9. 그래프와 시각화

정보 시각화는 데이터 분석에서 무척 중요한 일 중 하나다. 시각화는 특잇값을 찾아내거나, 데이터 변형이 필요한지 알아보거나, 모델에 대한 아이디어를 찾기 위한 과정의 일부이기도 하다. 혹자에게는 웹상에서 구현되는 시각화가 최종 목표일 수도 있다. 파이썬은 다양한 시각화 도구를 구비하고 있지만, 이 책에서는 matplotlib과 matplotlib 기반의 도구들을 우선적으로 살펴보겠다.

matplotlib은 주로 2D 그래프를 위한 데스크톱 패키지로, 출판물 수준의 그래프를 만들어내도록 설계되었다. matplotlib은 모든 OS의 다양한 GUI 백엔드를 지원하고 있으며 PDF, SVG, JPG, PNG, BMP, GIF 등 일반적으로 널리 사용되는 벡터 포맷과 래스터 포맷으로 그래프를 저장할 수 있다.

시간이 흐름에 따라 내부적으로 matplotlib을 사용하는 데이터 시각화 도구들이 생겨났는데 그중 하나가 이 장 후반에 살펴볼 seaborn 라이브러리다.

이 장에 포함된 코드 예제를 실행시키는 가장 손쉬운 방법은 주피터 노트북의 대화형 시각화 기능을 사용하는 것이다. 주피터 노트북을 실행시키자.

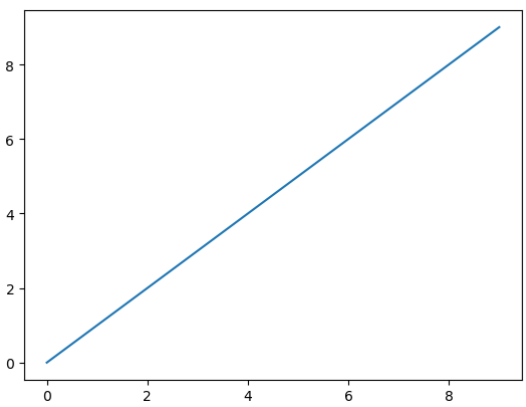
9.1 matplotlib API 간략하게 살펴보기

이 책에서는 matplotlib을 아래와 같은 네이밍 컨벤션으로 임포트하겠다.

```
In [116]: import matplotlib.pyplot as plt
```

주피터 노트북 환경에서 %matplotlib notebook 을 실행한 다음 (IPython의 경우 %matplotlib) 간단한 그래프를 그려보자.

```
In [2]: %matplotlib
Using matplotlib backend: Qt5Agg

In [3]: import numpy as np

In [4]: data = np.arange(10)

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

In [6]: plt.plot(data)
Out[6]: [<matplotlib.lines.Line2D at 0x1d9a1aa9e08>]
```



9.1.1 figure와 서브플롯

matplotlib에서 그래프는 Figure 객체 내에 존재한다. 그래프를 위한 새로운 figure는 plt.figure를 사용해서 생성할 수 있다.

```
In [7]: fig = plt.figure()
```

Ipython에서 실행했다면 빈 윈도우가 나타날 것이다. plt.figure에는 다양한 옵션이 있는데 그중 figsize는 파일에 저장할 경우를 위해 만들려는 figure의 크기와 비율을 지정할 수 있다.

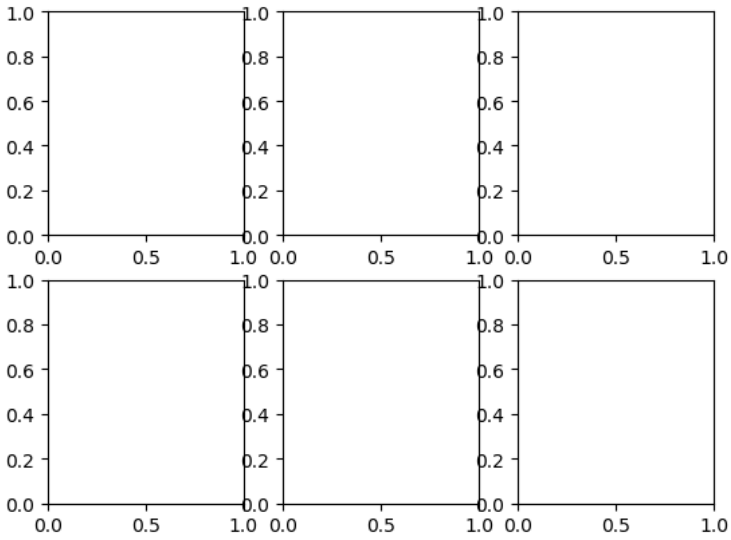
빈 figure로는 그래프를 그릴 수 없다. add_subplot을 사용해서 최소 하나 이상의 subplots를 생성해야 한다.

```
In [8]: ax1 =fig.add_subplot(2,2,1)
```

위 코드는 2x2 크기이고 4개의 서브플롯 중에서 첫 번째를 선택하겠다는 의미다. (서브플롯은 1부터 숫자가 매겨진다). 다음처럼 2개의 서브플롯을 더 추가하면 다음과 같은 모양이 된다.

matplotlib 문서에서 여러 가지 그래프 종류를 확인할 수 있다.

특정한 배치에 맞춰 여러 개의 서브플롯을 포함하는 figure를 생성하는 일은 흔히 접하는 업무인데 이를 위한 plt.subplot라는 편리한 메서드가 있다. 이 메서드는 NumPy 배열과 서브플롯 객체를 새로 생성하여 반환한다.



```
In [12]: fig, axes = plt.subplots(2,3)

In [13]: axes
Out[13]:
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001D9A2B44708>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D9A851DC08>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D9A8823948>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001D9A885BDC8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D9A8892588>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D9A88C2D08>]],
      dtype=object)
```

axes 배열은 axes[0,1] 처럼 2차원 배열로 쉽게 색인될 수 있어서 편리하게 사용할 수 있다.

서브플롯이 같은 x축 혹은 y축을 가져야 한다면 각각 sharex와 sharey를 사용해서 지정할 수 있다.

같은 범위 내에서 데이터를 비교해야 할 경우 특히 유용하다. 그렇지 않으면 matplotlib은 각 그래프의 범위를 독립적으로 조정한다. 이 메서드에 대한 자세한 내용은 다음과 같다.

pyplot.subplots 옵션

인자	설명
nrows	서브플롯의 로우 수
ncols	서브플롯의 컬럼 수
sharex	모든 서브플롯이 같은 x축 눈금을 사용하도록 한다.(xlim 값을 조절하면 모든 서브플롯에 적용된다.)
sharey	모든 서브플롯이 같은 y축 눈금을 사용하도록 한다.(ylim 값을 조절하면 모든 서브플롯에 적용된다.)
subplot_kw	add_subplot을 사용해서 각 서브플롯을 생성할 때 사용할 키워드를 담고 있는 사전
**fig_kw	figure를 생성할 때, 사용할 추가적인 키워드 인자.

서브플롯 간의 간격 조절하기

matplotlib은 서브플롯 간에 적당한 간격과 여백을 추가해준다. 이 간격은 전체 그래프의 높이와 너비에 따라 상대적으로 결정된다. 그러므로 프로그램을 이용하든 아니면 직접 GUI 윈도우의 크기를 조정하든 그래프의 크기가 자동으로 조절된다. 서브플롯 간의 간격은 Figure 객체의 subplot_adjust메서드를 사용해서 쉽게 바꿀 수 있다.

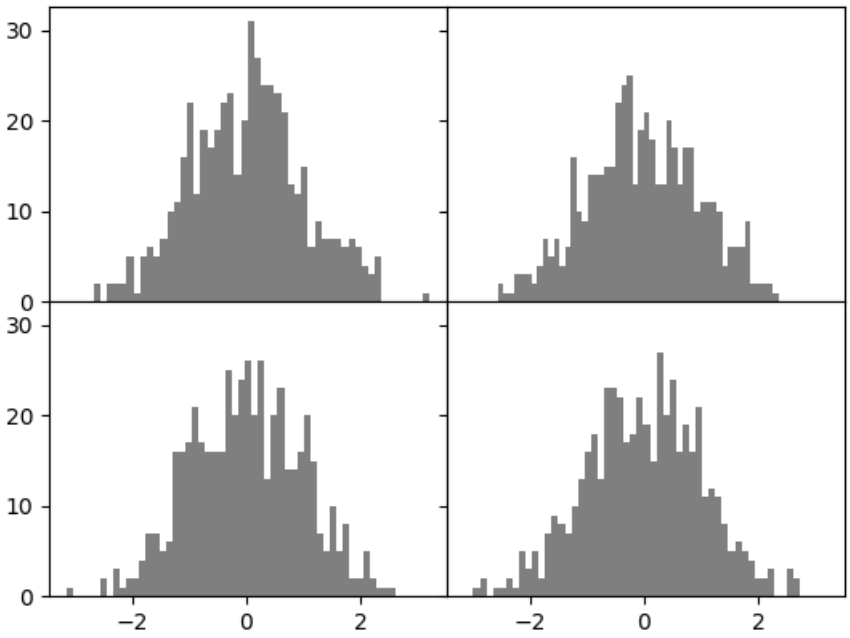
subplots_adjust 메서드는 최상위 함수로도 존재한다.

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

wspace와 hspace와 서브플롯 간의 간격을 위해서 figure의 너비와 높이에 대한 비율을 조절한다.

다음 코드는 서브플롯 간의 간격을 주지 않는 그래프를 생성하는 코드다.

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

In [25]: 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)
```



그래프를 그렸을 때 축이름이 겹치는 경우가 있다. matplotlib은 그래프에서 이름이 겹치는지 검사하지 않기 때문에 이와 같은 경우에는 눈금 위치와 눈금 이름을 명시적으로 직접 지정해야 한다.

9.1.2 색상, 마커, 선스타일

matplotlib에서 가장 중요한 plot 함수는 x와 y 좌푯값이 담긴 배열과 추가적으로 색상과 선스타일을 나타내는 축약 문자열을 인자로 받는다. 예를 들어 녹색 점선으로 그려진 x 대 y 그래프는 아래처럼 나타낼 수 있다.

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

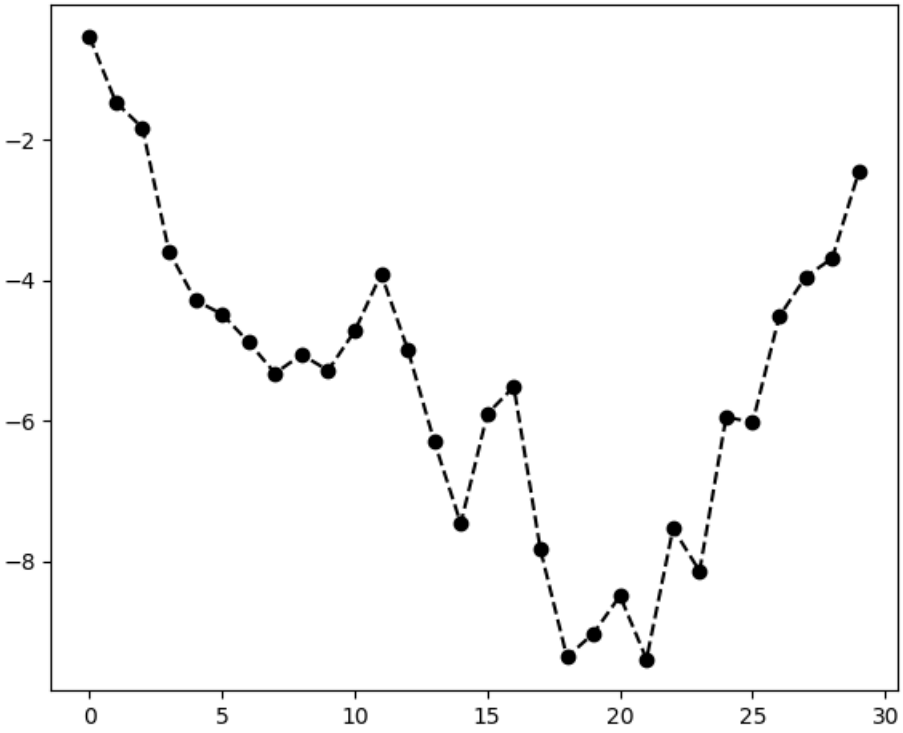
이와 같이 문자열로 색상과 선 스타일을 지정하는 방법은 편의를 위해 제공되고 있는데, 실무에서 프로그램으로 그래프를 생성할 때는 그래프를 원하는 형식으로 생성하기 위해 문자열을 지저분하게 섞어 쓰고 싶지 않을 것이다.

위에서 만든 그래프는 아래처럼 좀 더 명시적인 방법으로 쓰는 것이 좋다.

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

흔히 사용되는 색상을 위해 몇 가지 색상 문자열이 존재하지만 RGB 값(예: #CECECE)을 직접 지정해서 색상표에 있는 어떤 색상이라도 지정할 수 있다. 선 스타일에 대한 전체 목록은 plot 메서드의 도움말을 참고하자.

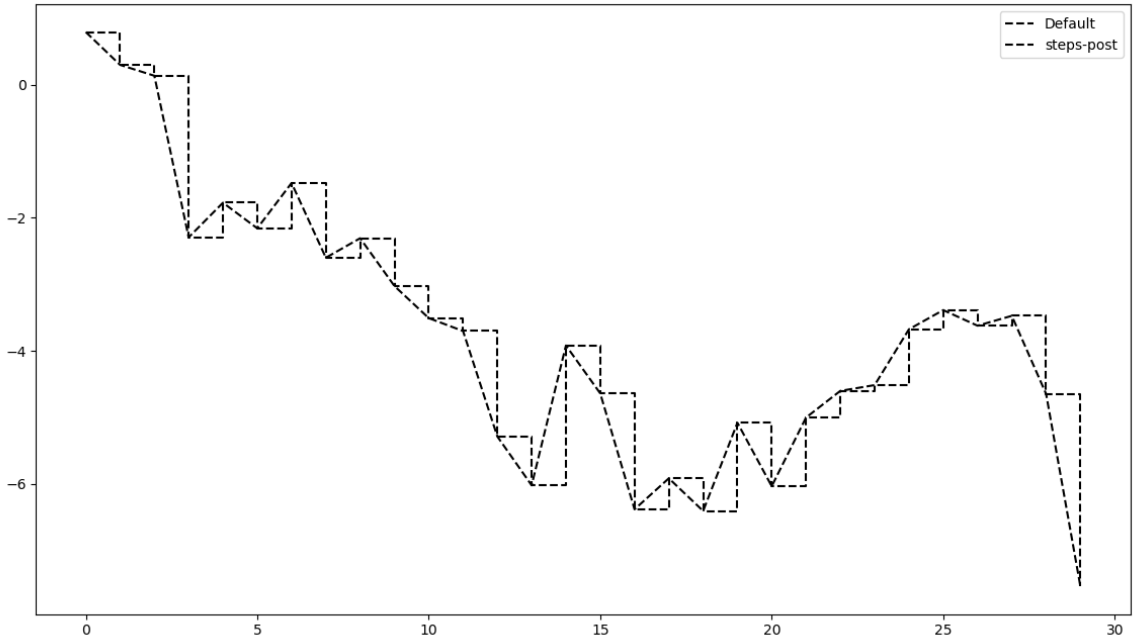
선그래프는 특정 지점의 실제 데이터를 돋보이게 하기 위해 **마커**를 추가하기도 한다. matplotlib은 점들을 잇는 연속된 선그래프를 생성하기 때문에 어떤 지점에 마커를 설정해야 하는지 확실치 않은 경우가 종종 있다. 마커도 스타일 문자열에 포함시킬 수 있는데 색상 다음에 마커 스타일이 오고 그 뒤에 선 스타일을 지정한다.



이 역시 좀 더 명시적인 방법으로 표현할 수 있다. 선그래프를 보면 일정한 간격으로 연속된 지점이 연결되어 있다. 이 역시 drawstyle 옵션을 이용해서 바꿀 수 있다.

```
In [55]: plt.plot(randn(30).cumsum(), color='k',linestyle = 'dashed',
    ...:     marker = 'o')
Out[55]: [<matplotlib.lines.Line2D at 0x1d9ae022dc8>]
```

```
In [56]: data = np.random.randn(30).cumsum()

In [57]: plt.plot(data,'k--', label='Default')
Out[57]: [<matplotlib.lines.Line2D at 0x1d9abc8ff48>]

In [60]: plt.plot(data,'k--', drawstyle = 'steps-post', label='steps-po
    ...: st')
Out[60]: [<matplotlib.lines.Line2D at 0x1d9a84fa408>]

In [61]: plt.legend(loc='best')
Out[61]: <matplotlib.legend.Legend at 0x1d9ae027848>
```



이 코드를 실행해보면 <matplotlib.legend.Legend at 0x1d9ae027848>과 같은 결과를 확인할 수 있다. matplotlib은 방금 추가된 그래프의 하위 컴포넌트에 대한 레퍼런스 객체를 리턴한다. 이 결과는 무시해도 된다. 여기서는 label 인자로 plot을 전달했기 때문에 plt.legend를 이용해서 각 선그래프의 범례를 추가할 수 있다.

NOTE 범례를 생성하려면 그래프를 그릴 때 label 옵션 지정 여부와 상관없이 반드시 plt.legend를 호출해야 한다. (축에 대한 범례를 추가하려면 ax.legend를 호출하자.)

9.1.3 눈금, 라벨, 범례

그래프를 꾸미는 방법은 크게 2가지가 있다.

pyplot 인터페이스를 사용해서 순차적으로 꾸미든가(즉, matplotlib.pylib) 아니면

matplotlib이 제공하는 API를 사용해서 좀 더 객체지향적인 방법으로 꾸미는 것이다.

pyplot 인터페이스는 대화형 사용에 맞춰 설계되었으며, xlim, xticks, sticklabels 같은 메서드로 이뤄져 있다. 이런 메서드로 표의 범위를 지정하거나 눈금 위치. 눈 금 이름을 조절할 수 있다.

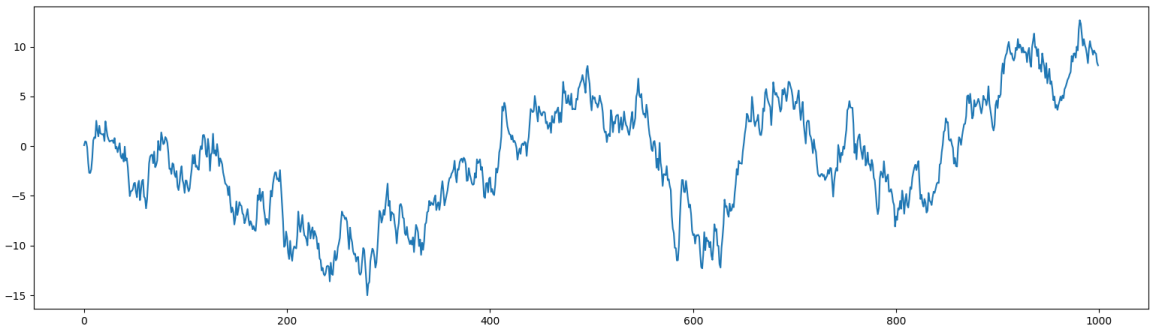
- **아무런 인자 없이 호출하면 현재 설정되어 있는 매개변수의 값을 반환한다. plt.xlim 메서드는 현재 x축의 범위를 반환한다.**
- **인자를 전달하면 매개변수의 값을 설정한다. 예를 들어 plt.xlim([0,10])을 호출하면 x축의 범위가 0부터 10까지로 설정한다.**

이 모든 메서드는 현재 활성화된 혹은 가장 최근에 생성된 AxesSubplot 객체에 대해 동작한다. 위에서 소개한 모든 메서드는 서브플롯 객체의 set/get 메서드로도 존재하는데, xlim이라면 ax.get_xlim과 ax.set_xlim 메서드가 존재한다. 나는 개인적으로 명시적인 것을 선호하기에 (그리고 특히 여러 개의 서브플롯을 다룰 때는) 서브플롯 인스턴스 메서드를 사용한다.

제목, 축 이름, 눈금, 눈금 이름 설정하기

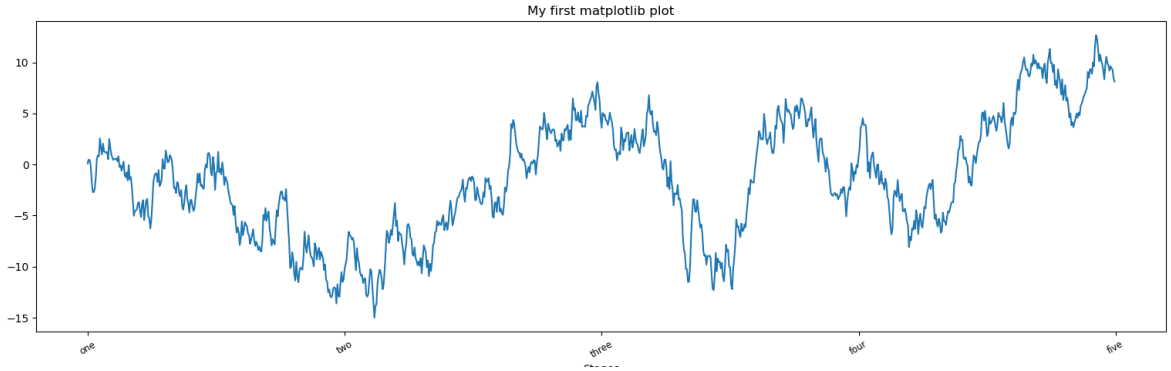
축을 꾸미는 방법을 설명하기 위해 무작위 값으로 간단한 그래프를 하나 생성해보겠다.

```
In [64]: ax = fig.add_subplot(1,1,1)

In [65]: fig = plt.figure()

In [66]: ax = fig.add_subplot(1,1,1)

In [67]: ax.plot(np.random.randn(1000).cumsum())
Out[67]: [<matplotlib.lines.Line2D at 0x1d9a8b09508
```



x축의 눈금을 변경하기 위한 가장 쉬운 방법은 set_xticks와 set_xticklabels 메서드를 사용하는 것이다. set_xticks 메서드는 전체 데이터 범위를 따라 눈금을 어디에 배치할지 지정한다. 기본적으로 이 위치에 눈금 이름이 들어간다. 하지만 다른 눈금 이름을 지정하고 싶다면 set_xticklabels를 사용하면 된다.

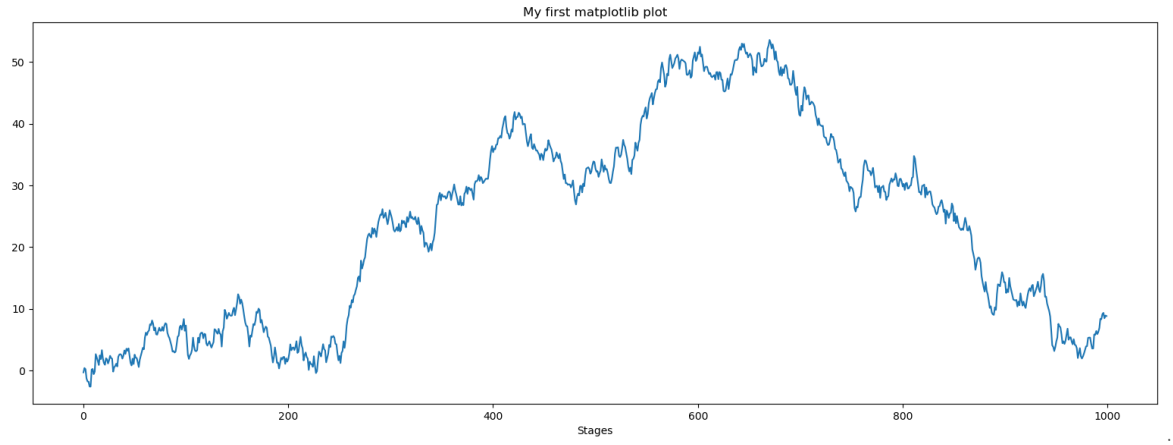
```
In [68]: ticks = ax.set_xticks([0,250,500,750,1000])

In [69]: labels = ax.set_xticklabels(['one','two','three','four','five'
    ...: ], rotation=30,fontsize = 'small')

In [70]: ax.set_title('My first matplotlib plot')
Out[70]: Text(0.5, 1, 'My first matplotlib plot')

In [71]: ax.set_xlabel('Stages')
Out[71]: Text(0.5, 15.603891973024515, 'Stages')
```



x대신 y를 써서 같은 과정을 y축에 대해 진행할 수도 있다. axes 클래스는 플롯의 속성을 설정할 수 있도록 set메서드를 제공한다. 위 예제는 아래와 같이 작성할 수도 있다.

```
In [72]: fig = plt.figure()

In [73]: ax = fig.add_subplot(1,1,1)

In [74]: ax.plot(np.random.randn(1000).cumsum())
Out[74]: [<matplotlib.lines.Line2D at 0x1d9adce4648>]
```

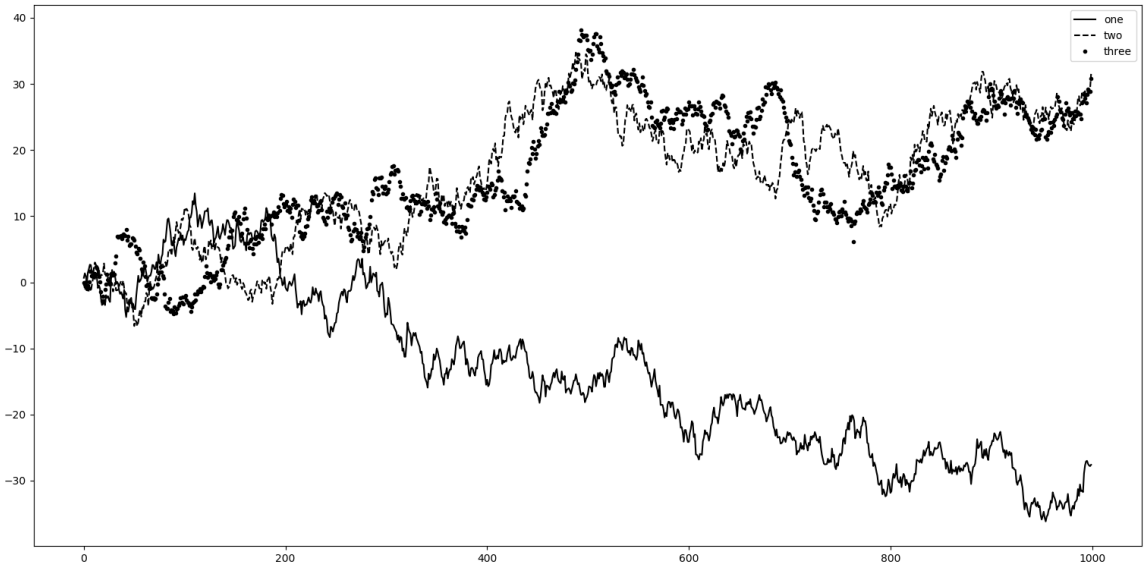
```
In [75]: props = {
    ...:              'title':'My first matplotlib plot',
    ...:              'xlabel':'Stages'}
    ...: ax.set(**props)
Out[75]:
[Text(0.5, 22.972222222222214, 'Stages'),
 Text(0.5, 1, 'My first matplotlib plot')]
```



범례 추가하기

범례는 그래프 요소를 확인하기 위해 중요한 요소다. 범례를 추기하는 몇 가지 방법이 있는데, 가장 쉬운 방법은 각 그래프에 label 인자를 넘기는 것이다.

```
In [76]: from numpy.random import randn

In [77]: fig = plt.figure(); ax = fig.add_subplot(1,1,1)

In [79]: ax.plot(randn(1000).cumsum(),'k',label='one')
Out[79]: [<matplotlib.lines.Line2D at 0x1d9a8b443c8>]

In [80]: ax.plot(randn(1000).cumsum(),'k--',label='two')
Out[80]: [<matplotlib.lines.Line2D at 0x1d9af866dc8>]

In [81]: ax.plot(randn(1000).cumsum(),'k.',label='three')
Out[81]: [<matplotlib.lines.Line2D at 0x1d9a8b73dc8>]

In [82]: ax.legend(loc='best')
Out[82]: <matplotlib.legend.Legend at 0x1d9af1a5048>
```



legend 메서드에는 범례 위치를 지정하기 위한 loc 인자를 제공한다. legend 메서드의 문저에서 더 자세한 정보를 확인할 수 있다.

loc는 범례를 그레프에서 어디에 위치시킬지 지정해주는 인자다. 까다로운 사람이 아니라면 최대한 방해가 되지 않는 곳에 두는 'best' 옵션만으로 충분할 것이다. 범례에서 제외하고 싶은 요소가 있다면 label 인자를 넘기지 않거나 label = '_nolegend_' 옵션을 사용하면 된다.

9.1.4 주석과 그림 추가하기

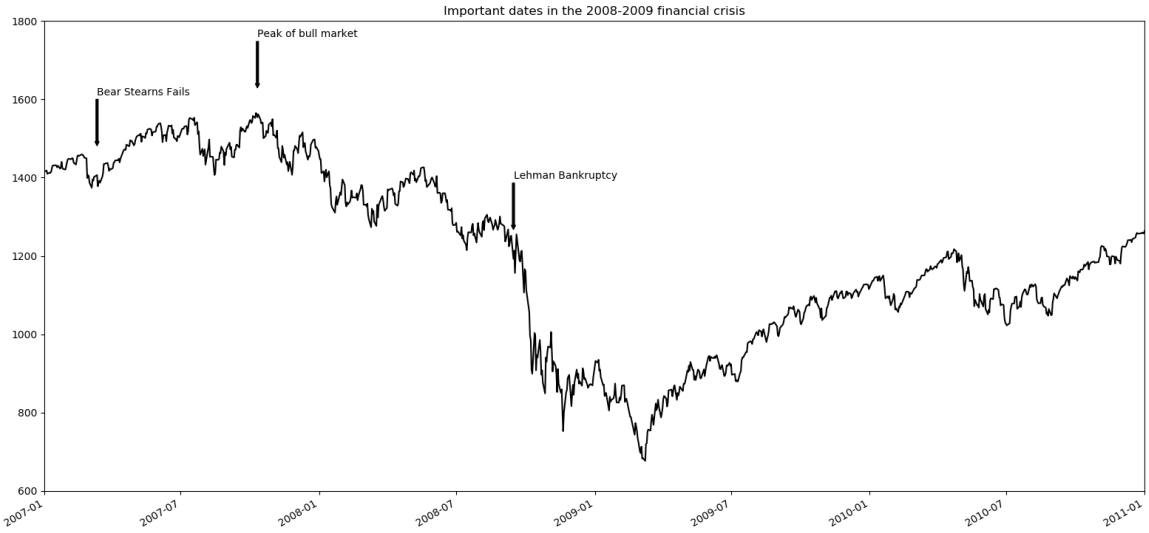
일반적인 그래프에 글자나 화살표 혹은 다른 도형으로 자기만의 주석을 그리고 싶은 경우가 있다. 주석과 글자는 text, arrow, annotate 함수를 이용해서 추가할 수 있다. text 함수는 그래프 내의 주어진 좌표 (x,y)에 부가적인 스타일로 글자를 그려준다.

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

주석은 글자와 화살표를 함께 써서 그릴 수 있는데, 예를 들어 야후! 파이낸스에서 얻은 2007년 부터의 S&P 500 지수 데이터로 그래프를 생성하고 2008~2009년 사이에 있었던 재정위기 중 중요한 날짜를 주석으로 추가해보자.

```
In [94]: fig = plt.figure()

In [95]: ax = fig.add_subplot(1,1,1)

In [96]: data = pd.read_csv('spx.csv',index_col = 0, parse_dates = True
    ...: )

In [97]: spx = data['SPX']

In [98]: spx.plot(ax=ax, style = 'k-')
Out[98]: <matplotlib.axes._subplots.AxesSubplot at 0x1d9b4389a88>

In [99]: crisis_data = [
    ...:                (datetime(2007,10,11),'Peak of bull market'),
    ...:                (datetime(2007,3,12),'Bear Stearns Fails'),
    ...:                (datetime(2008,9,15),'Lehman Bankruptcy')]
```

```
In [102]: 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'
     ...:               )


#2007-2010 구간으로 확대
In [103]: ax.set_xlim(['1/1/2007','1/1/2011'])
Out[103]: (732677.0, 734138.0)

In [104]: ax.set_ylim([600,1800])
Out[104]: (600, 1800)

In [105]: ax.set_title('Important dates in the 2008-2009 financial cris
     ...: is')
Out[105]: Text(0.5, 1, 'Important dates in the 2008-2009 financial crisis')
```



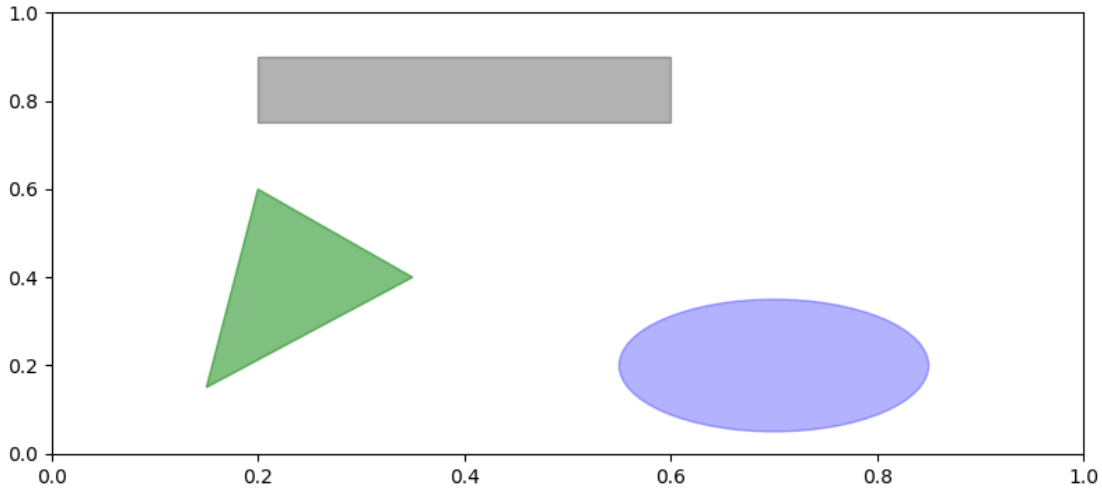
이 그래프에서 알고 넘어가야 할 몇몇 중요한 내용이 있는데, ax.annotate 메서드를 이용해서 x,y 좌표로 지정한 위치에 라벨을 추가했으며 set_xlim과 set_ylim 메서드를 이용해서 그래프의 시작과 끝 경계를 직접 지정했다. 마지막으로 ax.set_title 메서드로 그래프의 제목을 지정했다.

도형을 그리려면 좀 더 신경을 써야 한다. matplotlib은 일반적인 도형을 표현하기 위한 patches 라는 객체를 제공한다. 그중 Rectangle과 Circle 같은 것은 matplotlib.pyplot 에서도 찾을 수 있지만 전체 모음은 matplotlib.patches에 있다.

그래프에 도형을 추가하려면 patches 객체인 shp를 만들고 서브플롯에 ax.add_patch(shp)를 호출한다.

```
In [106]: fig = plt.figure()

In [107]: ax = fig.add_subplot(1,1,1)

In [108]: rect = plt.Rectangle((0.2,0.75),0.4,0.15,color = 'k', alpha =
     ...: 0.3)

In [109]: circ = plt.Circle((0.7,0.2),0.15, color = 'b', alpha = 0.3 )

In [110]: pgon = plt.Polygon([[0.15,0.15],[0.35,0.4],[0.2,0.6]],color='
```

```
      ...: g',alpha = 0.5)

In [111]: ax.add_patch(rect)
Out[111]: <matplotlib.patches.Rectangle at 0x1d9b47f6808>

In [112]: ax.add_patch(circ)
Out[112]: <matplotlib.patches.Circle at 0x1d9b431f308>

In [113]: ax.add_patch(pgon)
Out[113]: <matplotlib.patches.Polygon at 0x1d9b4801148>
```



9.1.5 그래프를 파일로 저장하기

활성화된 figure는 plt.savefig 메서드를 이용해서 파일로 저장할 수 있다. 이 메서드는 figure 객체의 인스턴스 메서드인 savefig와 동일하다. figure를 SVG 포맷으로 저장하려면 다음처럼 하면 된다.

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

파일 종류는 확장자로 결정된다. 그러므로, .svg 대신에 .pdf를 입력하면 PDF파일을 얻는다. 출판용 그래픽 파일을 생성할 때 나가 자주 사용하는 몇 가지 중요한 옵션이 있는데, 바로 dpi와 bbox_inches이다. dpi는 인치당 도트 해상도를 조절하고 bbox_inches는 실제 figure 둘레의 공백을 잘라낸다.

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

savefig 메서드는 파일에 저장할 뿐만 아니라 BytesIO 처럼 파일과 유사한 객체에 저장하는 것도 가능하다.

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

다음 표는 savefig의 다른 옵션을 정리한 것이다.

Figure.savefig 옵션

인자	설명
fname	파일 경로나 파이썬의 파일과 유사한 객체를 나타내는 문자열, 저장되는 포맷은 파일 확장자를 통해 결정된다. 예를 들어.pdf는 pdf 포멧
dpi	figure의 인치당 도트 해상도, 기본값은 100이며, 설정 가능하다.
facecolor.edgecolor	서브플롯 바깥 배경 색상, 기본값은 'w(흰색)'이다.
format	명시적인 파일 포맷('png', 'pdf', 'svg', ..)
bbox_inches	figure에서 저장할 부분. 만약 'tight'를 지정하면 figure 둘레에 비어 있는 공간을 모두 제거한다.

9.1.6 matplotlib 설정

matplotlib은 출판물용 그래프를 만드는 데 손색이 없는 기본 설정과 색상 스키마를 함께 제공한다.

다행스럽게도 거의 모든 기본 동작은 많은 전역 인자를 통해 설정 가능한데, 그래프 크기, 서브플롯 간격, 색상, 글자크기, 격자 스타일과 같은 것들을 설정 가능하다.

matplotlib의 환경 설정 시스템은 두 가지 방법으로 다룰 수 있는데, 첫 번째는 rc 메서드를 사용해서 프로그래밍적으로 설정하는 방법이다. 예를 들어 figure 크기를 10 x 10으로 전역 설정해두고 싶다면 다음 코드를 실행한다.

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

rc 메서드의 첫 번째 인자는 설정하고자 하는 'figure', 'axes', 'xtick', 'ytick', 'grid', 'legend' 및 다른 컴포넌트의 이름이다. 그 다음으로 설정할 값에 대한 키워드 인자를 넘기게 된다. 이 옵션을 쉽게 작성하려면 파이썬의 사전 타입을 사용한다.

```
In [126]: font_options = {'family':'monospace',
     ...:                 'weight':'bold',
     ...:                 'size':'small'}

plt.rc('font',**font_options)
```

9.2 pandas에서 seaborn으로 그래프 그리기

matplotlib은 사실 꽤 저수준의 라이브러리다. 데이터를 어떻게 보여줄 것인지 부터(선그래프, 막대그래프, 산포도 등) 범례와 제목, 눈금 라벨, 주석 같은 기본 컴포넌트로 그래프를 작성해야 한다.

pandas를 사용하다 보면 로우와 컬럼 라벨을 가진 다양한 컬럼 데이터를 다룬다.

pandas는 Series와 DataFrame 객체를 간단하게 시각화할 수 있는 내장 메서드를 제공한다. 다른 라이브러리로는 통계 그래픽 라이브러리인 seaborn이 있다. seaborn은 흔히 사용하는 다양한 시각화 패턴을 쉽게 구현할 수 있도록 도와준다.

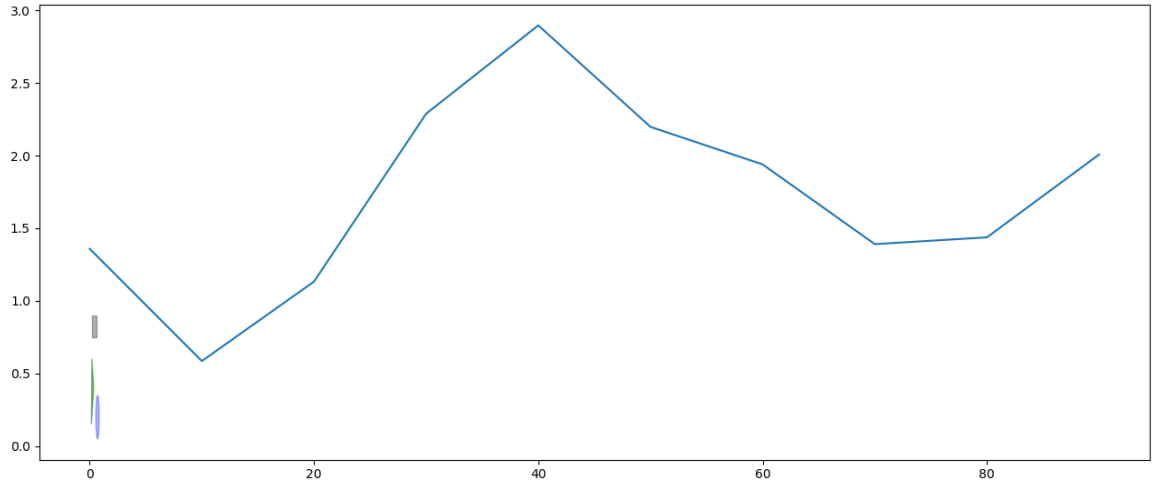
Tip) seaborn 라이브러리를 임포트하면 더 나은 가독성과 미려함을 위해 matplotlib의 기본 컬러 스킴과 플롯 스타일을 변경했다. 일부 독자는 seaborn API를 사용하지 않더라도 일반적인 matplotlib 그래프의 스타일을 개선하기 위한 간편한 방법으로 seaborn 라이브러리를 임포트하는 것을 선호할지도 모르겠다.

9.2.1 선그래프

Series와 DataFrame은 둘 다 plot 메서드를 이용해 다양한 형태의 그래프를 생성할 수 있다.

기본적으로 plot 메서드는 선그래프를 생성한다.

```
In [128]: s = pd.Series(np.random.randn(10).cumsum(),
     ...:                index = np.arange(0,100,10))

In [129]: s.plot()
Out[129]: <matplotlib.axes._subplots.AxesSubplot at 0x1d9b4325a48>
```



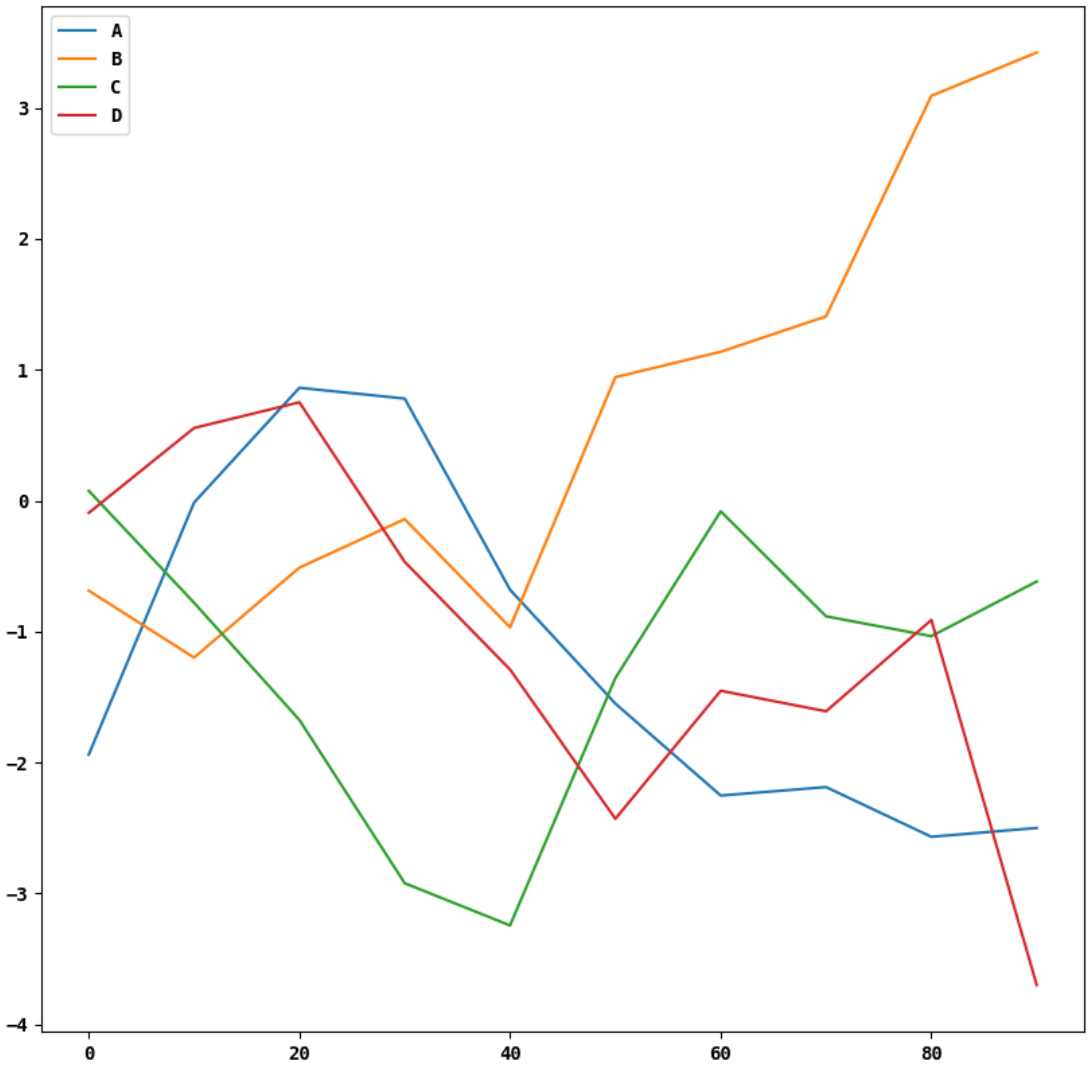
Series 객체의 색인은 matplotlib 에서 그래프를 생성할 때 x축으로 해석되며 use_index=False 옵션을 넘겨서 색인을 그래프의 축으로 사용하는 것을 막을 수 있다. x축의 눈금과 한계는 xticks와 xlim옵션으로 조절할 수 있으며 y축 역시 yticks와 ylim 옵션으로 조절할 수 있다.

다음 표에서 사용가능한plot메서드의 옵션을 정리했다. 그중 몇 가지는 설명하겠다.

대부분의 pandas 그래프 메서드는 부수적으로 ax인자를 받는데, 이 인자는 matplotlib의 서브플롯 객체가 될 수 있다. 이를 이용해 그리드 배열 상에서 서브플롯의 위치를 좀 더 유연하게 할 수 있다.

DataFrame의 plot메서드는 하나의 서브플롯 안에 각 컬럼별로 선 그래프를 그리고 자동적으로 범례를 생성한다.

```
In [131]: df = pd.DataFrame(np.random.randn(10,4).cumsum(0),
     ...:                   columns = ['A','B','C','D'],
     ...:                   index = np.arange(0,100,10))

In [132]: df.plot()
Out[132]: <matplotlib.axes._subplots.AxesSubplot at 0x1d9b47f6b08>
```



plot 속성에는 다양한 종류의 그래프 패밀리가 존재한다. 예를 들어 df.plot()은 df.plot.line()과 동일하다. 이런 메서드에 대해서는 잠시 뒤에 알아보자.

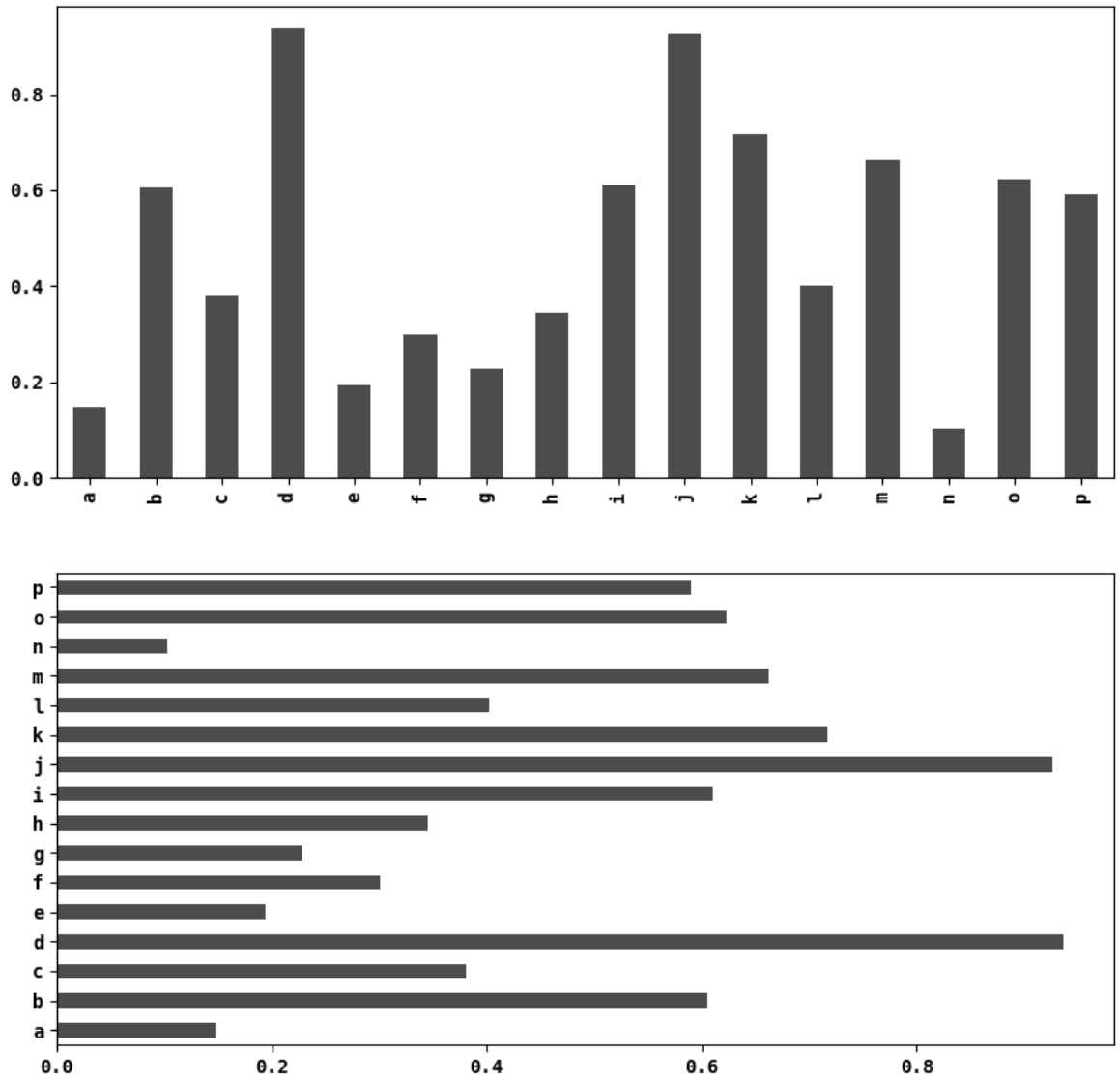
Note_ plot메서드에 전달할 수 있는 부수적인 키워드 인자들은 그대로 matplotlib의 함수로 전달된다. 따라서 matplotlib API를 자세히 공부하면 더 다양한 방식으로 그래프를 꾸밀 수 있다.

9.2.2 막대그래프

plot.bar()와 plot.barh()는 각각 수직막대그래프와 수평막대그래프를 그린다.

이 경우 Series 또는 DataFrame의 색인은 수직막대그래프(bar)인 경우 x눈금, 수평막대그래프(barh)인 경우 y눈금으로 사용된다.

```
In [142]: fig, axes = plt.subplots(2,1)

In [143]: data = pd.Series(np.random.rand(16), index =list('abcdefghijk
     ...: lmnop'))

In [144]: data.plot.bar(ax=axes[0],color = 'k', alpha = 0.7)
Out[144]: <matplotlib.axes._subplots.AxesSubplot at 0x1d9b429ba88>

In [145]: data.plot.barh(ax=axes[1],color = 'k', alpha = 0.7)
Out[145]: <matplotlib.axes._subplots.AxesSubplot at 0x1d9b4360c48>
```



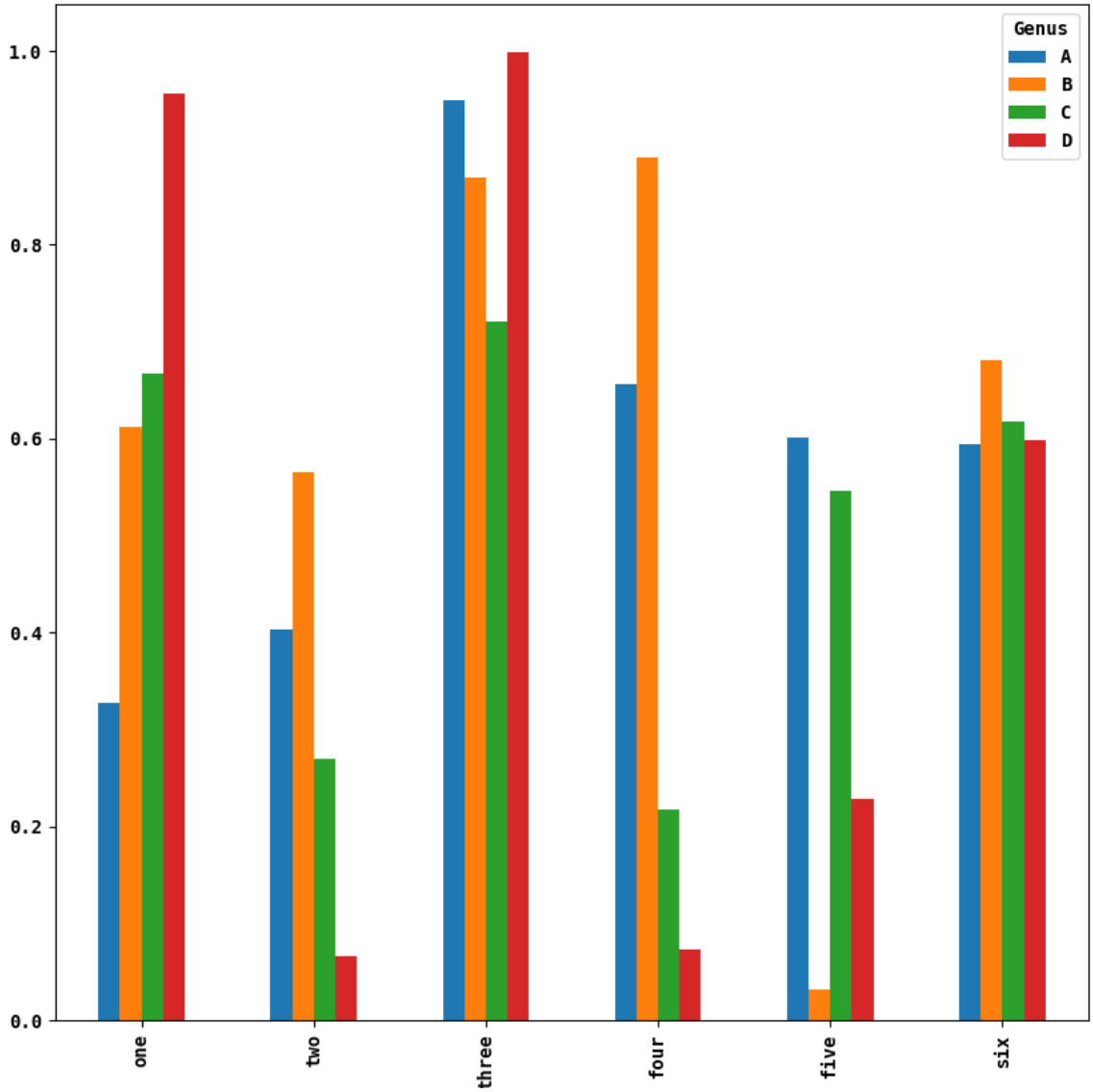
color = 'k' 옵션과 alpha = 0.7 옵션은 그래프를 검은색으로 그리고 투명도를 지정한 것이다.

Note_ 막대그래프를 그릴 때 유용한 방법은 Series의 value_counts 메서드 (s.value_counts().plot.bar())를 이용해서 값의 빈도를 그리는 것이다.

DataFrame에서 막대그래프는 각 로우의 값을 함께 묶어서 하나의 그룹마다 각각의 막대를 보여준다.

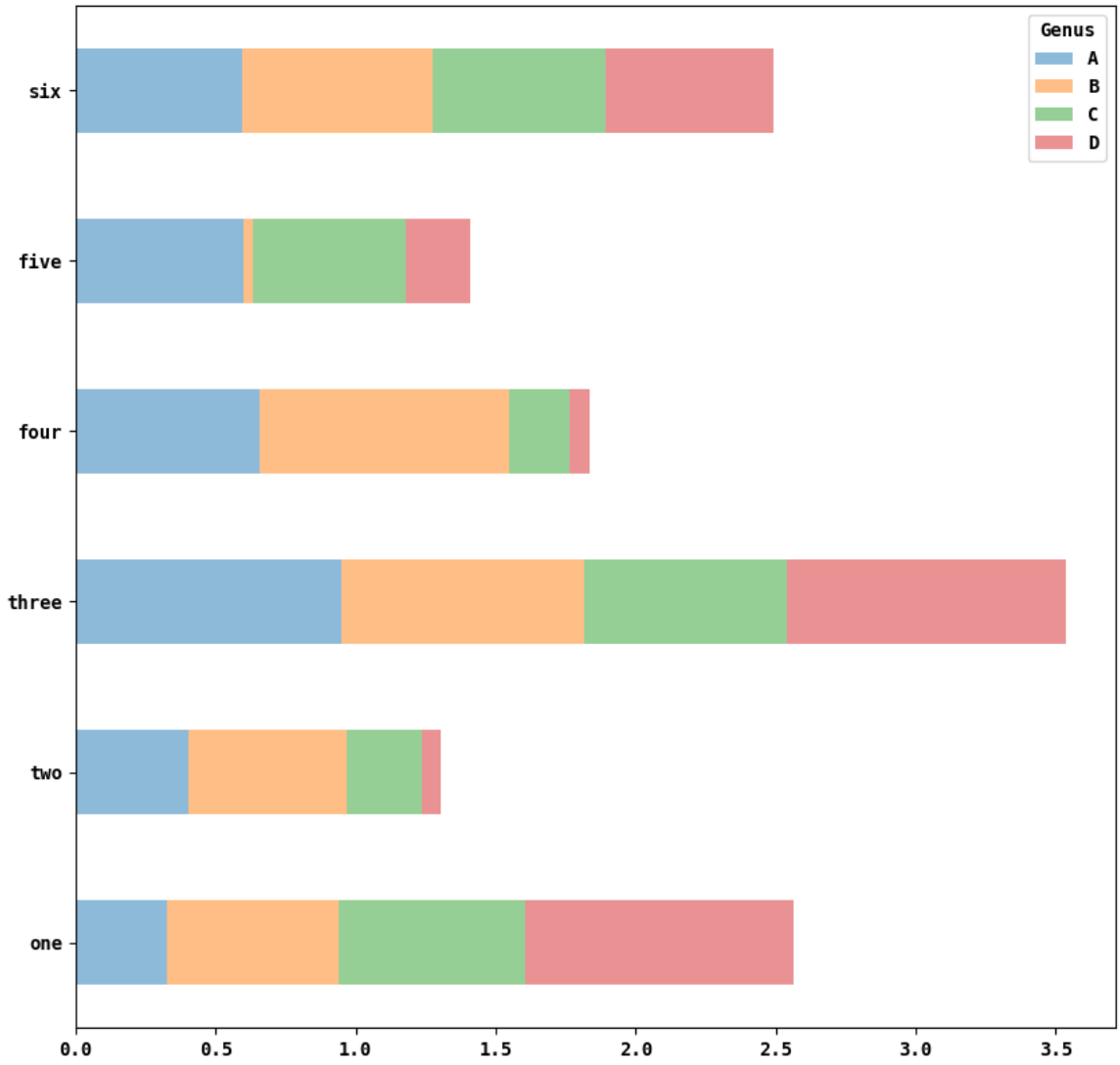
```
In [146]: df = pd.DataFrame(np.random.rand(6,4),
     ...:                    index = ['one','two','three','four','five',
     ...: 'six'],
     ...:                    columns=pd.Index(['A','B','C','D'],name='Ge
     ...: nus'))
```

```
In [147]: df
Out[147]:
Genus         A         B         C         D
one    0.327608  0.612006  0.667201  0.955855
two    0.402690  0.565665  0.269497  0.066820
three  0.948981  0.869267  0.720326  0.998532
four   0.656014  0.890000  0.217043  0.072368
five   0.600652  0.031866  0.546096  0.228095
six    0.594725  0.681314  0.617322  0.599028
```



DataFrame의 컬럼인 'Genus'가 범례의 제목으로 사용되었음을 확인하자. 누적막대그래프는 stacked = True 옵션을 사용해서 생성할 수 있는데, 각 로우의 값들이 하나의 막대에 누적되어 출력된다.

```
In [149]: df.plot.barh(stacked=True, alpha = 0.5)
Out[149]: <matplotlib.axes._subplots.AxesSubplot at 0x1d9bd8534c8>
```



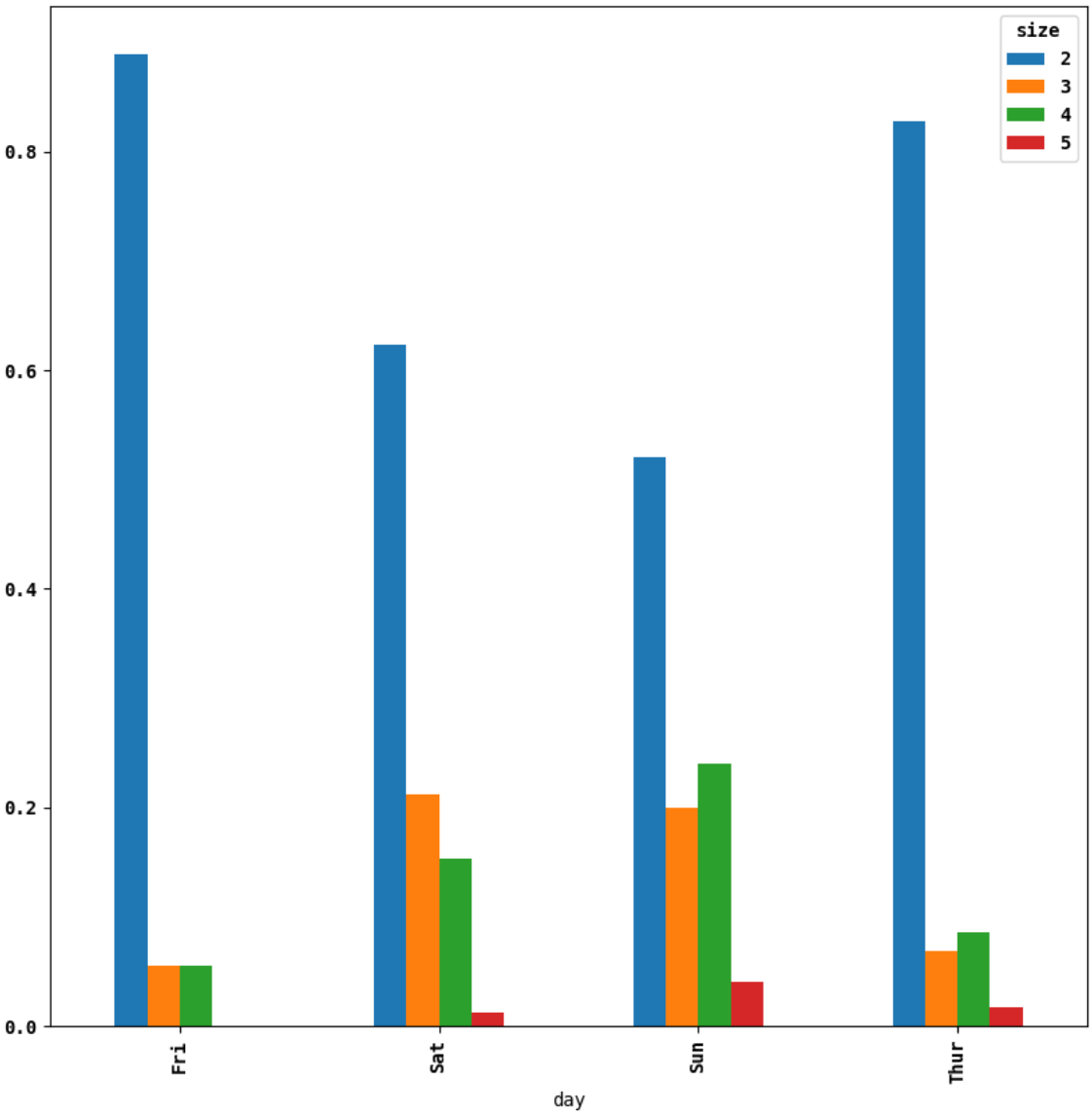
책의 앞에서 살려봤던 팁 데이터를 다시 살펴보자. 이 데이터에서 요일별 파티 숫자를 뽑고 파티 숫자 대비 팁 비율을 보여주는 막대그래프를 그려보자. read_csv 메서드를 사용해서 데이터를 불러오고 요일과 파티 숫자에 따라 교차 테이블을 생성했다.

```
In [150]: tips = pd.read_csv('tips.csv')

In [151]: party_counts = pd.crosstab(tips['day'],tips['size'])

In [152]: party_counts
Out[152]:
size  1   2   3   4  5  6
day
Fri   1  16   1   1  0  0
Sat   2  53  18  13  1  0
Sun   0  39  15  18  3  1
Thur  1  48   4   5  1  3

#1인과 6인 파티는 제외
In [154]: party_counts = party_counts.loc[:,2:5]
```

그리고 각 로우의 합이 1이 되도록 정규화하고 그래프를 그려보자.

```
In [155]: party_counts
Out[155]:
size   2   3   4  5
```

```
day
Fri   16   1   1  0
Sat   53  18  13  1
Sun   39  15  18  3
Thur  48   4   5  1

In [156]: party_pcts = party_counts.div(party_counts.sum(1), axis =0)

In [157]: party_pcts
Out[157]:
size         2         3         4         5
day
Fri   0.888889  0.055556  0.055556  0.000000
Sat   0.623529  0.211765  0.152941  0.011765
Sun   0.520000  0.200000  0.240000  0.040000
Thur  0.827586  0.068966  0.086207  0.017241

In [158]: party_pcts.plot.bar()
Out[158]: <matplotlib.axes._subplots.AxesSubplot at 0x1d9bdbc36c8>
```

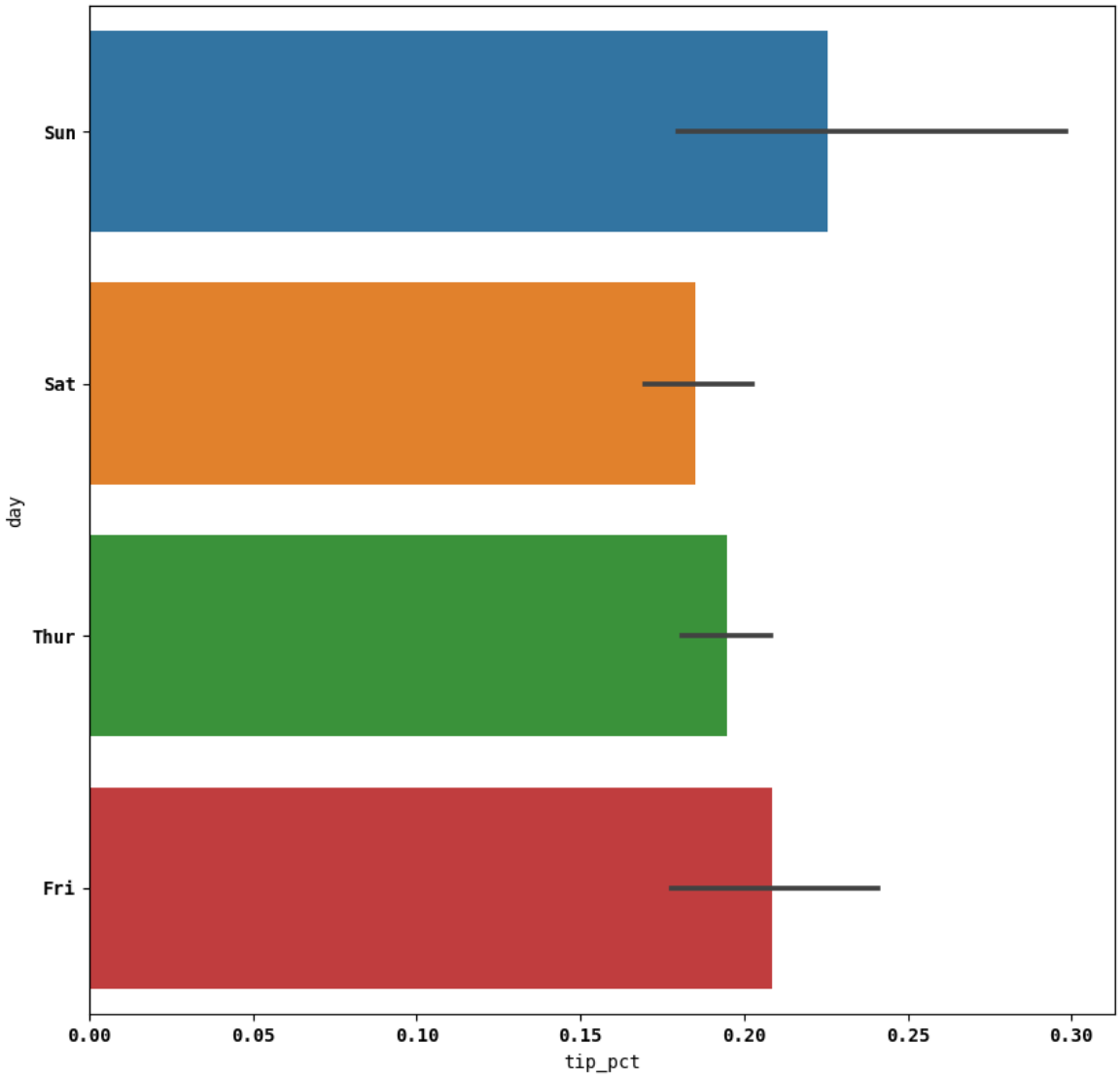


이 데이터에서 파티의 규모는 주말에 커지는 경향이 있음을 알 수 있다.

그래프를 그리지 전에 요약을 해야 하는 데이터는 seaborn 패키지를 이용하면 훨씬 간단하게 처리할 수 있다.

이번 에는 seaborn 패키지로 팁 데이터를 다시 그려보자.

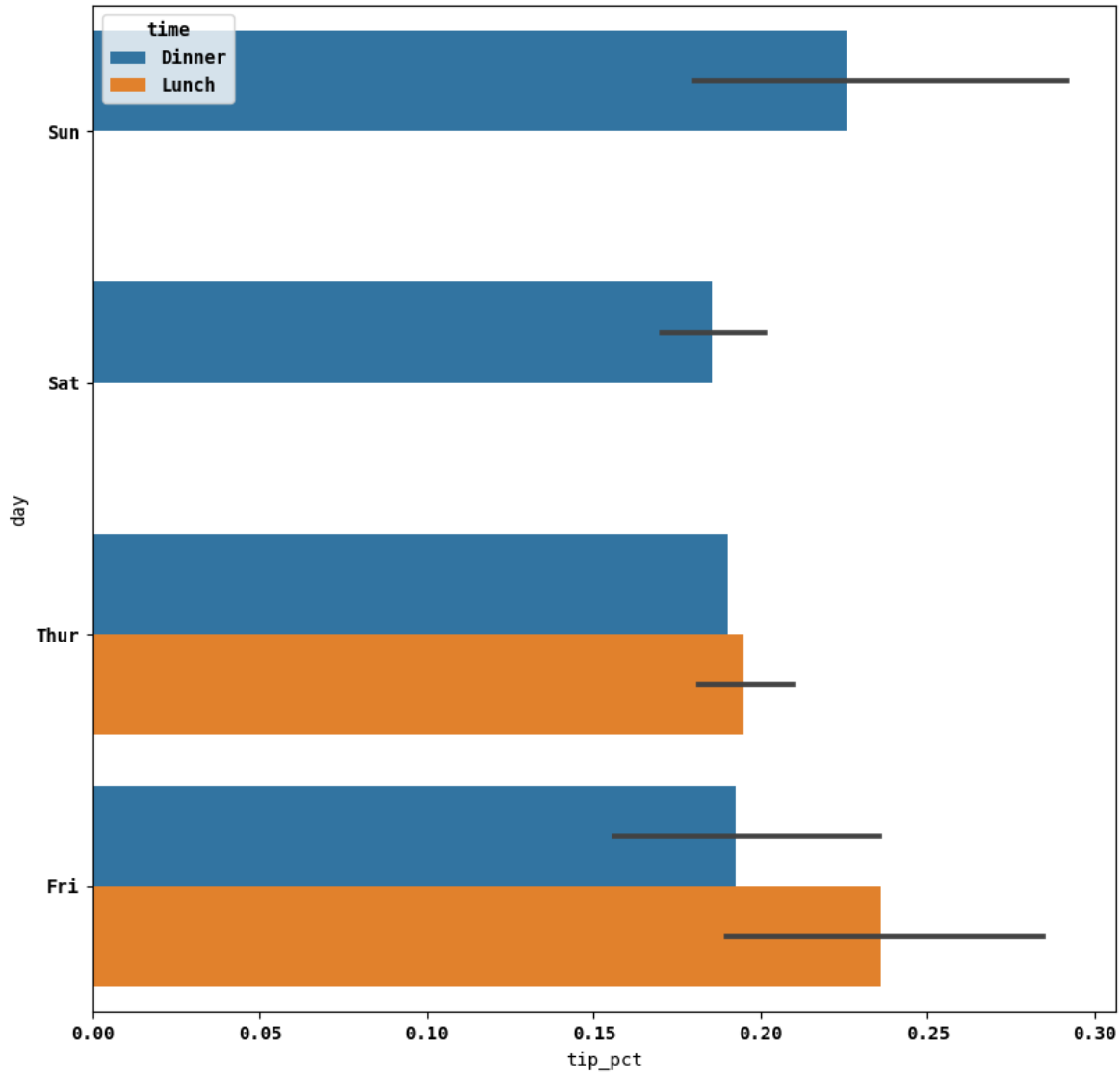
```
In [159]: import seaborn as sns

In [160]: tips['tip_pct'] = tips['tip']/(tips['total_bill']-tips['tip']
     ...: )

In [161]: tips.head()
Out[161]:
   total_bill   tip smoker  day    time  size   tip_pct
0       16.99  1.01     No  Sun  Dinner     2  0.063204
1       10.34  1.66     No  Sun  Dinner     3  0.191244
2       21.01  3.50     No  Sun  Dinner     3  0.199886
3       23.68  3.31     No  Sun  Dinner     2  0.162494
4       24.59  3.61     No  Sun  Dinner     4  0.172069

In [162]: sns.barplot(x='tip_pct',y='day',data = tips, orient = 'h')
Out[162]: <matplotlib.axes._subplots.AxesSubplot at 0x1d9bdbc36c8>
```



seaborn 플로팅 함수의 data 인자는 pandas의 DataFrame을 받는다. 다른 인자들은 컬럼 이름을 참조한다. day 컬럼의 각 값에 대한 데이터는 여럿 존재하므로, tip_pct의 평균값으로 막대그래프를 그린다. 막대그래프 위에 덧그려진 검은 선은 95%의 신뢰구간을 나타낸다(이 값은 옵션으로 설정 가능하다).

seaborn.barplot 메서드의 hue 옵션을 이용하면 추가 분류에 따라 나눠 그릴 수 있다.

```
In [165]: sns.barplot(x='tip_pct', y='day', hue = 'time', data = tips,
     ...: orient = 'h')
```

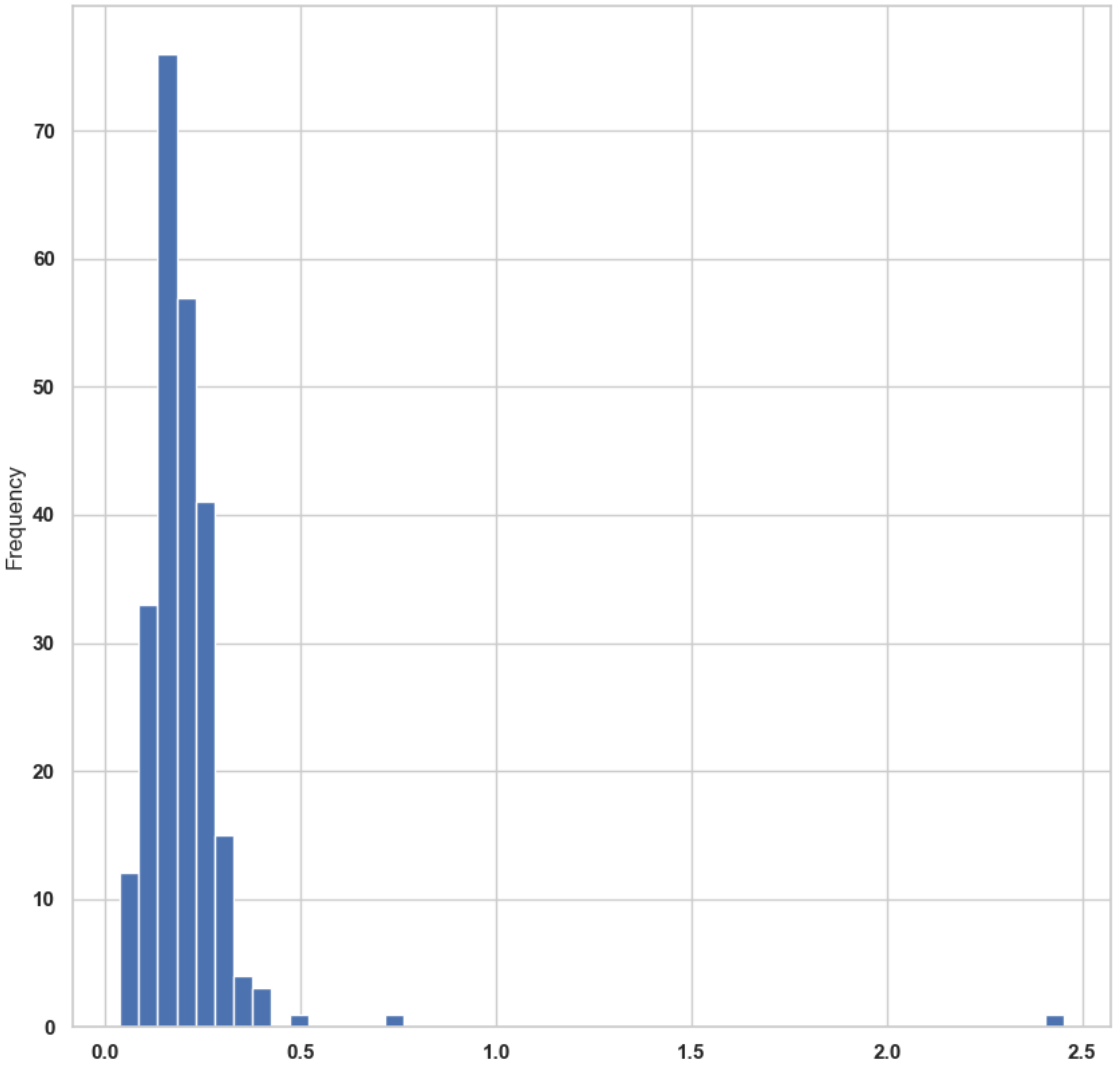


seaborn 라이브러리는 자동으로 기본 색상 팔레트, 그래프 배경, 그리드 선 색상 같은 꾸밈새를 변경한다. seaborn.set 메서드를 이용해서 이런 꾸밈새를 변경할 수 있다.

```
In [166]: sns.set(style = 'whitegrid')
```

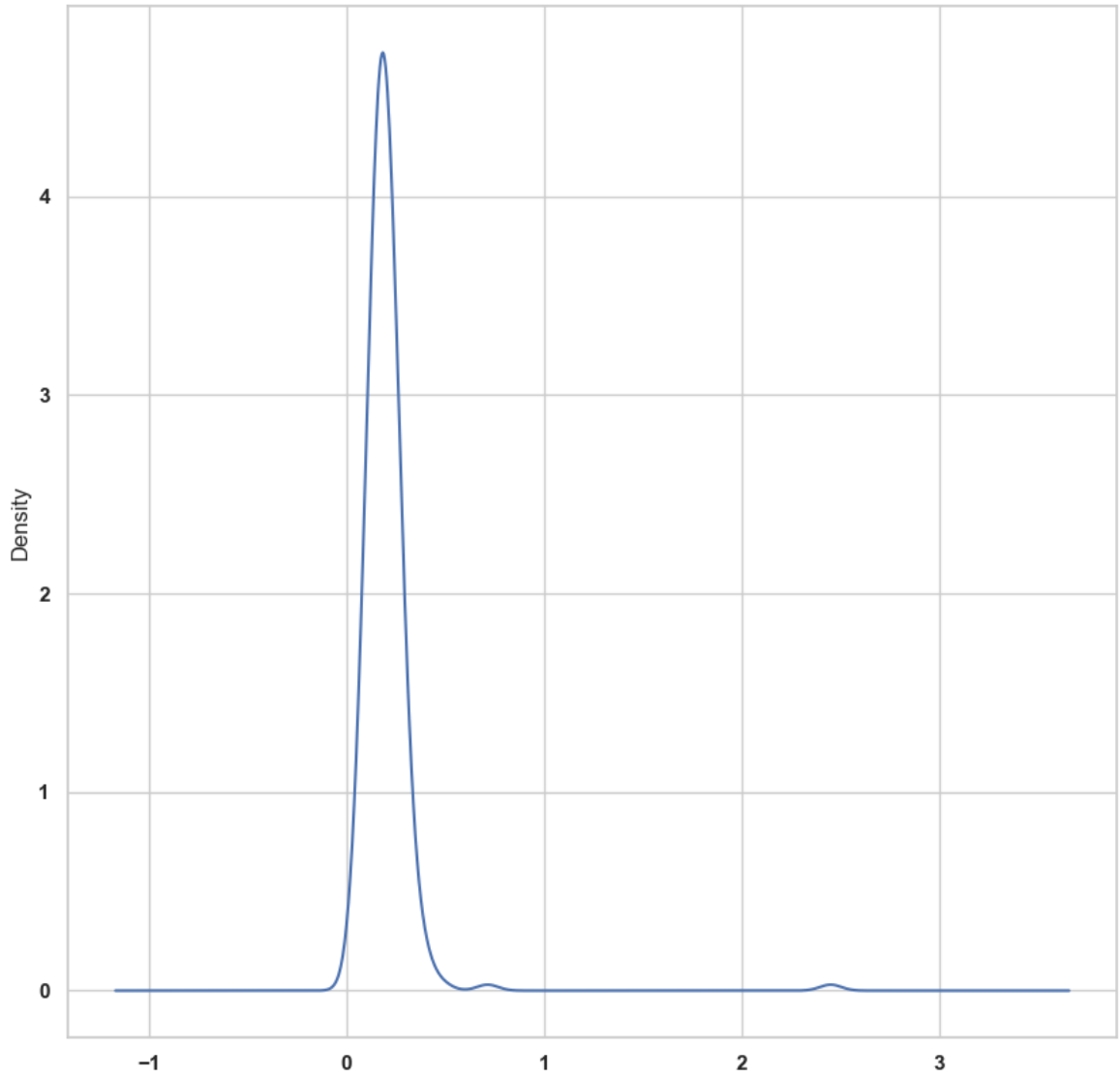
9.2.3 히스토그램과 밀도 그래프

히스토그램은 막대그래프의 한 종류로, 값 들의 빈도를 분리해서 보여준다. 데이터 포인트는 분리되어 고른 간격의 막대로 표현되며 데이터의 숫자가 막대의 높이로 표현된다. 앞에서 살펴본 팁 데이터를 사용해서 전체 결제금액 대비 팁 비율을 Series의 plot.hist 매서드를 사용해서 만들어보자.




```
In [171]: tips['tip_pct'].plot.hist(bins=50)
Out[171]: <matplotlib.axes._subplots.AxesSubplot at 0x1d9bd943188>
```

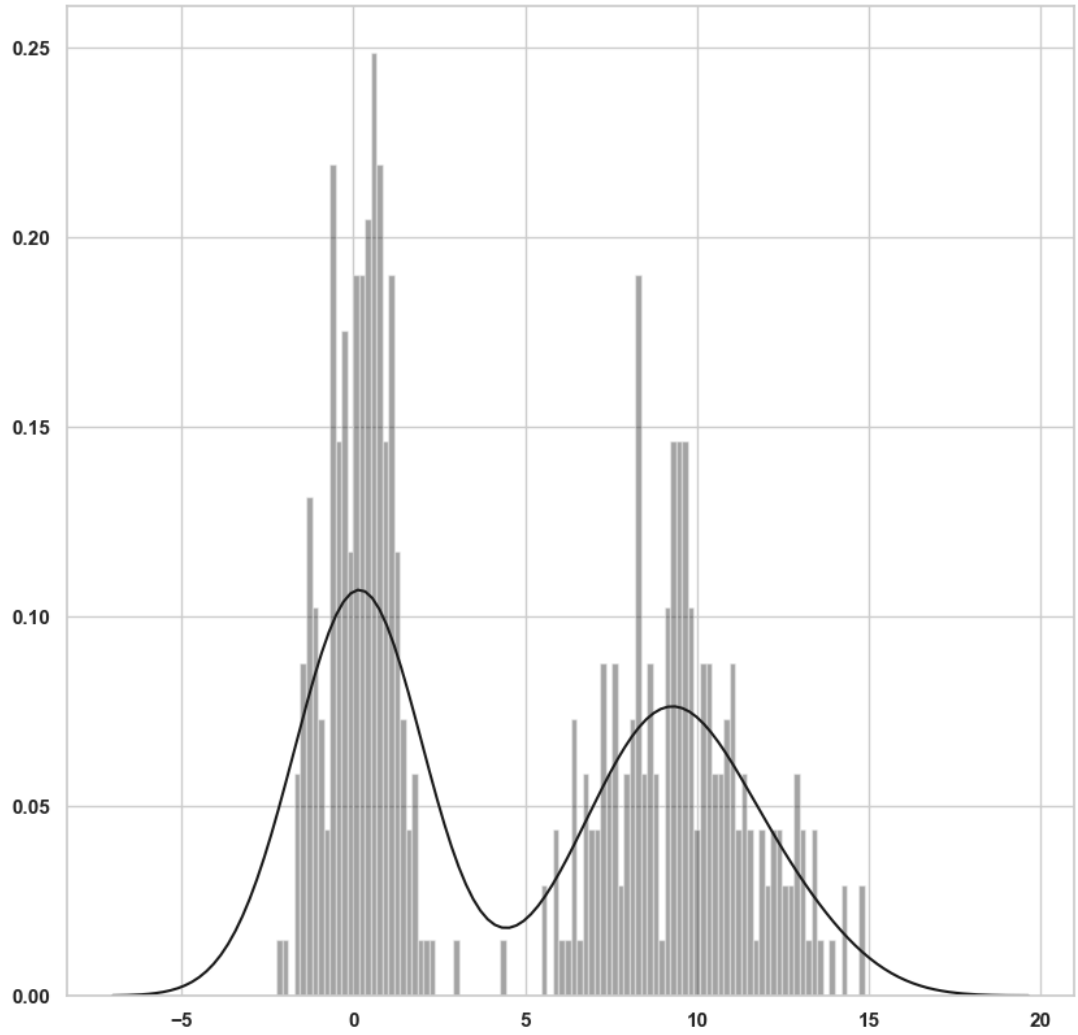
이와 관련 있는 다른 그래프로 밀도 그래프가 있는데 밀도 그래프는 관찰값을 사용해서 추정되는 연속된 확률 분포를 그린다. 일반적인 과정은 kernel 메서드를 잘 섞어서 이 분포를 근사하는 방법인데 이보다 단순한 정규 분포다. 그래서 밀도 그래프는 KDE(Kernel Density Estimate(커널 밀도 추정)) 그래프라고도 알려져 있다. plot.kde를 이용해서 밀도 그래프를 표준 KDE 형식으로 생성한다.




```
In [173]: tips['tip_pct'].plot.density()
Out[173]: <matplotlib.axes._subplots.AxesSubplot at 0x1d9c1235d88>
```

seaborn 라이브러리의 distplot 메서드를 이용해서 히스토그램과 밀도 그래프를 한 번에 손 쉽게 그릴 수 있다. 예를 들어 두 개의 다른 표준정규분포로 이뤄진 양봉분포를 생각해보자.

```
In [174]: comp1 = np.random.normal(0,1,size=200)

In [175]: comp2 = np.random.normal(10,2,size=200)

In [177]: values = pd.Series(np.concatenate([comp1,comp2]))

In [180]: sns.distplot(values, bins = 100, color = 'k')
Out[180]: <matplotlib.axes._subplots.AxesSubplot at 0x1d9c2035b88>
```

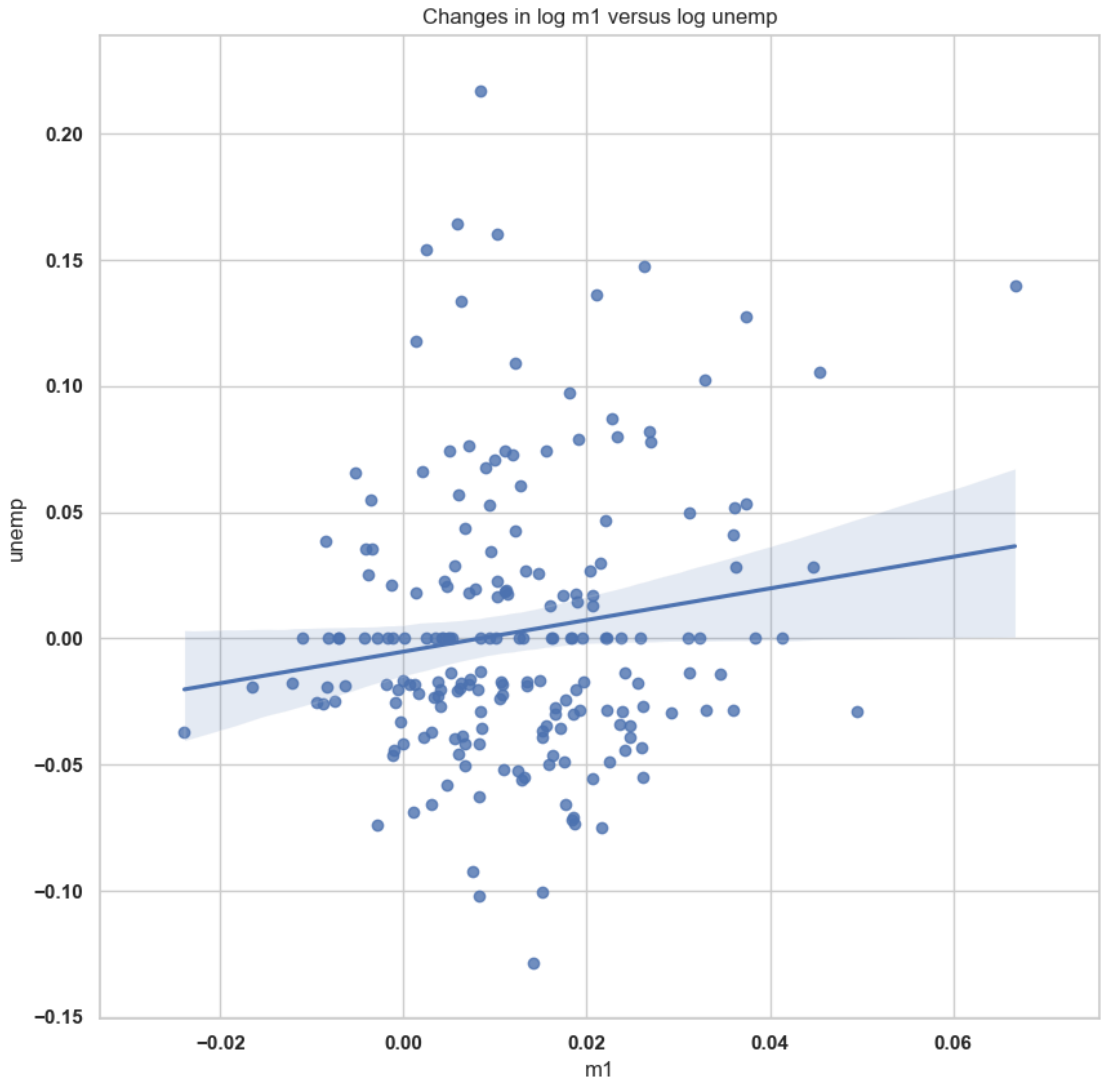


9.2.4 산포도

산포도(scatter plot, point plot)는 두 개의 1차원 데이터 묶음 간의 관계를 나타내고자 할 때 유용한 그래프이다. statsmodels 프로젝트에서 macrodata 데이터 묶음을 불러온 다음 몇 가지 변수를 선택하고 로그차를 구해보자.

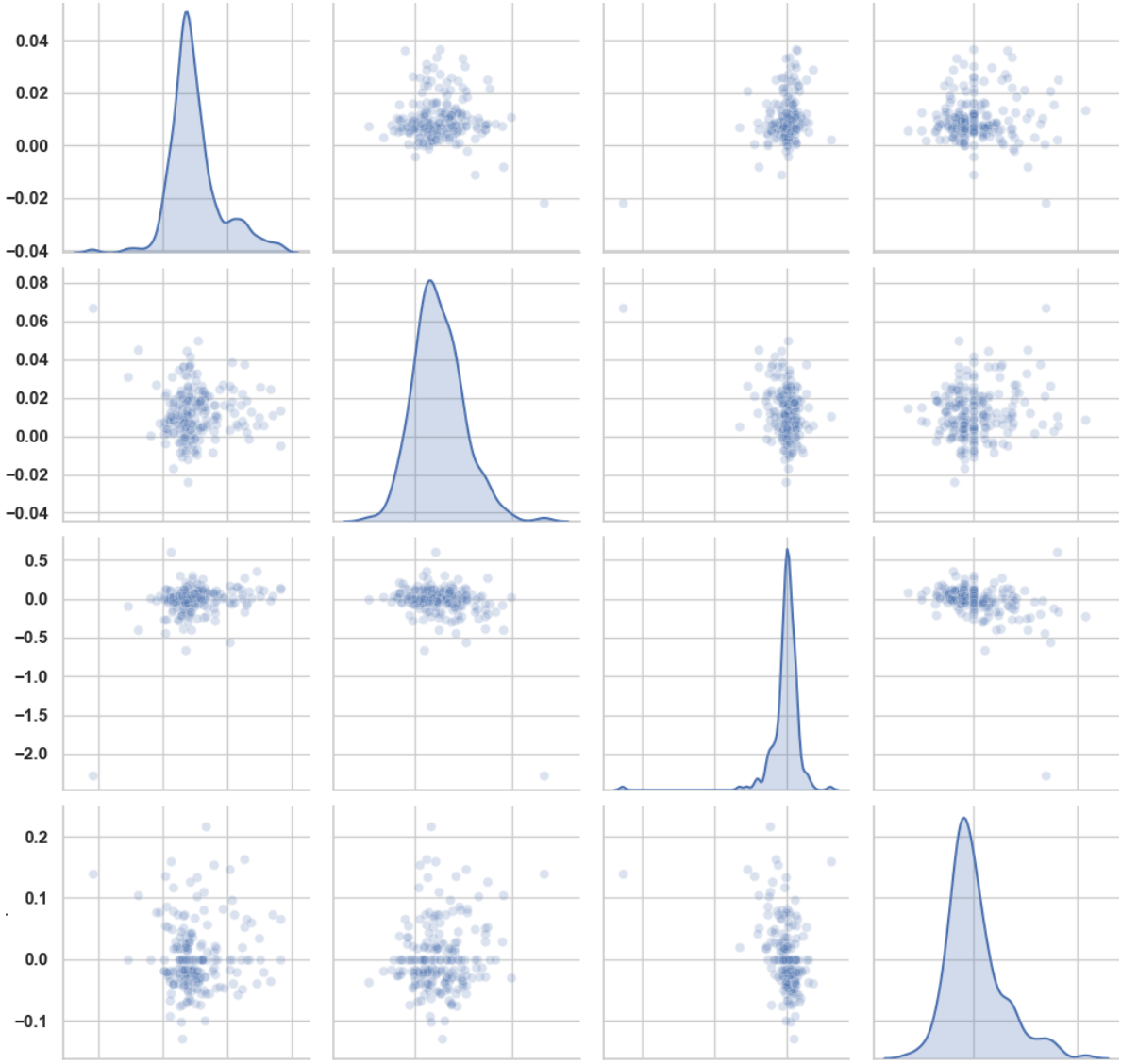
```
In [183]: macro = pd.read_csv('macrodata.csv')

In [184]: data = macro[['cpi', 'm1', 'tbilrate', 'unemp']]

In [186]: trans_data = np.log(data).diff().dropna()

In [188]: trans_data[-5:]
Out[188]:
          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
```

seaborn 라이브러리의 regplot 메서드를 이용해서 산포도와 선형회귀곡선을 함께 그릴 수 있다.




```
In [190]: plt.title('Changes in log %s versus log %s' % ('m1', 'unemp')
     ...: )
```

탐색 데이터 분석에서는 변수 그룹 간의 모든 산포도를 살펴보는 일이 매우 유용한데, 이를 **짝지은** 그래프 또는 **산포도 행렬**이라고 부른다. 이런 그래프를 직접 그리는 과정은 다소 복잡하기 때문에 seaborn에서는 pairplot 함수를 제공하여 대각선을 따라 각 변수에 대한 히스토그램이나 밀도 그래프도 생성할 수 있다.




```
In [191]: sns.pairplot(trans_data, diag_kind = 'kde', plot_kws={'alpha'
     ...: : 0.2})
Out[191]: <seaborn.axisgrid.PairGrid at 0x1d9c1261f08>
```

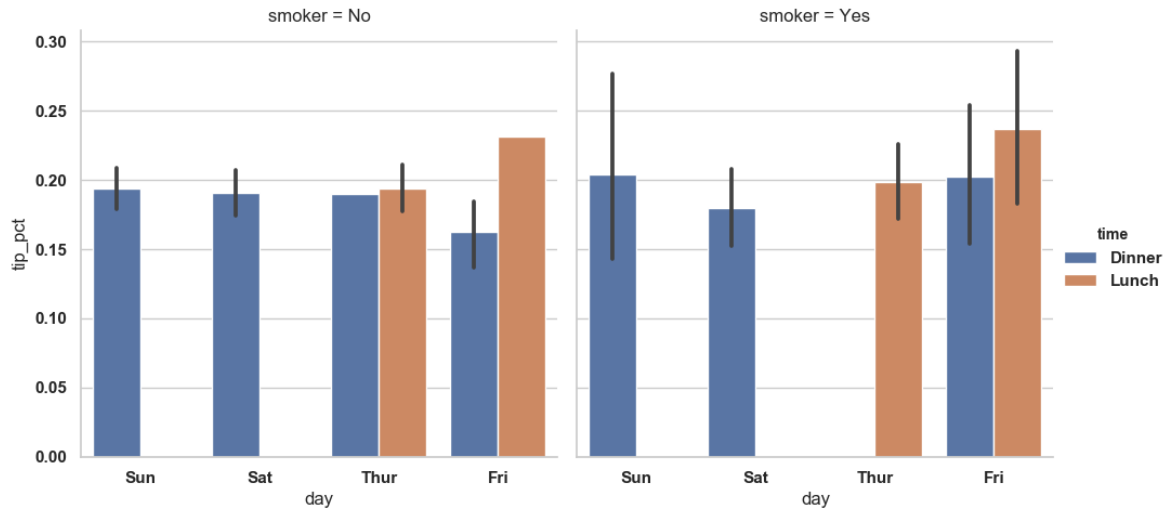
plot_kws 인자는 각각의 그래프에 전달할 개별 설정값을 지정한다.

9.2.5 패싯 그리드와 범주형 데이터

추가적인 그룹 차원을 가지는 데이터를 어떻게 시각화해야 할까? 다양한 범주형 값을 가지는 데이터를 시각화하는 한 가지 방법은 **패싯 그리드**를 이용하는 것이다. seaborn은 **factorplot**이라는 유용한

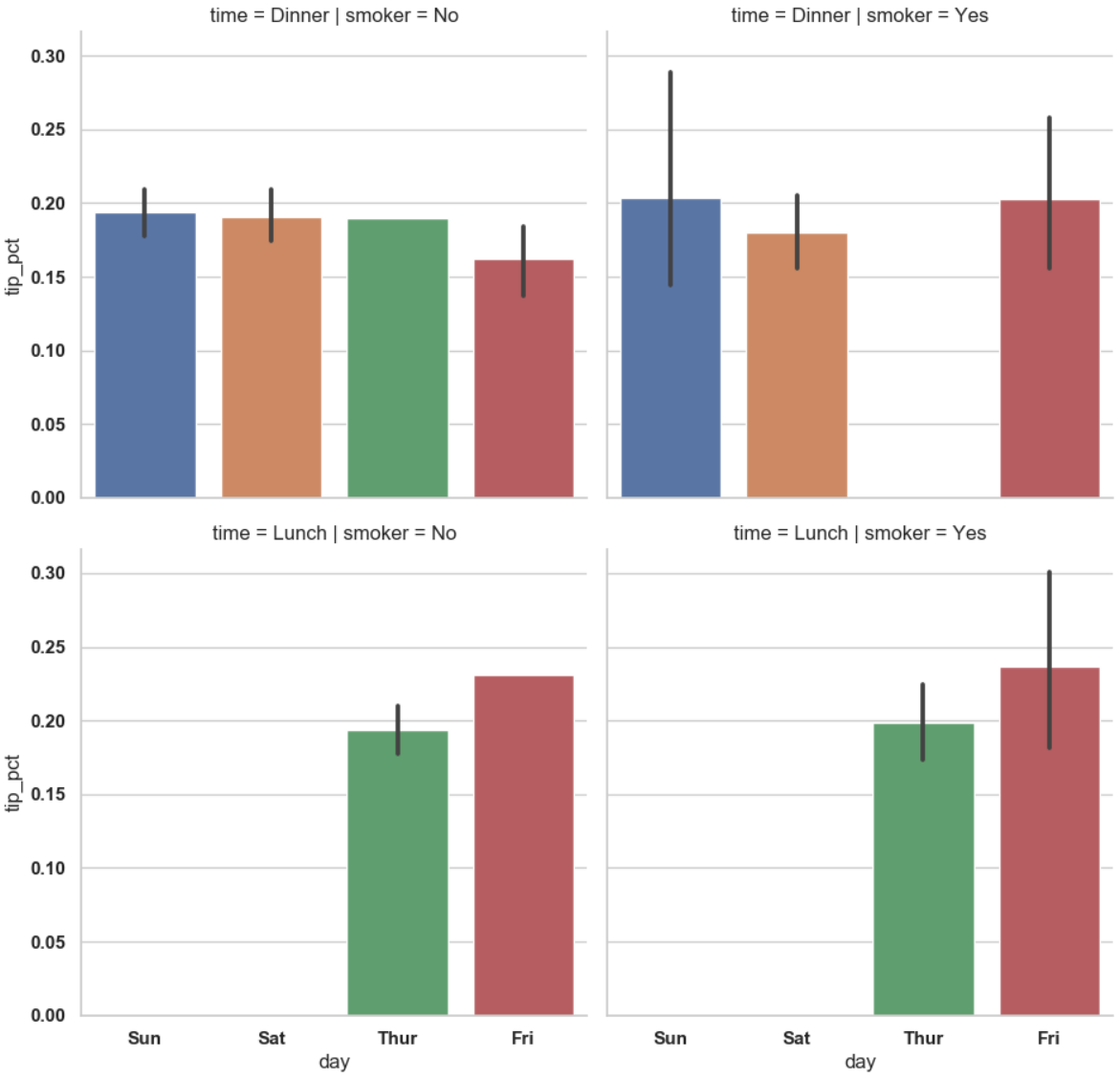
내장 함수를 제공하여 다양한 면을 나타내는 그래프를 쉽게 그릴 수 있게 도와준다.

```
In [192]: sns.factorplot(x='day', y='tip_pct', hue = 'time', col='smoke
     ...: r', kind = 'bar', data = tips[tips.tip_pct < 1])
```



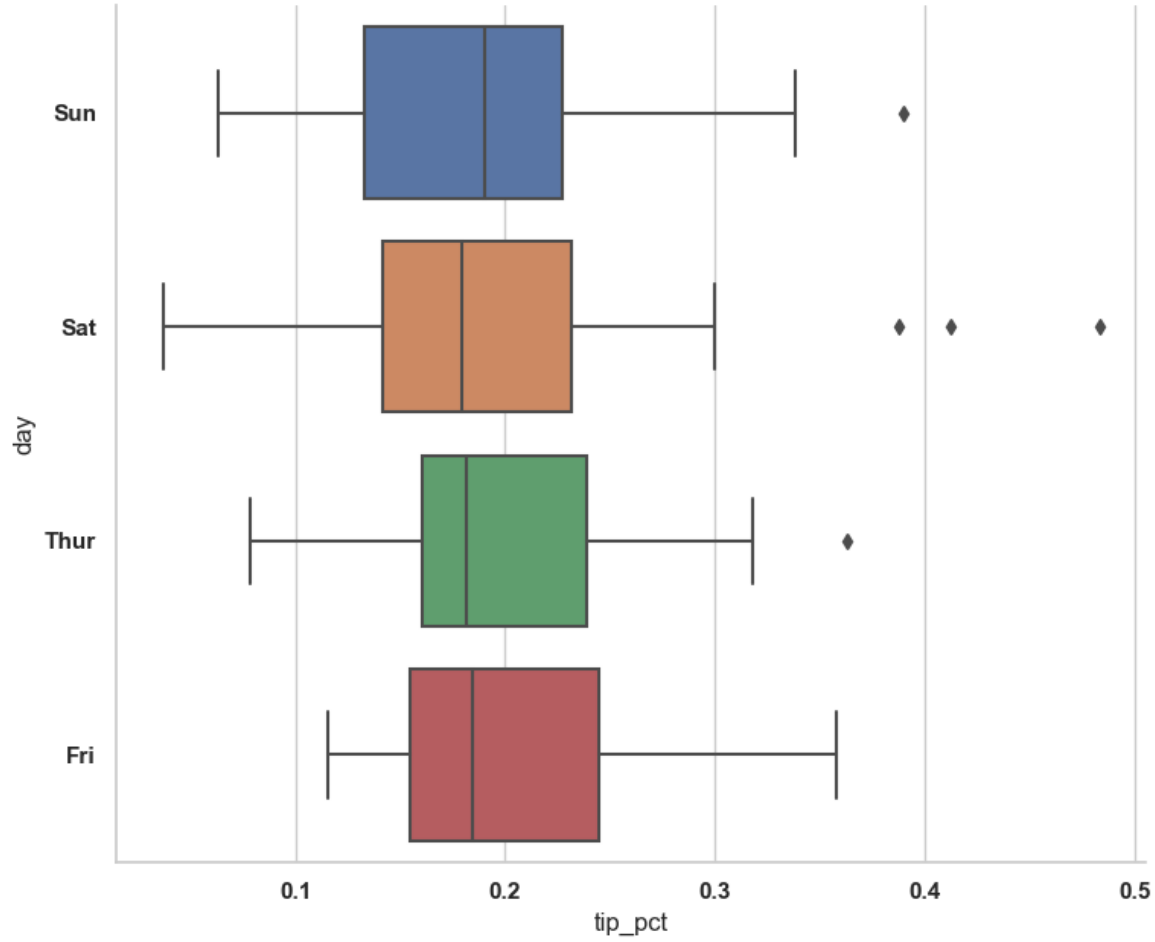
'time'으로 그룹을 만드는 대신 패싯 안에서 막대그래프의 색상을 달리해서 보여줄 수 있다. 또한 패싯 그리드에 time 값에 따른 그래프를 추가할 수도 있다.

```
In [194]: sns.factorplot(x='day', y='tip_pct', row = 'time', col='smoke
     ...: r', kind='bar', data = tips[tips.tip_pct < 1])
```



factorplot은 보여주고자 하는 목적에 어울리는 다른 종류의 그래프도 함께 지원한다. 예를 들어 중간 중간값과 사분위 그리고 특잇값을 보여주는 상자그림이 효과적인 시각화 방법일 수도 있다.

```
In [195]: sns.factorplot(x='tip_pct',y='day', kind = 'box', data = tips
     ...: [tips.tip_pct<0.5])
```



일반적인 용도의 seaborn.FacetGrid 클래스를 이용해서 나만의 패싯 그리드를 만들고 원하는 그래프를 그릴 수도 있다 .

9.3 다른 파이썬 시각화 도구

Bokeh(보케)나 Ploty(플로틀리) 같은 도구를 이용하면 웹 브라우저 상에서 파이썬으로 동적 대화형 그래프를 그릴 수 있다.

웹이나 출판을 위한 정적 그래프를 생성한다면 matplotlib과 pandas, seaborn을 기본으로 사용해라

9.4 마치며

이 장의 목적은 pandas, matplotlib 그리고 seaborn을 이용한 기본적인 데이터 시각화에 발을 담그도록 하는 것이다. 데이터 분석 결과를 시각적으로 공유해야 하는 것이 중요한 업무라면 효과적인 데이터 시각화에 대한 자료를 더 많이 찾아보라.