
[ 13 148 472 313  54]
```

[ ]:
```
# Create some normally distributed data
mean = [0, 0]
cov = [[1, 1], [1, 2]]
```

```
x, y = np.random.multivariate_normal(mean, cov, 3000).T

# Set up the axes with gridspec
fig = plt.figure(figsize=(6, 6))
grid = plt.GridSpec(4, 4, hspace=0.2, wspace=0.2)
main_ax = fig.add_subplot(grid[:-1, 1:])
y_hist = fig.add_subplot(grid[:-1, 0], xticklabels=[], sharey=main_ax)
x_hist = fig.add_subplot(grid[-1, 1:], yticklabels=[], sharex=main_ax)

# scatter points on the main axes
main_ax.plot(x, y, 'ok', markersize=3, alpha=0.2)

# histogram on the attached axes
x_hist.hist(x, 40, histtype='stepfilled',
            orientation='vertical', color='gray')
x_hist.invert_yaxis()

y_hist.hist(y, 40, histtype='stepfilled',
            orientation='horizontal', color='gray')
y_hist.invert_xaxis()
```

6.1 2차원 히스토그램과 구간화

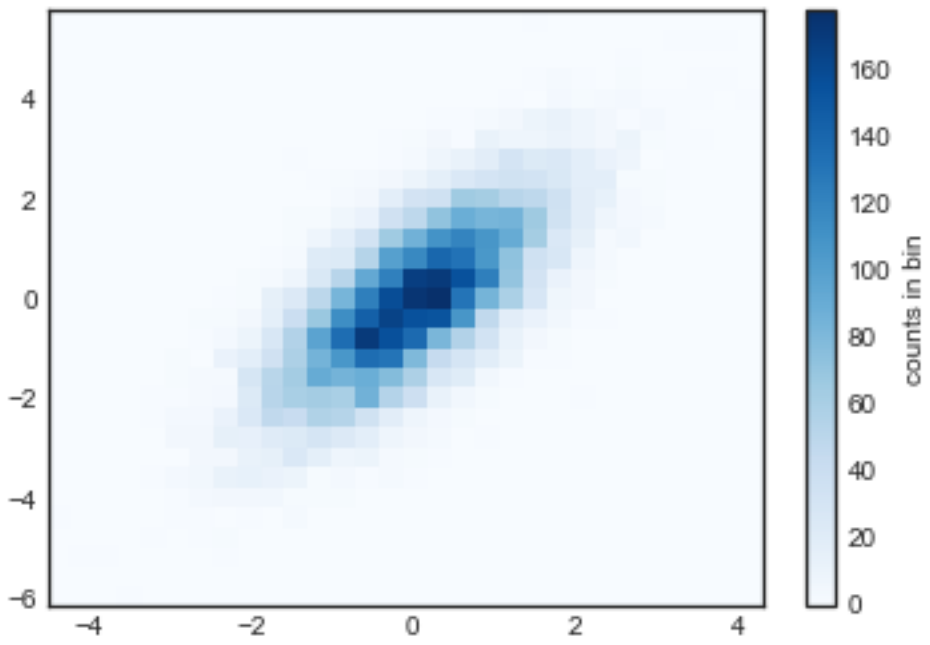
숫자 선을 구간으로 나누어 1차원 히스토그램을 만드는 것처럼 점을 2차원 구간에 나누어 2차원에서도 히스토 그램을 만들 수 있다.

[45]:
```
import numpy as np
mean = [0, 0]
cov = [[1, 1], [1, 2]]
x, y = np.random.multivariate_normal(mean, cov, 10000).T
```

6.1.1 `plt.hist2d`: 차원 히스토그램

Matplotlib의 `plt.hist2d` 함수를 이용:

[46]:
```
plt.hist2d(x, y, bins=30, cmap='Blues')
cb = plt.colorbar()
cb.set_label('counts in bin')
```



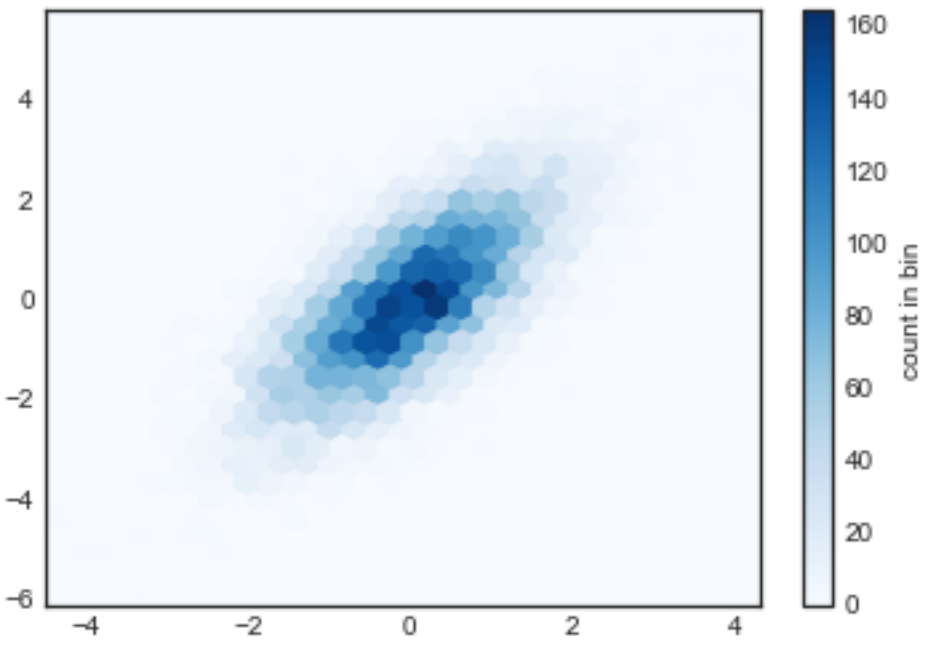
`plt.hist`와 마찬가지로 `plt.hist2d`는 구간을 세밀하게 조정하는 다양한 추가 옵션을 가지고 있으며, 그 옵션은 함수 독스트링에 잘 설명되어 있다. `plt.hist`가 `np.histogram` 에 대응하는 것처럼 `plt.hist2d`는 다음과 같이 사용할 수 있는 `np.histogram2d`에 대응한다.

[47]:
```
counts, xedges, yedges = np.histogram2d(x, y, bins=30)
```

6.1.2 `plt.hexbin`: 육각형 구간화

2차원 히스토그램은 축에 사각형 모자이크를 만든다. 이러한 모자이크에 사용할 만한 자연스러운 모양으로 정육각형도 있다.

[48]:
```
plt.hexbin(x, y, gridsize=30, cmap='Blues')
cb = plt.colorbar(label='count in bin')
```



`plt.hexbin`은 여러 가지옵션을 제공 각점 o 메 대한 가중치를 지정하고 각 구간의 결과를 Numpy 집계 (가중치 평균, 가중치 표준편차 등)로 변경하는 옵션이 있다.

$$f(x) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(x-m)^2}{2\sigma^2}}$$

$$m = 0\text{이고}\ \sigma = 1\text{일 때},\ \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$

[49]:
```
import matplotlib
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(19680801)

# example data
mu = 100  # mean of distribution
```

```
sigma = 15  # standard deviation of distribution
x = mu + sigma * np.random.randn(437)

num_bins = 50

fig, ax = plt.subplots()

# the histogram of the data
n, bins, patches = ax.hist(x, num_bins, density=True)

# add a 'best fit' line
y = ((1 / (np.sqrt(2 * np.pi) * sigma)) *
     np.exp(-0.5 * (1 / sigma * (bins - mu))**2))
ax.plot(bins, y, '--')
ax.set_xlabel('Smarts')
ax.set_ylabel('Probability density')
ax.set_title(r'Histogram of IQ: $\mu=100$, $\sigma=15$')

# Tweak spacing to prevent clipping of ylabel
fig.tight_layout()
plt.show()
```