Pythonによるデータ分析入門_第９章プロットと可視化

9.1 Matplotlib APIの概要（P278）

- 簡単なプロットであれば以下でよいが、カスタマイズしたい場合は
- Matplotlib APIを習得したほうがよい

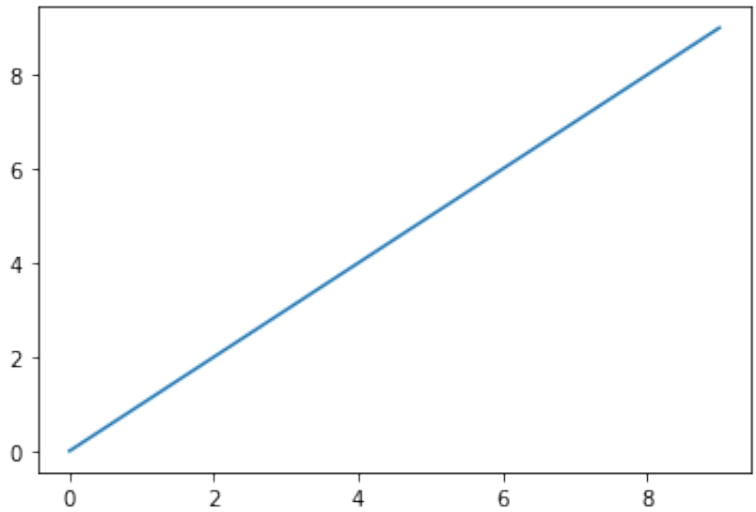
In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
data = np.arange(10)
data
```

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

In [3]:
```
plt.plot(data)
```

Out[3]:
```
[<matplotlib.lines.Line2D at 0x7ff9b04ee9d0>]
```

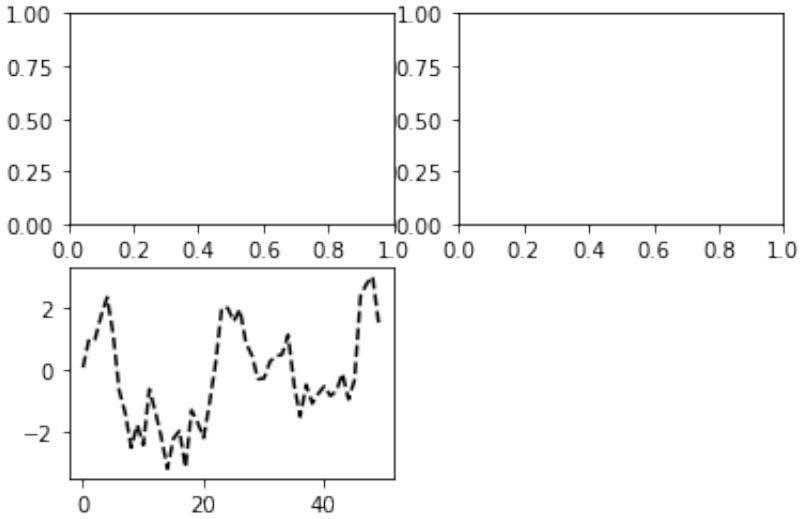


9.1.1 図とサブプロット（P279）

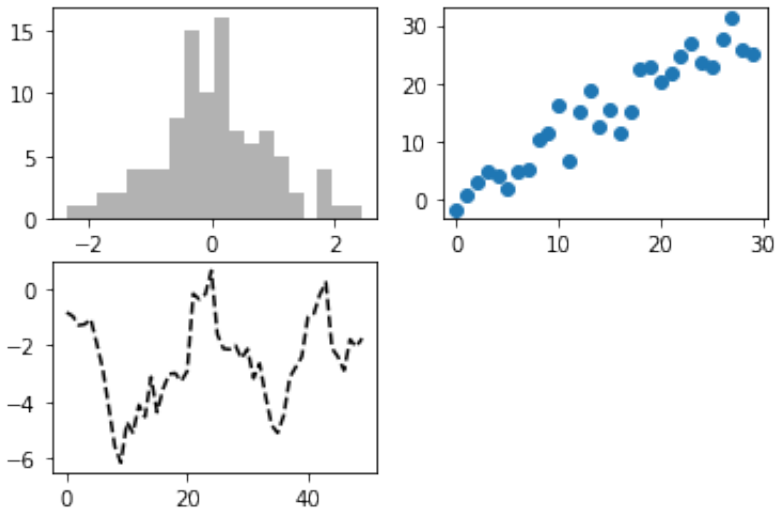
- サブプロットとは１つのグラフを描く領域のこと
- Jupyterの場合、セルが実行された後にプロットがリセットされるので１つのセルに
- すべてのプロットコマンドを入れた方が良い

In [13]:
```
fig = plt.figure()
ax1 = fig.add_subplot(2,2,1)
ax2 = fig.add_subplot(2,2,2)
ax3 = fig.add_subplot(2,2,3)

# plotコマンドを実行すると、Matplotlibはadd_subplotで作られた
# 最後３番目のサブプロットに描画を行う！
plt.plot(np.random.randn(50).cumsum(), 'k--')
```

Out[13]:
```
[<matplotlib.lines.Line2D at 0x7ff9b25fc790>]
```

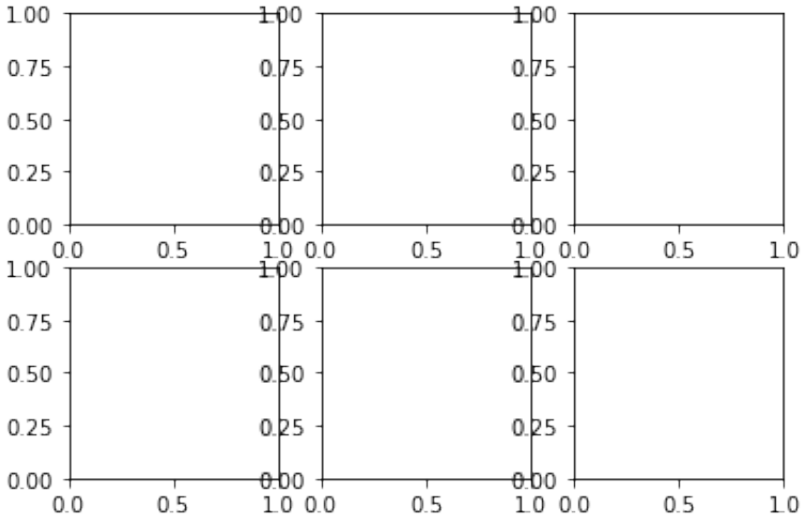


In [19]:
```
# ax1やax2のインスタンスメソッドを呼び出せば、それらサブプロットへ描画できる
fig = plt.figure()
ax1 = fig.add_subplot(2,2,1)
ax2 = fig.add_subplot(2,2,2)
ax3 = fig.add_subplot(2,2,3)

plt.plot(np.random.randn(50).cumsum(), 'k--')
ax1.hist(np.random.randn(100), bins=20, color='k', alpha=.3)
ax2.scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))
plt.show()
```



In [20]:
```
# plt.subplotで複数のサブプロットOBJを作成できる
fig, axes = plt.subplots(2,3)
axes
```

Out[20]:
```
array([[<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>],
       [<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>]], dtype=object)
```



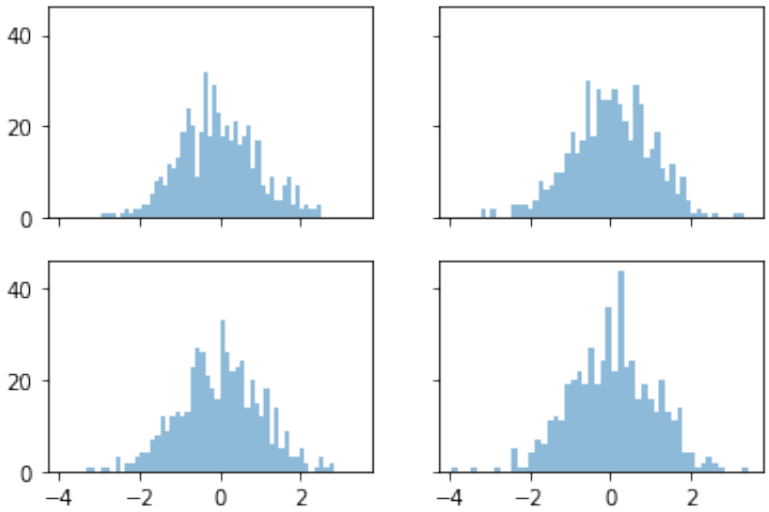
引数	説明
nrows	サブプロットの行数
ncols	サブプロットの列数
sharex	すべてのサブプロットで同じX軸の目盛りを使うように指定（xlimの変更がすべてのサブプロットに影響）
sharey	すべてのサブプロットで同じY軸の目盛りを使うように指定（ylimの変更がすべてのサブプロットに影響）
subplot_kw	各サブプロットを作成するために呼び出されるadd_subplotに渡されるキーワード引数のdic
**fig_kw	subplotsに与える、作図の際に用いる追加のキーワード引数（plt.subplots(2,2,figsize=(8,6)）など

9.1.1.1 サブプロットのまわりの空白を調整する（P283）

- デフォルトではサブプロットのまわりにかなりの空白ができ
- サブプロットの間にもスペースが開く
- スペースの調整はFigureオブジェクトのsubplots_adjustメソッドを使う

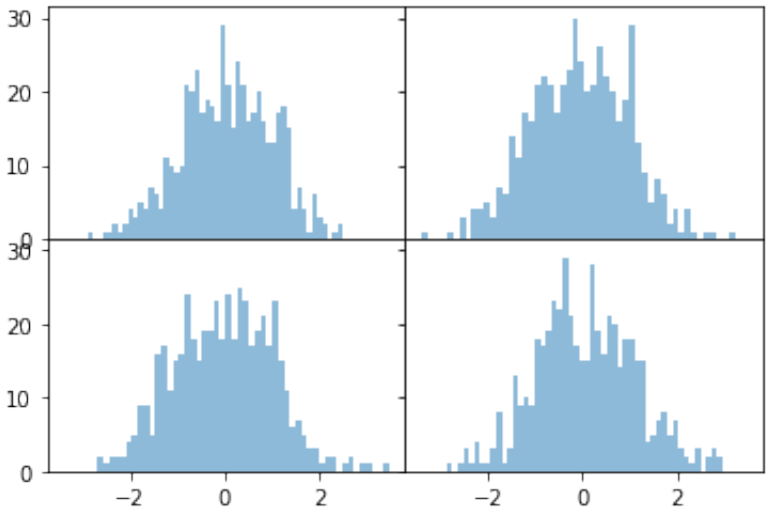
subplots_adjust(left=None, bottom=None, right=None, wspace=None, hspace=None) wspaceとhspaceは、図の幅と高さのうち、サブプロット間のスペースとして 使う領域の割合（%）

In [23]:
```
# デフォルト
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, alpha=.5)
```



In [27]:

```
# subplots_adjustで余白をなし
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, alpha=.5)

plt.subplots_adjust(wspace=0, hspace=0)
# 軸のラベルが重なってしまうため、自分で目盛りの位置とラベルを修正する必要あり（後述）
```

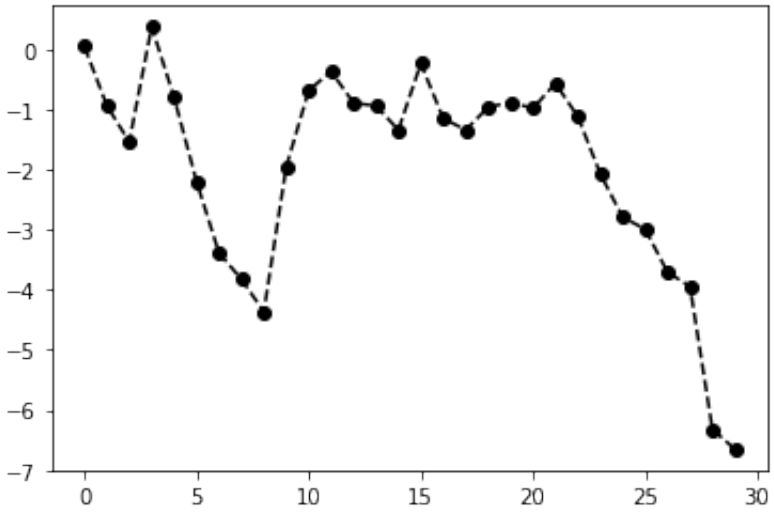


9.1.2 色、マーカー、線種（P284）

In [38]:

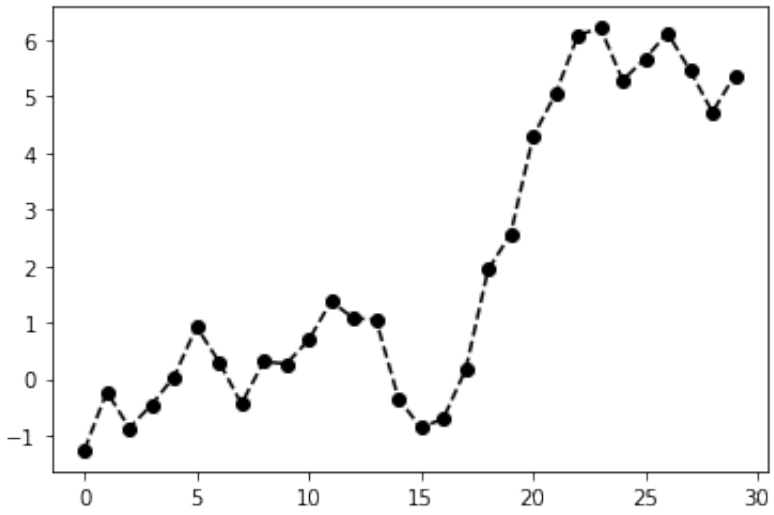
```
from numpy.random import randn
# この書き方、楽だね...（'ko--'）
plt.plot(randn(30).cumsum(), 'ko--')
```

Out[38]: [<matplotlib.lines.Line2D at 0x7ff9b2fea100>]



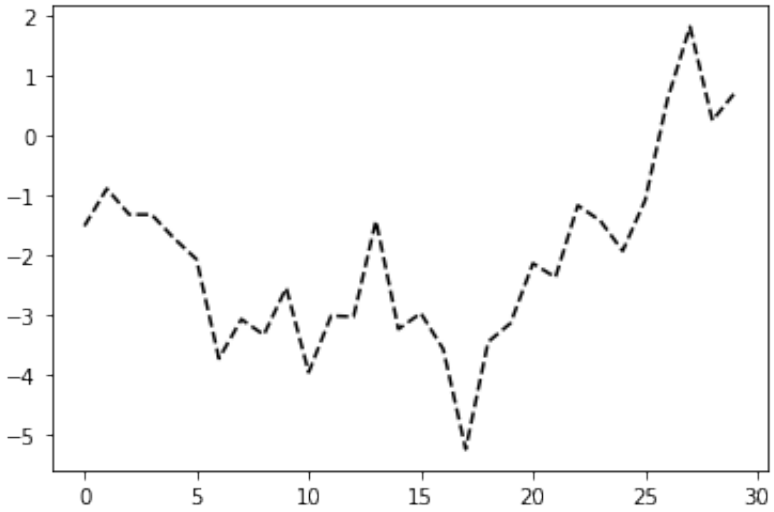
In [36]:
```
# 上記例は明示的に書くと
plt.plot(randn(30).cumsum(), color='k', linestyle='dashed', marker='o')
```

Out[36]: [<matplotlib.lines.Line2D at 0x7ff9b35509d0>]



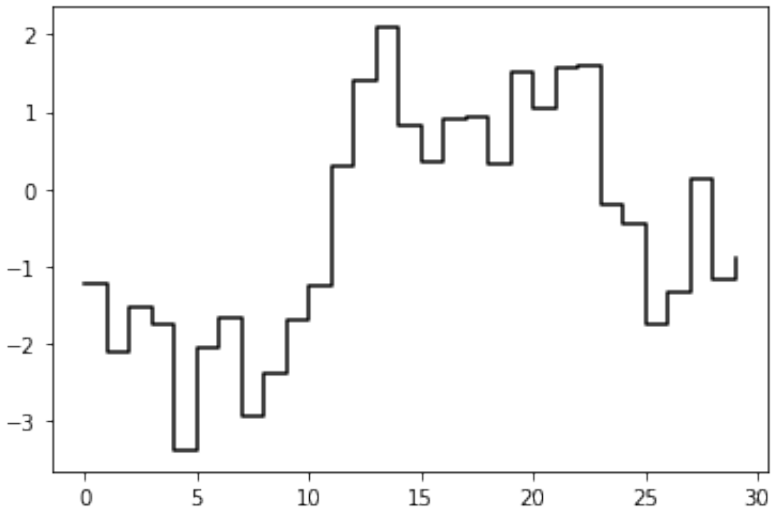
In [39]:
```
# 折線の点の直線は変更できる
data = np.random.randn(30).cumsum()
plt.plot(data, 'k--', label='Default')
```

Out[39]: [<matplotlib.lines.Line2D at 0x7ff9b2c65fd0>]



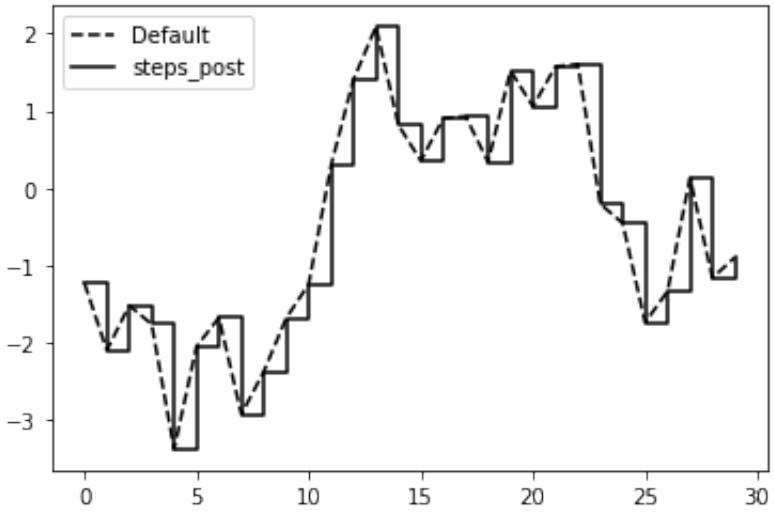
In [43]:
```
plt.plot(data, 'k-', drawstyle='steps-post', label='steps_post')
```

Out[43]: [<matplotlib.lines.Line2D at 0x7ff9b369d130>]



In [42]:
```
# 上記２つを同じサブプロットに表示
data = np.random.randn(30).cumsum()
plt.plot(data, 'k--', label='Default')
plt.plot(data, 'k-', drawstyle='steps-post', label='steps_post')
plt.legend(loc='best')
```

Out[42]: <matplotlib.legend.Legend at 0x7ff9b25dd7c0>



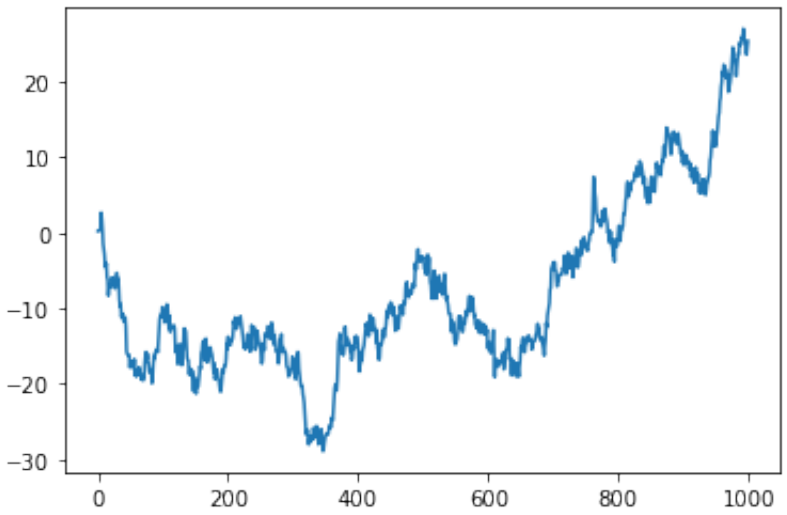
9.1.3 目盛、ラベル、判例（P286）

- プロットの装飾には２通り
- pyplotインターフェースを使う方法
  - 対話的な用途に向く
- オブジェクト的なMatplotlibネイティブのAPIを使う方法

9.1.3.1 タイトル、軸のラベル、目盛り、目盛りのラベルを設定する（P287）

In [44]:
```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.plot(np.random.randn(1000).cumsum())
```

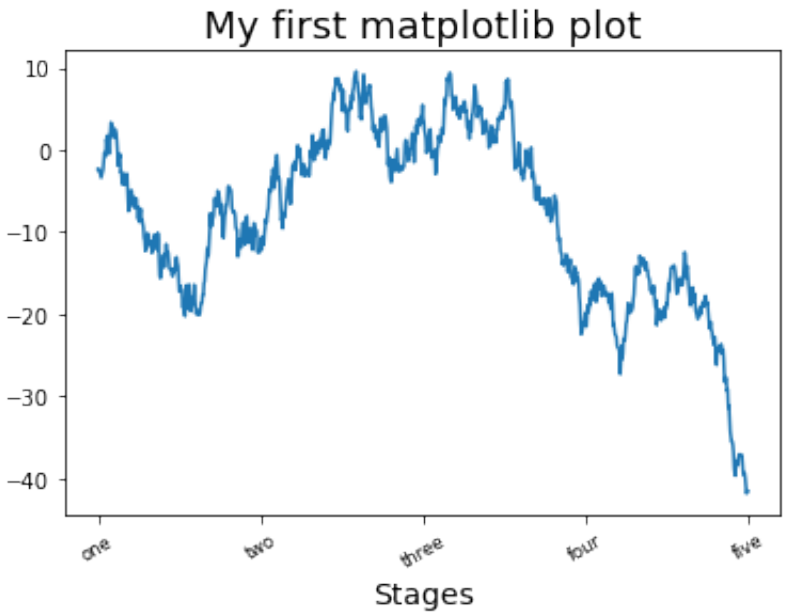
Out[44]: [<matplotlib.lines.Line2D at 0x7ff9b2ef0f10>]



- set_xticks：データ範囲のどこに目盛りを入れるか。デフォルトはメモリと同じ値にラベルも表示される
- set_xticklabels：どんな値でもラベルとして設定できる

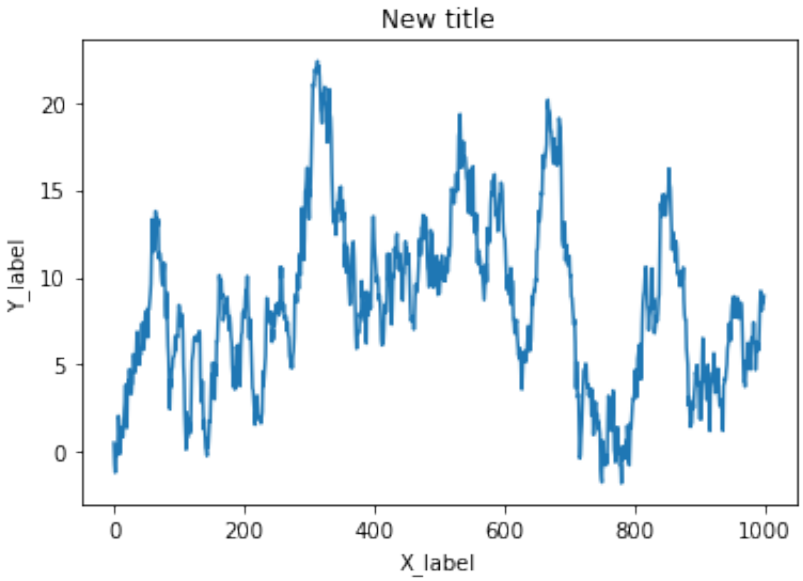
In [61]:
```
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.plot(np.random.randn(1000).cumsum())

# X軸の目盛を変更、set_xticksで目盛の場所を指定し、set_xticklabelsでその目盛をリネー
ticks = ax.set_xticks([0, 250, 500, 750, 1000])
labels = ax.set_xticklabels(['one', 'two', 'three', 'four', 'five'],\
                            rotation=30, fontsize='small')

ax.set_title('My first matplotlib plot', fontsize=18)
ax.set_xlabel('Stages', fontsize=14)

# Y軸も同様
```

Out[61]:
```
Text(0.5, 0, 'Stages')
```



In [64]:
```
# 軸のクラス（AxesSubplot）にはsetメソッドがあり
# プロットの属性を一括で設定できる

fig = plt.figure()
ax = fig.add_subplot(1,1,1)
ax.plot(np.random.randn(1000).cumsum())

props = {
    'title': 'New title',
    'xlabel': 'X_label',
    'ylabel': 'Y_label',
}

ax.set(**props)
```

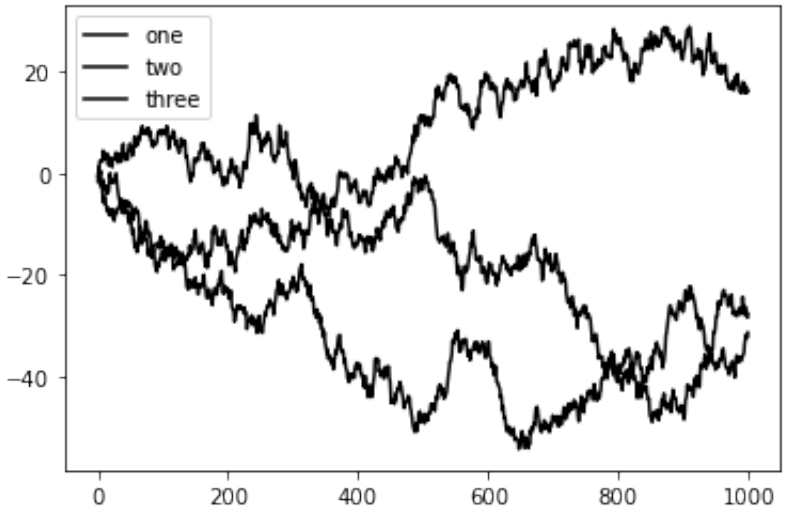
Out[64]:
```
[Text(0.5, 1.0, 'New title'), Text(0.5, 0, 'X_label'), Text(0, 0.5, 'Y_label')]
```



9.1.3.2 凡例の追加（P289）

In [75]:
```
fig = plt.figure(); ax = fig.add_subplot(1,1,1)

ax.plot(np.random.randn(1000).cumsum(), 'k', label='one')
ax.plot(np.random.randn(1000).cumsum(), 'k', label='two')
ax.plot(np.random.randn(1000).cumsum(), 'k', label='three')

# bestは最も邪魔にならない場所を選んでくれる
ax.legend(loc='best')

# 凡例から一部のデータを除きたい場合は、labelを指定しないか、label='_nolegend'と指定
```

Out[75]: <matplotlib.legend.Legend at 0x7ff9b3f5ea60>



9.1.4 サブプロットへの注釈や描画（P290）

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

In [ ]:
```
# 利用可能なフォントのリストを表示できる（量が多いため実行してない）
import matplotlib.font_manager as fm
fm.findSystemFonts()

# matplotlibでは、ttf,otf,afmの拡張子のみ認識できる
# 日本語に多い.ttcは扱えない（当日。今はできるはず？）
```

In [87]:
```
import pandas as pd
import pathlib
base_dir = pathlib.Path().cwd()
p = base_dir / 'pydata-book/examples/spx.csv'
```

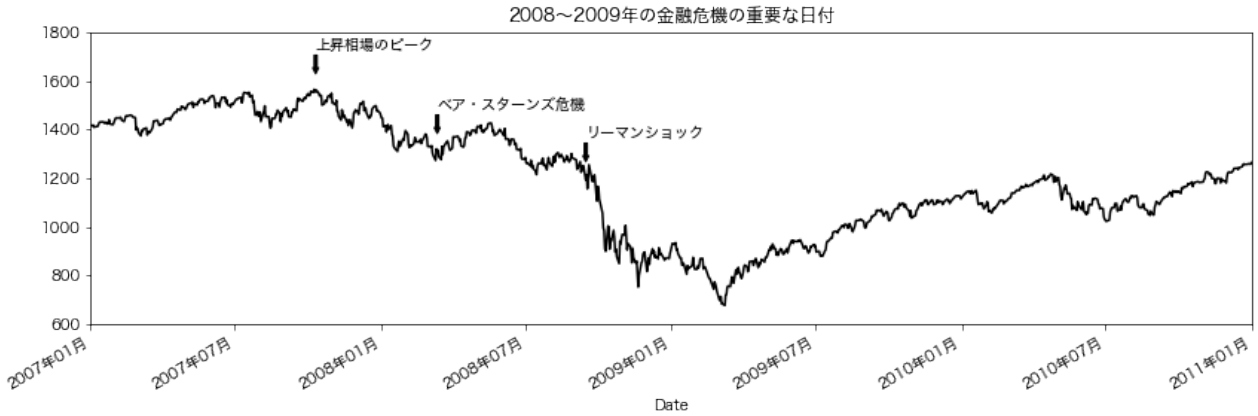
In [93]:
```
data = pd.read_csv(p, index_col=0, parse_dates=True)
data.head()
```

Out[93]:

	SPX
Date	
1990-02-01	328.79
1990-02-02	330.92
1990-02-05	331.85
1990-02-06	329.66
1990-02-07	333.75

In [123…

```
# import matplotlib
# matplotlib.use('Agg')

import matplotlib.dates as dates
from datetime import datetime

# 日本語フォントの設定
font_options = {'family': 'Hiragino Sans'}
plt.rc('font', **font_options)

fig = plt.figure()

# 追加
fig.set_size_inches(14,4)

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

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

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

crisis_data = {
    (datetime(2007, 10, 11), '上昇相場のピーク'),
    (datetime(2008, 3, 12), 'ベア・スターンズ危機'),
    (datetime(2008, 9, 15), 'リーマンショック')
}

# spx.asof(date)でcrisis_dataの日付と同日のSPXの値+αの位置に追記
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'\
               )

# X軸のラベルの日付表示を日本語に
datefmt = dates.DateFormatter('%Y年%m月')
ax.xaxis.set_major_formatter(datefmt)

# 2007~2010年をズーム
ax.set_xlim(['1/1/2007', '1/1/2011'])
ax.set_ylim([600, 1800])

ax.set_title('2008〜2009年の金融危機の重要な日付')

# 外枠がグレーになったので、facecolor='w'とした
# 別セルで独立して打っても中身が真っ白なまま出力される。上記一緒に実行しないとだめか。
# PDF出力はエラーになった
plt.savefig("/Users/mbp441/Desktop/test_fig.png", dpi=400, facecolor='w')
```



9.1.5 プロットのファイルへの保存（P293）

- 上述のとおり
- 以下「Matplotlib&Seaborn実装ハンドブック」（P81～）より画像ファイルとして保存するを試してみる
- レンダラーを呼び出してグラフをファイルに書き出す
- レンダラー（レンダリングエンジン）とはデータを特定の形式で描画するプログラムのこと

レンダラー	ファイル形式	説明
AGG	png	ラスター形式の高品質の画像を描画
PS	ps eps	ベクター形式
PDF	pdf	PDF
SVG	svg	ベクター形式

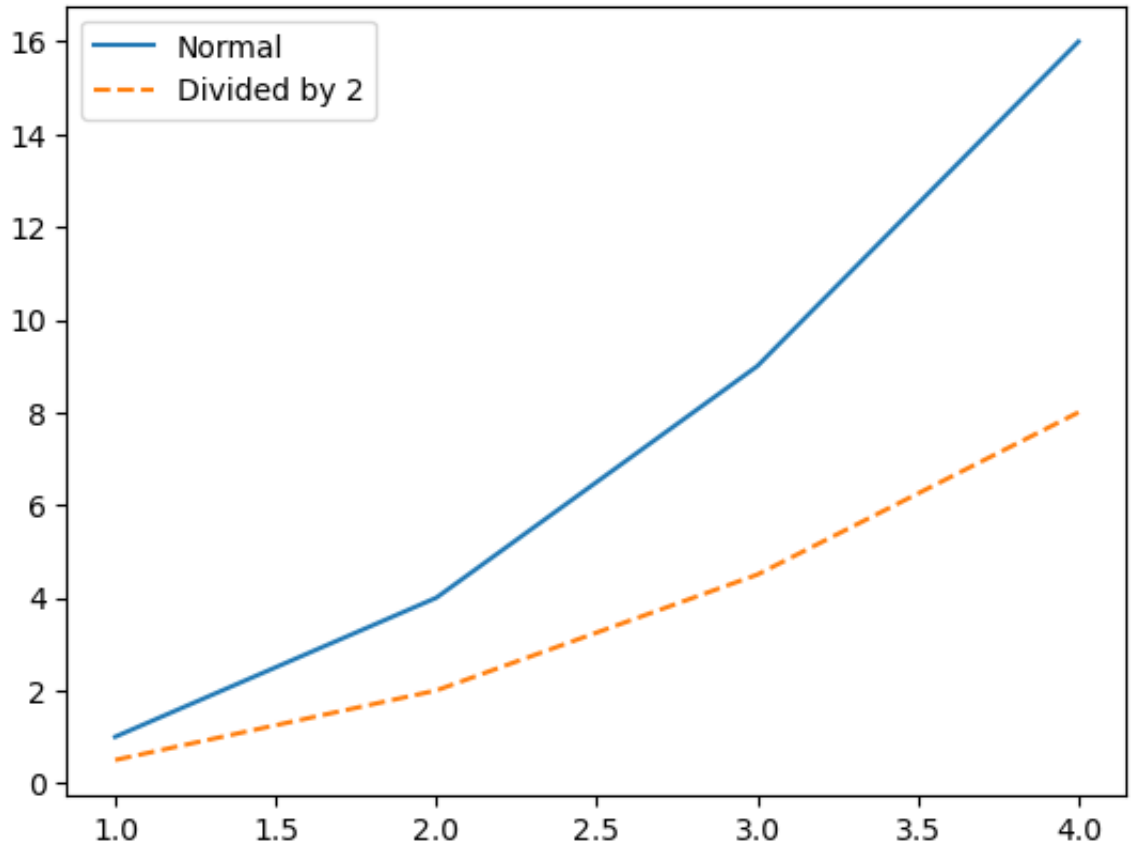
- 前のplt情報が残っている場合がある
- その場合は「plt.rcdefaults()」でリセットするとよい
- 以下であればpngもPDFも問題なく出力できた
- ただし一部日本語を使っているとPDF出力時にエラーが出る（日本語環境を構築していないため）

In [132…

```
plt.rcdefaults()
```

In [142…

```
import numpy as np
import matplotlib
# レンダラーの呼び出しはmatplotlibのuse()関数で行う（以下）
# ただし、これがなくても出力できる（？）
matplotlib.use('Agg')

import matplotlib.pyplot as plt

x = np.array([1,2,3,4])
y = np.array([1,4,9,16])

plt.plot(x, y, linestyle='solid', label='Normal')
plt.plot(x, y/2, linestyle='dashed', label='Divided by 2')
plt.legend()

plt.savefig("/Users/mbp441/Desktop/XXX.png", dpi=200)
plt.savefig("/Users/mbp441/Desktop/XXX.pdf", dpi=200)

# bbbox_inchesとpad_inchesは効果がなかった...理由不明

# plt.savefig("/Users/mbp441/Desktop/XXX_2.png", dpi=200, bbbox_inches='tig
# plt.savefig("/Users/mbp441/Desktop/XXX_2.png", dpi=200, pad_inches=0)
```



9.1.6 Matplolibの設定（P294）

- matplotlibのデフォルト値の設定はカスタマイズできる
- rcメソッドを利用
- リセットするにはplt.rcdefaults()を利用

plt.rc(group, **kwargs)

Alias	Property
'lw'	'linewidth'
'ls'	'linestyle'
'c'	'color'
'fc'	'facecolor'
'ec'	'edgecolor'
'mew'	'makeredgewidth'
'aa'	'antialiased'

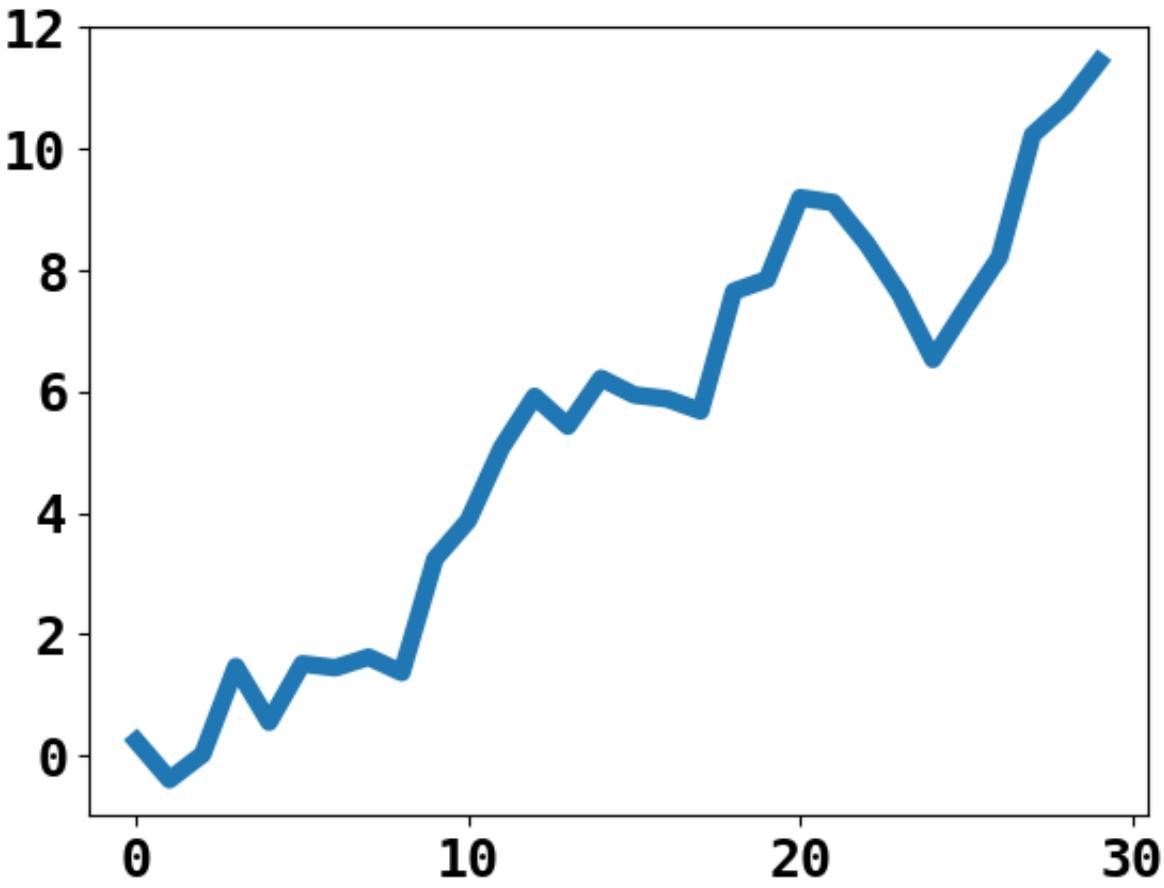
In [155…

```
plt.rc('lines', lw=6, c='r')

font = { 'family': 'monospace',
         'weight': 'bold',
         'size': 18,}

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

In [156…

```
plt.plot(randn(30).cumsum())
```

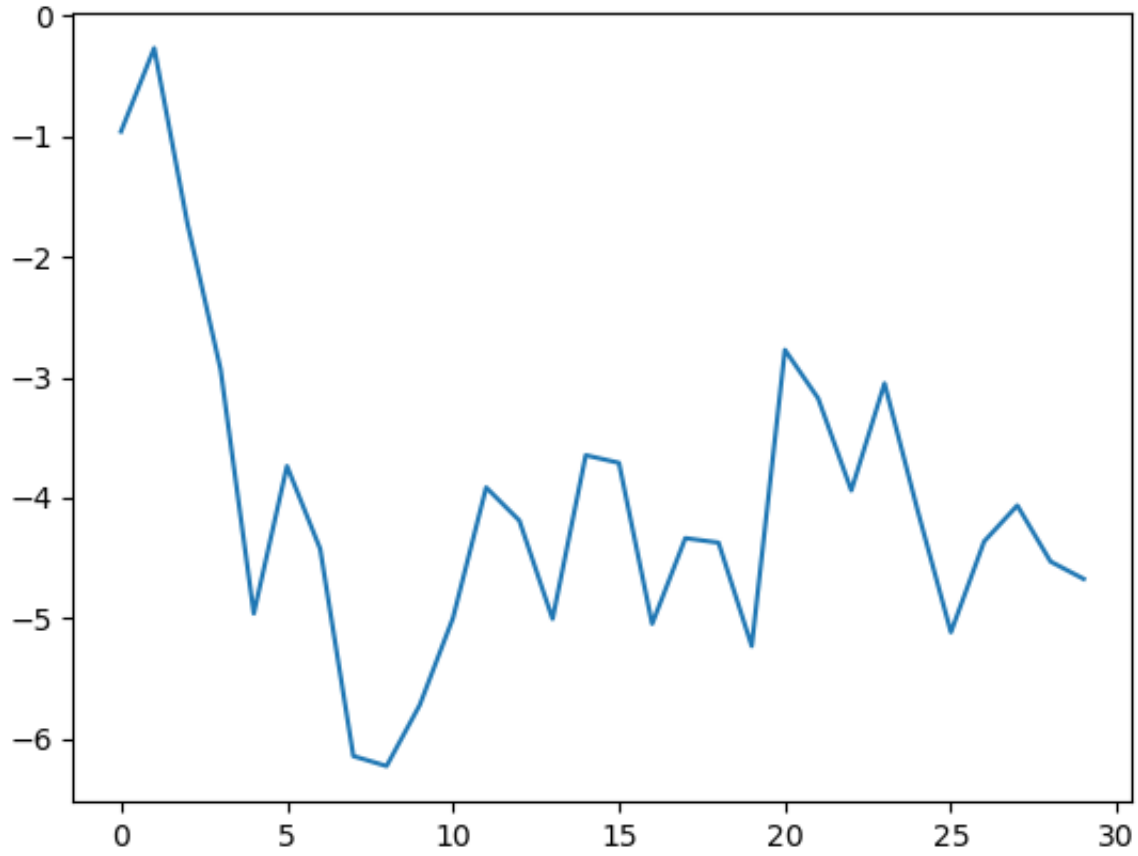
Out[156… [<matplotlib.lines.Line2D at 0x7ff99ebc3580>]



In [159…
```
plt.rcdefaults()
```

In [160…
```
plt.plot(randn(30).cumsum())
```

Out[160… [<matplotlib.lines.Line2D at 0x7ff99d427a60>]



9.2 pandasとseabornのプロット関数（P295）

※デフォルトの設定値を変更（SPOT）

In [174…

```
# デフォルトの設定値（サイズ）
plt.rcParams.get('figure.figsize')
```

Out[174… [3.0, 3.0]

In [201…

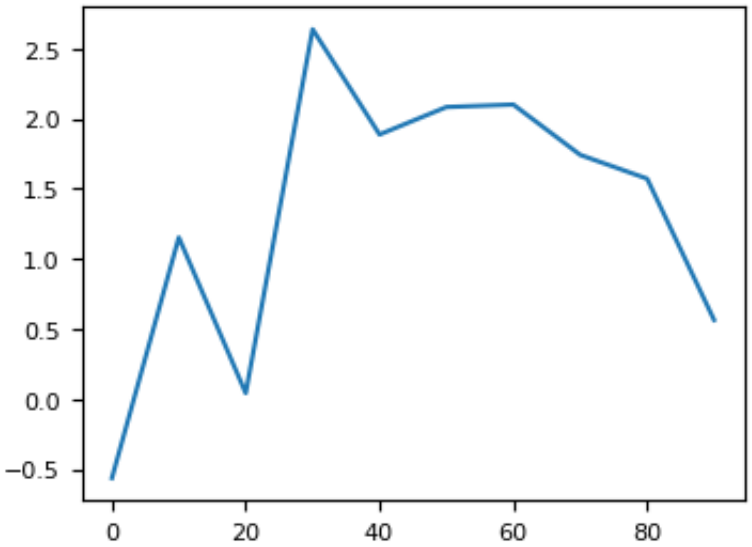
```
# デフォルトを変更する（サイズ）

params = {
    'figure.figsize': [4, 3],
#     'figure.autolayout': True, # ラベルのフォントサイズが大きくてもはみ出ないよう
#     'font.weight': 'normal',
#     'font.family': 'Open Sans', # ~/.matplotlib/fontList.cacheで使用可能フ
    'font.size': 8,
#     'legend.fontsize': 18,
#     'lines.linewidth': 2,
#     'axes.linewidth': 2,
#     'xtick.major.width': 1,
#     'ytick.major.width': 1,
#     'pdf.fonttype': 42 # pdfにhelveticaを埋め込み
}

plt.rcParams.update(params)
# デフォルト値に戻す
# plt.rcdefaults()
```

9.2.1 折線グラフ（P295）

In [203…

```
# seriesがx軸にプロットされる。
s = pd.Series(np.random.randn(10).cumsum(), \
              index=np.arange(0,100,10))
s.plot()
```

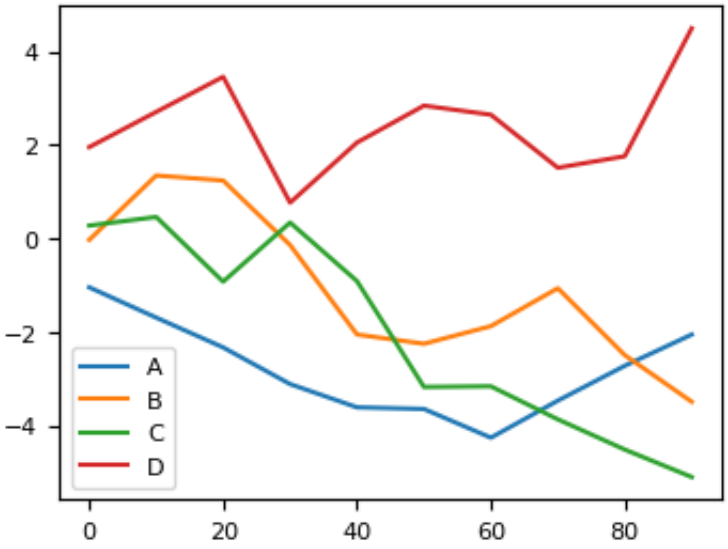
Out[203… <AxesSubplot:>



In [205…

```
df = pd.DataFrame(np.random.randn(10,4).cumsum(0),\
                  columns=list('ABCD'), \
                  index=np.arange(0,100,10))
df.plot()
```

Out[205… <AxesSubplot:>



Series.plotメソッドの引数

引数	説明
label	凡例のラベル
ax	プロットするMatplotlibのサブプロットオブジェクト
style	ko--'などの線種指定文字列
alpha	不透明度（0：透明～不透明：1）
kind	グラフの種類（'area', 'bar', 'density', 'hist', 'kde', 'line', 'pie'）
logy	Y軸にログスケールを使う
use_index	目盛りのラベルにオブジェクトのインデックスを使う
rot	目盛りのラベルの回転角
xticks	X軸の目盛りに使う値
yticks	Y軸のメモリに使う値
xlim	X軸の範囲（ex.[0 ,10]）
ylim	Y軸の範囲
grid	軸のグリッドを表示する（デフォルトで有効）

DataFrameのplotのみに指定できる引数

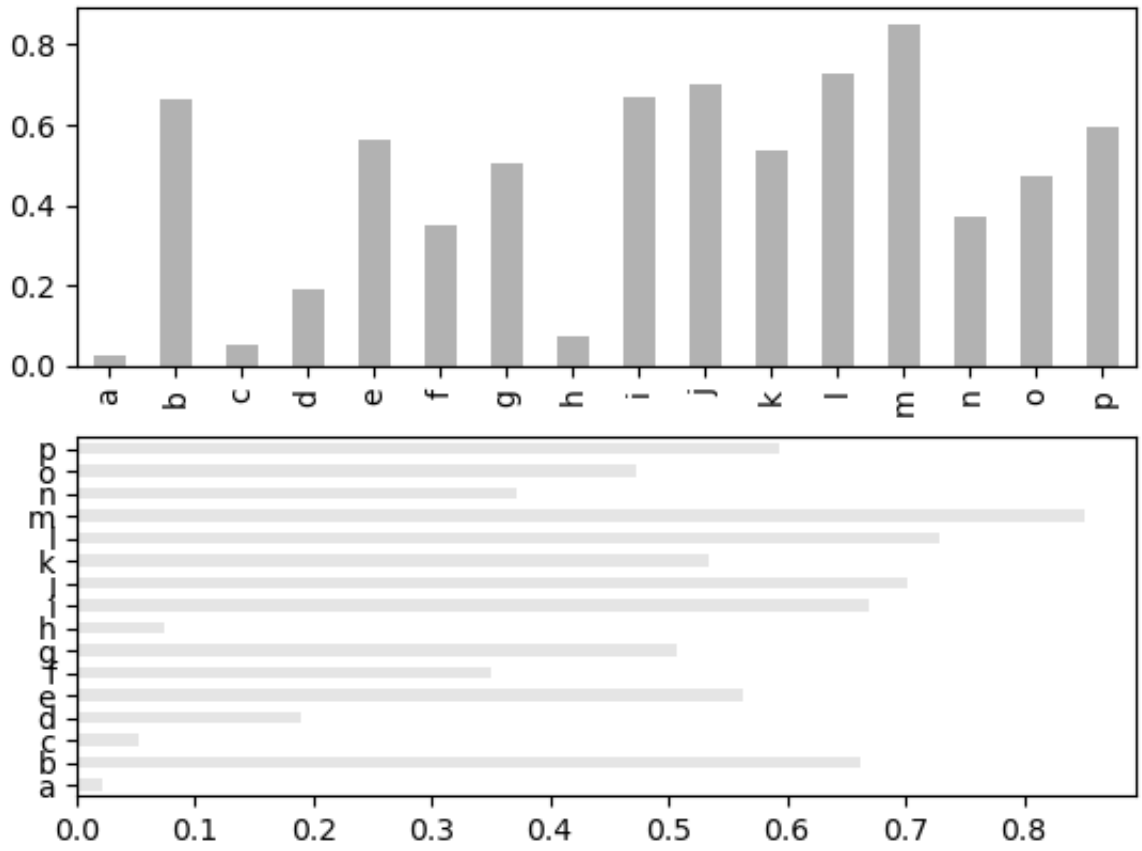
引数	説明
subplots	dfの各列を別々のサブプロットにプロット
sharex	subplots=Trueの場合、サブプロット間でX軸を共有し、X軸の目盛や範囲を関連づける
sharey	subplots=Trueの場合、サブプロット間でY軸を共有する
figsize	作成するFigureのサイズ。文字列で指定する
title	プロットのタイトル
legend	凡例
sort_columns	列をアルファベット順にプロット

9.2.2 棒グラフ（P298）

In [210…
```
plt.rcdefaults()
```

In [223…
```
fig, axes = plt.subplots(2,1)
data = pd.Series(np.random.rand(16), index=list('abcdefghijklmnop'))

data.plot.bar(ax=axes[0], color='k', alpha=0.3)

data.plot.barh(ax=axes[1], color='k', alpha=0.1)
```

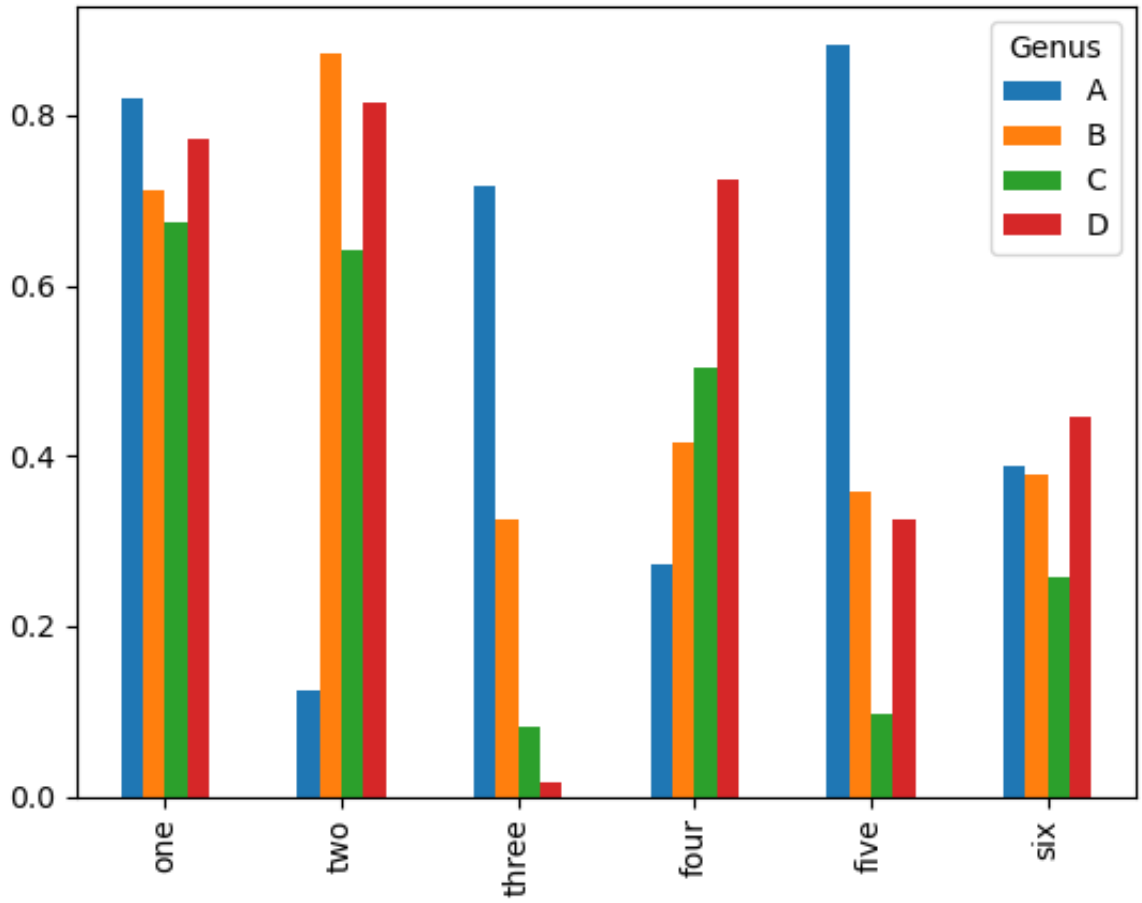
Out[223… <AxesSubplot:>



In [221…
```
# dataframeから棒グラフを描いた場合、各行の値は棒のグループとして
# まとめて並べられる
df = pd.DataFrame(np.random.rand(6,4),\
        index=['one','two','three','four','five','six'],\
        columns=pd.Index(list('ABCD'), name='Genus'))

display(df)

df.plot.bar()
```

Genus	A	B	C	D
one	0.820274	0.712910	0.673945	0.771734
two	0.125398	0.873668	0.642543	0.813839
three	0.717379	0.325814	0.082112	0.015361
four	0.272910	0.414640	0.503782	0.724888
five	0.883857	0.359078	0.097671	0.324853
six	0.389143	0.377691	0.257666	0.446471

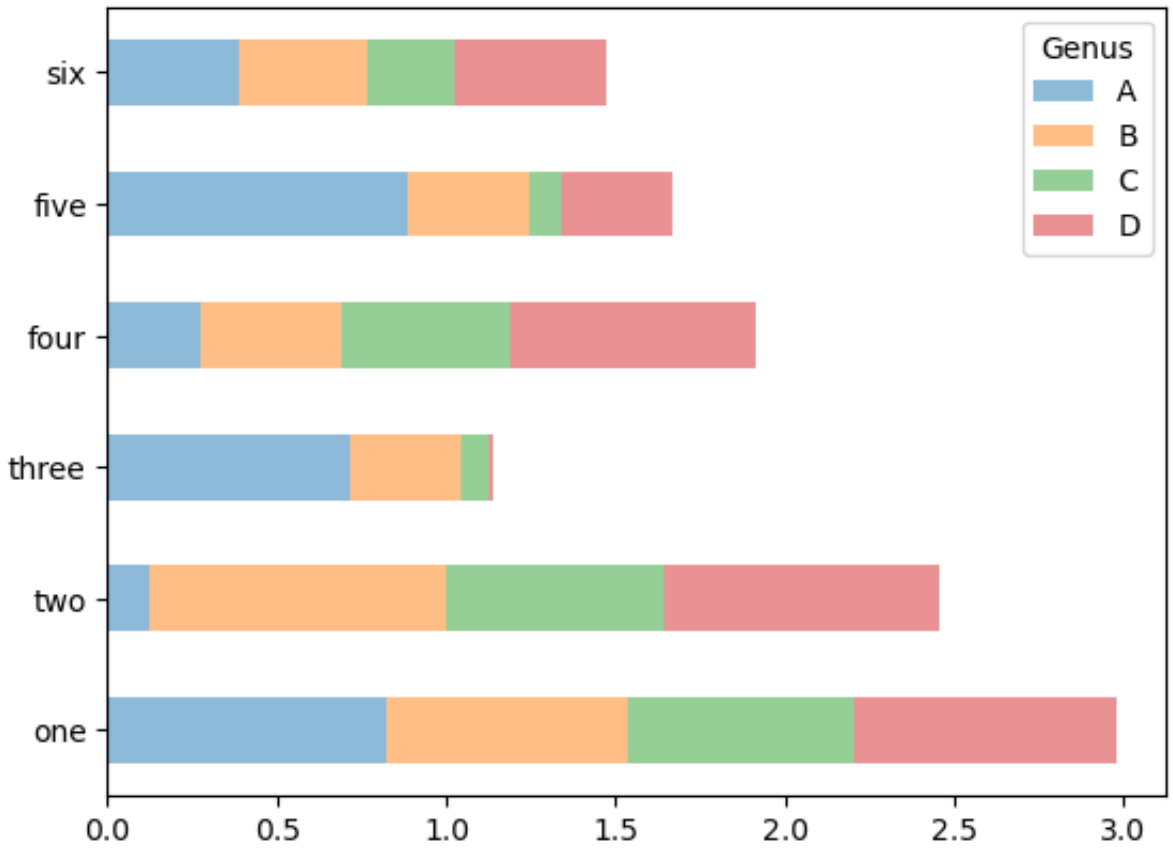
Out[221… <AxesSubplot:>



In [222…

```
# dfに対してstacked=Trueを与えてプロットすれば積み上げ棒グラフ
df.plot.barh(stacked=True, alpha=0.5)
```

Out[222… <AxesSubplot:>



In [228…

```
# 各曜日の各団体の人数
p = pathlib.Path.cwd()  / "pydata-book/examples/tips.csv"
tips = pd.read_csv(p)
tips.head(2)
```

Out[228…

	total_bill	tip	smoker	day	time	size
0	16.99	1.01	No	Sun	Dinner	2
1	10.34	1.66	No	Sun	Dinner	3

In [229…

```
# クロス集計分析 crosstab
party_counts = pd.crosstab(tips['day'], tips['size'])
party_counts.head()
```

Out[229…

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

In [231…
```
party_counts = party_counts.loc[:, 2:5]
party_counts
```

Out[231…

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 [235…
```
# sum(1)で行ごとの合計を示す
party_counts.sum(1)
```

Out[235…
```
day
Fri     18
Sat     85
Sun     75
Thur    58
dtype: int64
```

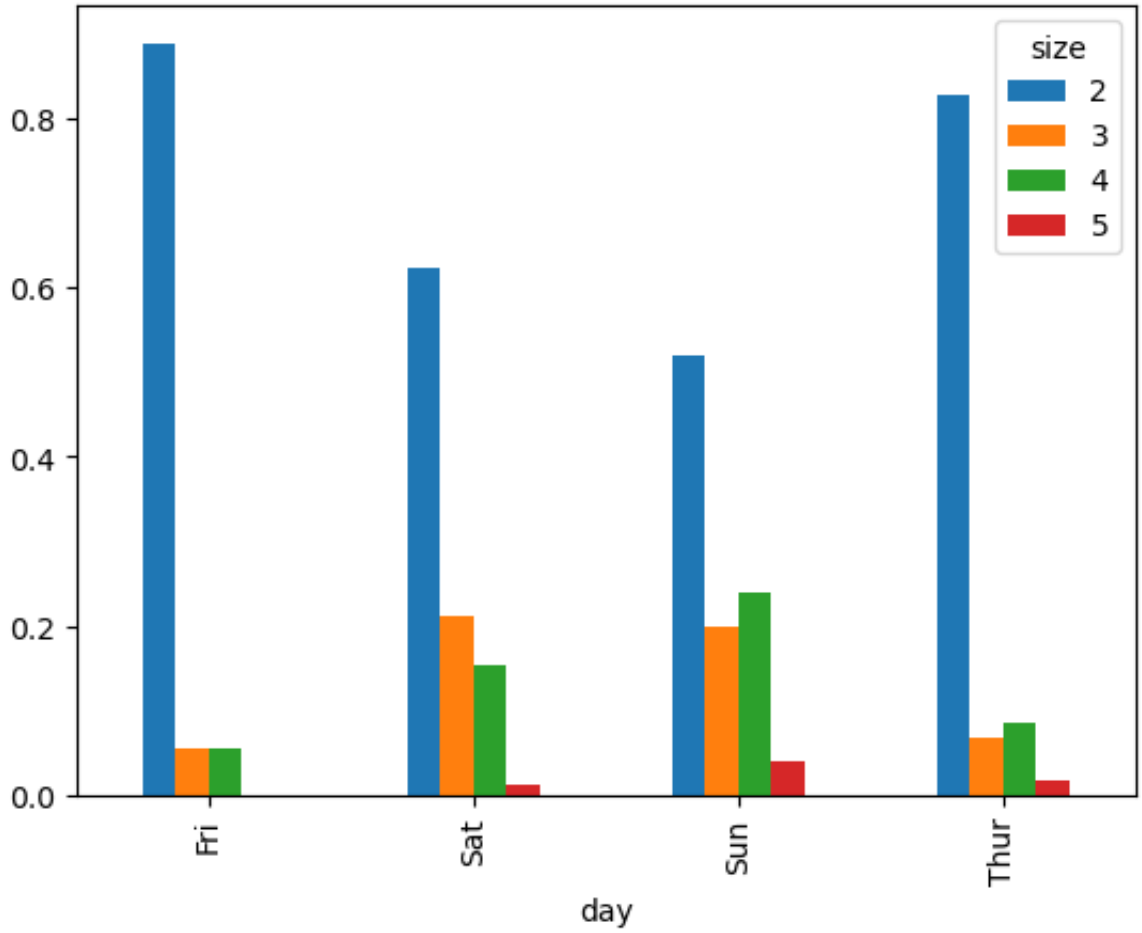
In [236…
```
# 正規化（sum(1)で行の合計）
party_pcts = party_counts.div(party_counts.sum(1), axis=0)
party_pcts
```

Out[236…

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 [249…
```
# 各曜日において人数で分類したときの団体の割合（P302）
party_pcts.plot.bar()
```

Out[249… <AxesSubplot:xlabel='day'>



- グラフを見るとこのデータセットでは団体の人数は週末に増えている
- プロットの前に集計や要約を必要とするデータについては、seabornを使用すると簡単
- 曜日ごとのチップの割合をみる

In [244…

```
import seaborn as sns

tips['tip_pct'] = tips['tip'] / (tips['total_bill'] - tips['tip'])

tips.head()
```

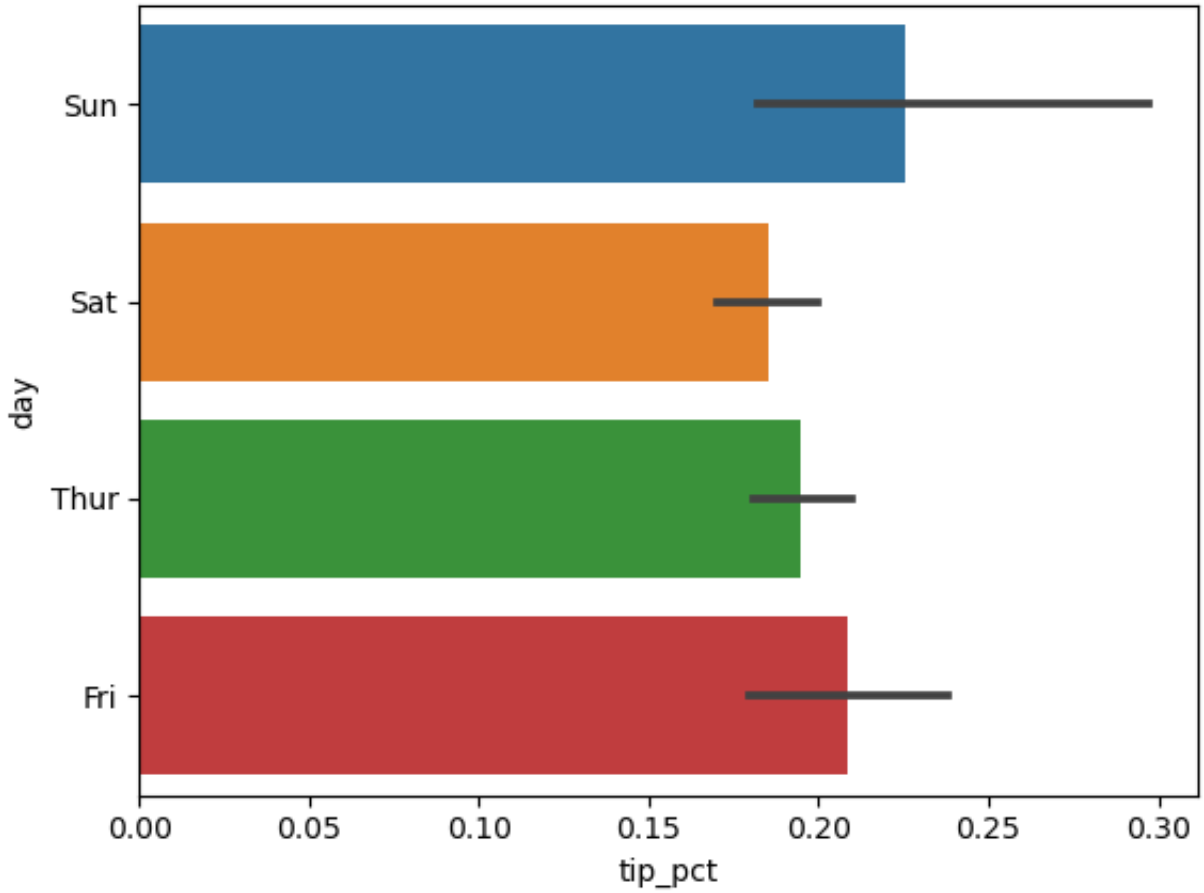
Out[244…

	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 [248…

```
# 曜日ごとのチップのパーセンテージ（エラーバー付き）（次は、P303から）
sns.barplot(x='tip_pct', y='day', data=tips, orient='h')
```

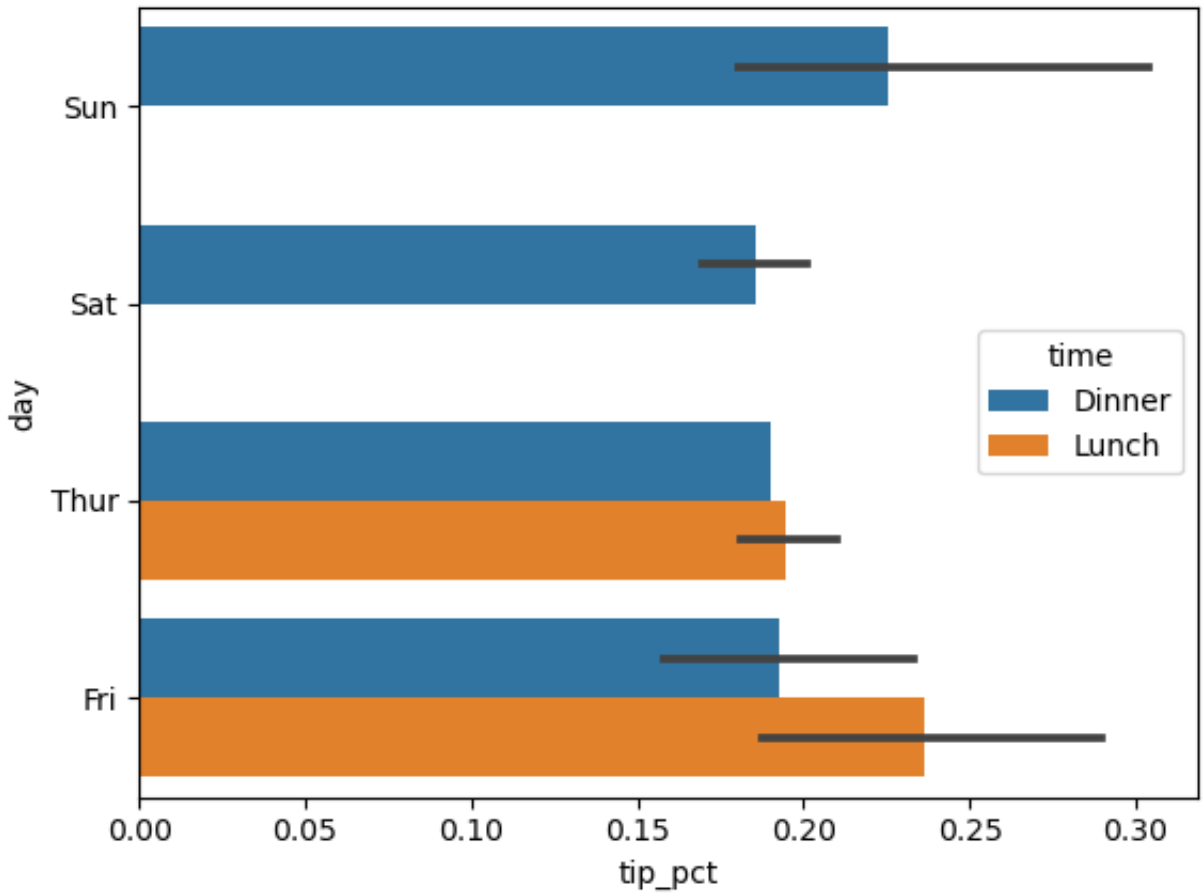
Out[248… <AxesSubplot:xlabel='tip_pct', ylabel='day'>



In [251…

```
# hue：カテゴリ型データを指定し、カテゴリごとにわけて集計
sns.barplot(x='tip_pct', y='day', hue='time', data=tips, orient='h')
```

Out[251… <AxesSubplot:xlabel='tip_pct', ylabel='day'>



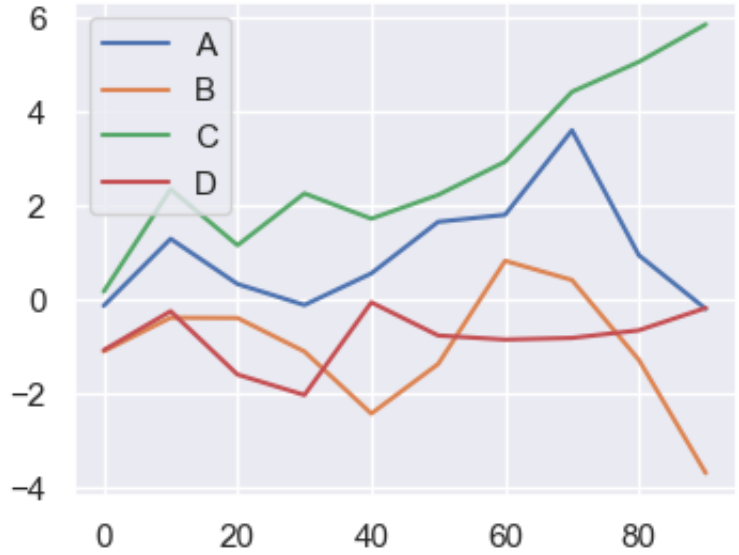
※searbornのスタイルを変える

- 他にも多々あるが詳細は公式を参照

In [267…

```
# sns.set()でデフォルトスタイルが適用される
# plotする前であればどこでもよい
sns.set()
df = pd.DataFrame(np.random.randn(10,4).cumsum(0),\
                  columns=list('ABCD'), \
                  index=np.arange(0,100,10))

#sns.set()
df.plot()
```

Out[267… <AxesSubplot:>



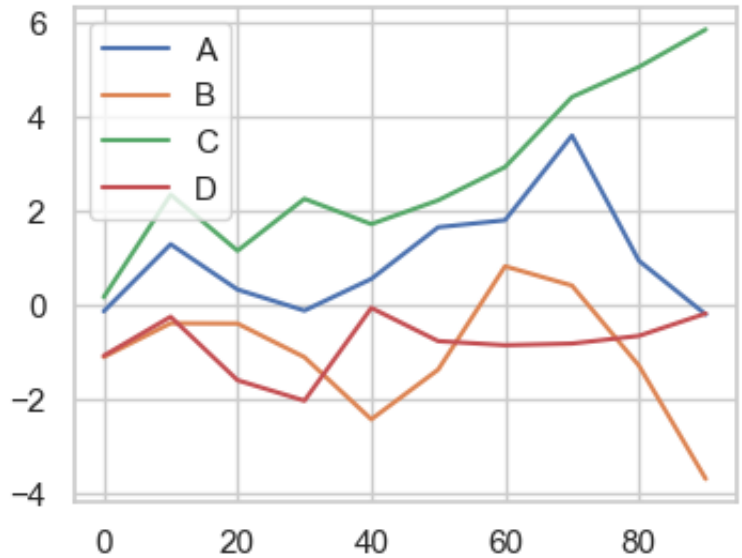
sns.set_style(***)

- darkgrid:背景暗、グリッドあり、デフォルト
- dark:背景暗、グリッドなし
- whitegrid:背景白、グリッドなし
- white:背景白、グリッドなし
- ticks:背景白、軸のみグリッドあり（通常のグラフ）

In [273…

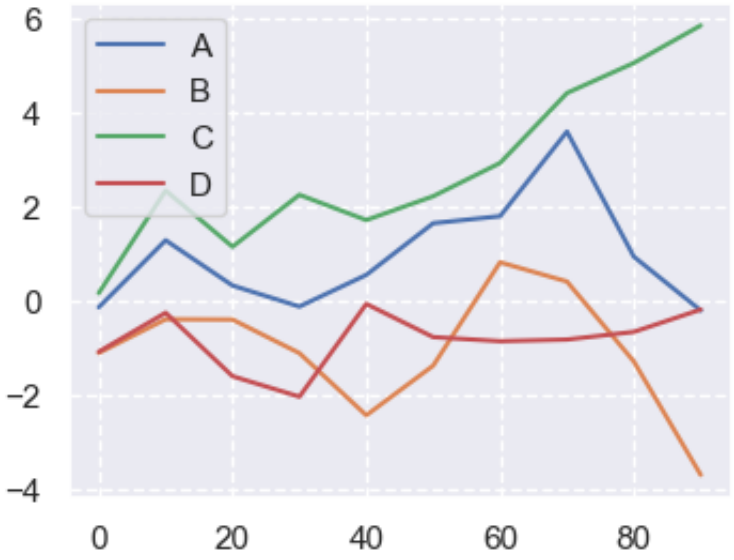
```
sns.set_style('whitegrid')
df.plot()
```

Out[273… <AxesSubplot:>



In [279…
```
sns.set_style('darkgrid', {'grid.linestyle': '--'})
df.plot()
```

Out[279… <AxesSubplot:>



In [258…
```
# デフォルトの設定値（サイズ）
plt.rcParams.get('figure.figsize')
```

Out[258… [6.4, 4.8]

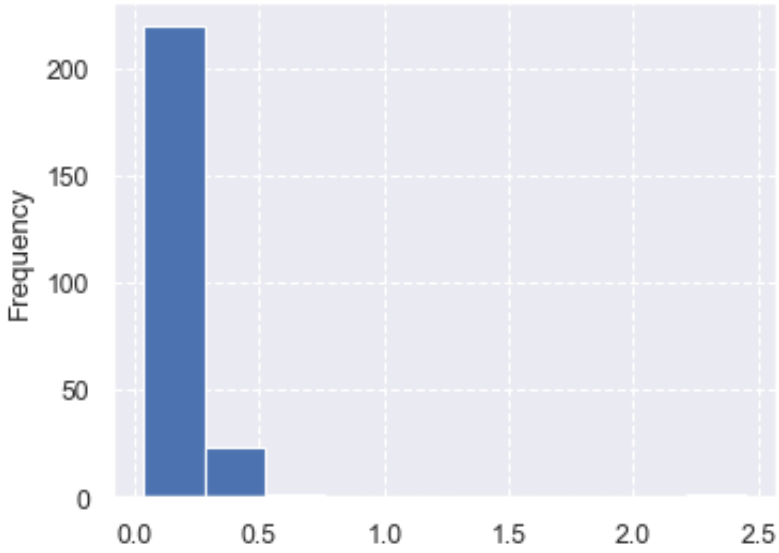
In [259…
```
# デフォルトを変更する（サイズ）
params = {
    'figure.figsize': [4, 3],
    'font.size': 8,
}

plt.rcParams.update(params)
```

9.2.3 ヒストグラムと密度プロット（P304）

ヒストグラム

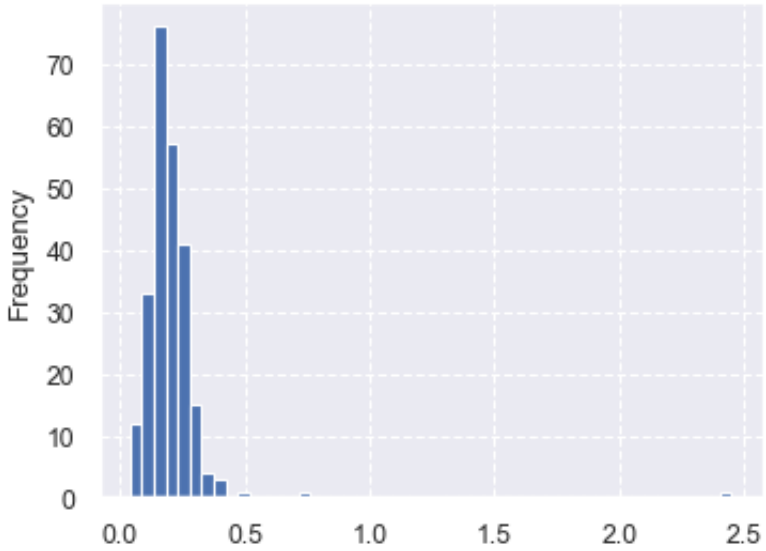
In [294…
```
tips['tip_pct'].plot.hist(bins=10)
```

Out[294… <AxesSubplot:ylabel='Frequency'>



In [295…
```
tips['tip_pct'].plot.hist(bins=50)
```

Out[295… <AxesSubplot:ylabel='Frequency'>

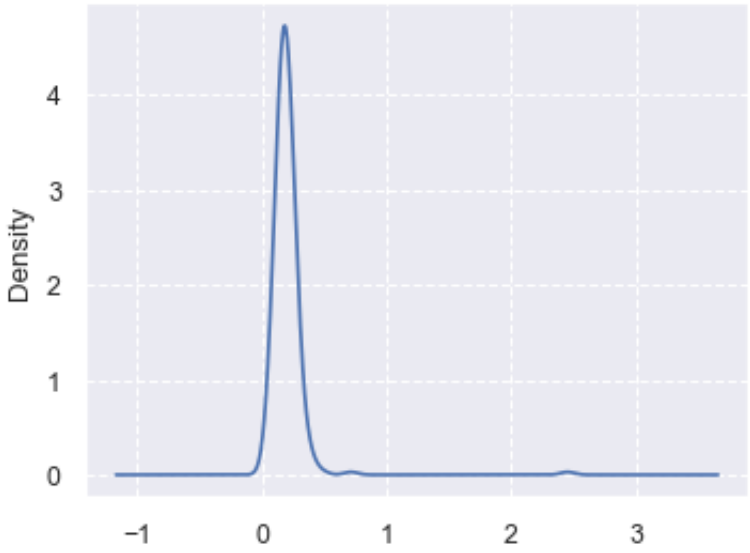


密度プロット

- カーネル密度推定（KDE）とも呼ばれる
- plot.kdeで一般的な混合正規分布カーネル密度推定を用いた密度プロットを作成できる

In [296…
```
tips['tip_pct'].plot.density()
```

Out[296… <AxesSubplot:ylabel='Density'>

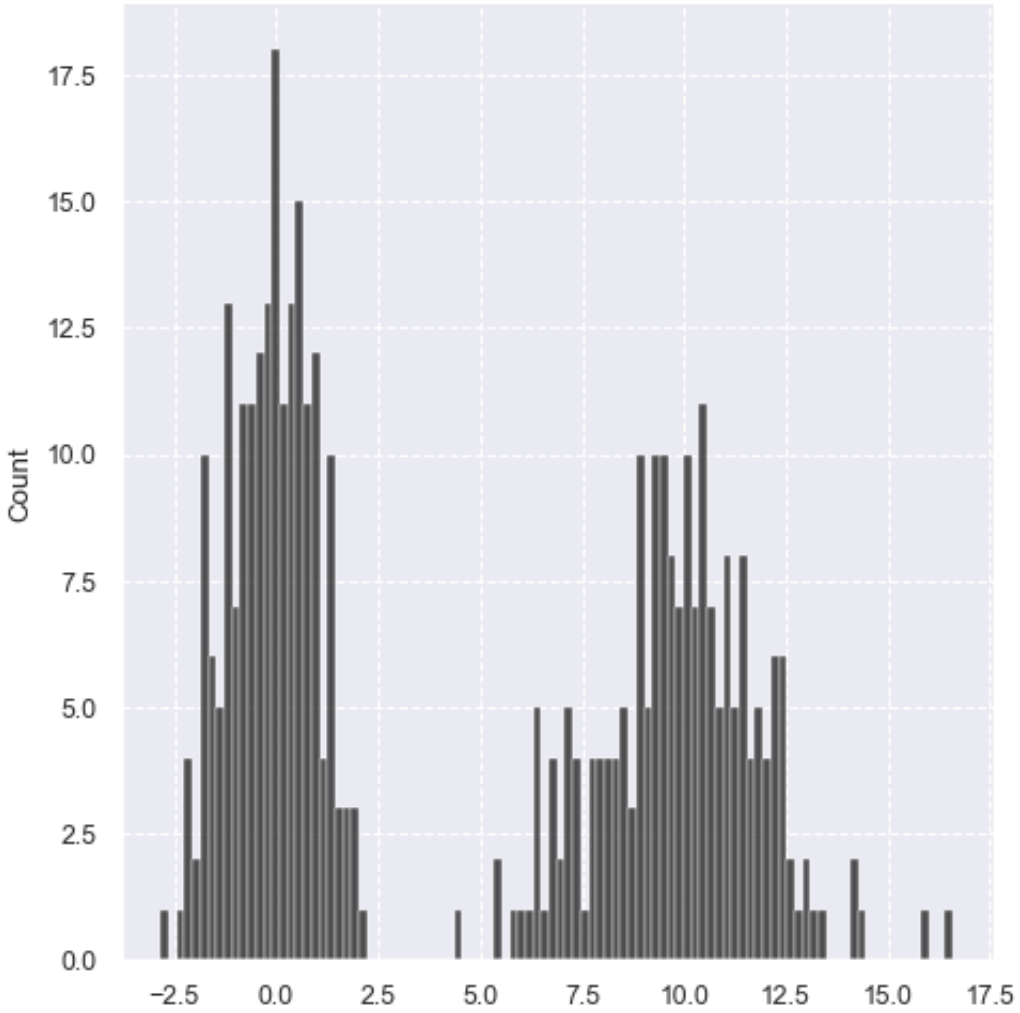


- seabornには、ヒストグラムや密度プロットをもっと簡単に作れるdisplotメソッドがある
- displotメソッドではヒストグラムと連続型の密度推定の両方のプロットを同時に作成可能

In [298…

```
com1 = np.random.normal(0,1,size=200)
com2 = np.random.normal(10,2,size=200)
values = pd.Series(np.concatenate([com1,com2]))
sns.displot(values, bins=100, color='k')
```

Out[298… <seaborn.axisgrid.FacetGrid at 0x7ff99da6e3d0>



9.2.4 散布図（P307）

In [301…

```
macro = pd.read_csv(pathlib.Path.cwd() / 'pydata-book/examples/macrodata.cs
macro.head()
```

Out[301…

	year	quarter	realgdp	realcons	realinv	realgovt	realdpi	cpi	m1	tbilrate	u
0	1959.0	1.0	2710.349	1707.4	286.898	470.045	1886.9	28.98	139.7	2.82	
1	1959.0	2.0	2778.801	1733.7	310.859	481.301	1919.7	29.15	141.7	3.08	
2	1959.0	3.0	2775.488	1751.8	289.226	491.260	1916.4	29.35	140.5	3.82	
3	1959.0	4.0	2785.204	1753.7	299.356	484.052	1931.3	29.37	140.0	4.33	
4	1960.0	1.0	2847.699	1770.5	331.722	462.199	1955.5	29.54	139.6	3.50	

In [302…

```
data = macro[['cpi', 'm1', 'tbilrate', 'unemp']]
trans_data = np.log(data).diff().dropna()
trans_data[-5:]
```

Out[302…

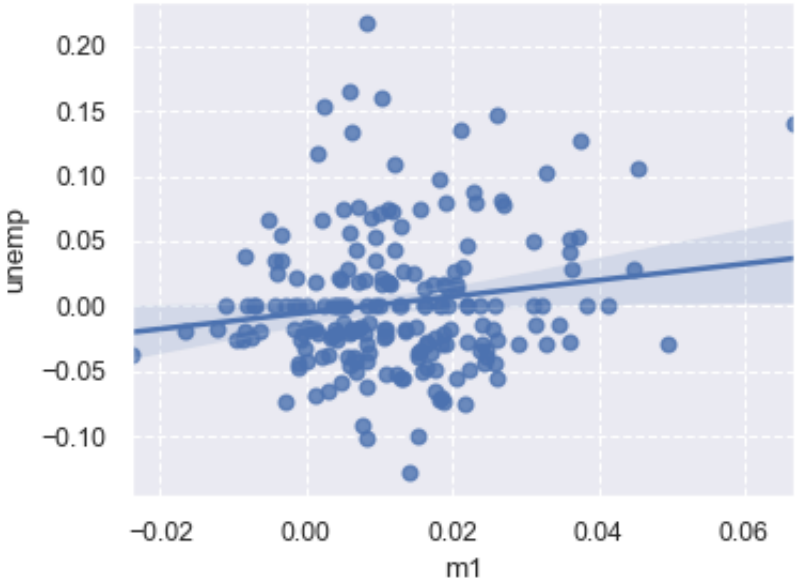
	cpi	m1	tbilrate	unemp
198	-0.007904	0.045361	-0.396881	0.105361
199	-0.021979	0.066753	-2.277267	0.139762
200	0.002340	0.010286	0.606136	0.160343
201	0.008419	0.037461	-0.200671	0.127339
202	0.008894	0.012202	-0.405465	0.042560

In [304…

```
# regplotは散布図を作成し、線形回帰により回帰直線を当てはめる
sns.regplot(x='m1', y='unemp', data=trans_data)
```

Out[304…

```
<AxesSubplot:xlabel='m1', ylabel='unemp'>
```

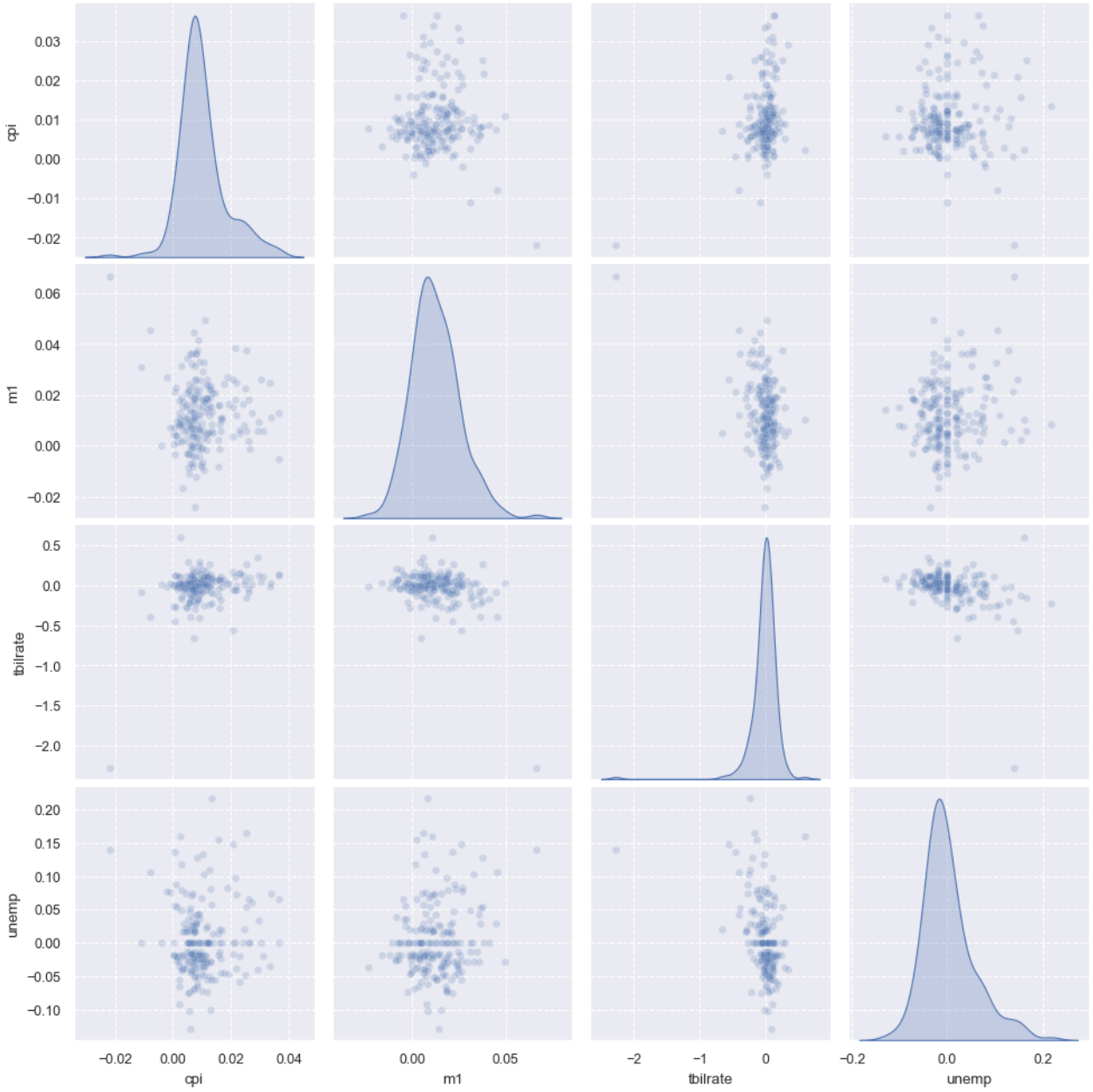


- 上記はペアプロット図や散布図行列と呼ぶ
- 0からつくるのは大変なので、seabornにはpairplotという便利な関数がある
- pairplot関数は散布図行列の対角線上に各変数のヒストグラムや密度推定を描く
- よくわからない（？）

In [305…

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

Out[305… <seaborn.axisgrid.PairGrid at 0x7ff9a5520520>



9.2.5 ファセットグリッドとカテゴリ型データ（P309）

- カテゴリ変数を含んだデータを可視化する方法：
- 特定の属性値でまとめ並べて表示できる「ファセットグリッド」
- seabornでは様々なファセットプロットを簡単に作ることができる「catplot」がある
- 以前はfactorplotだった

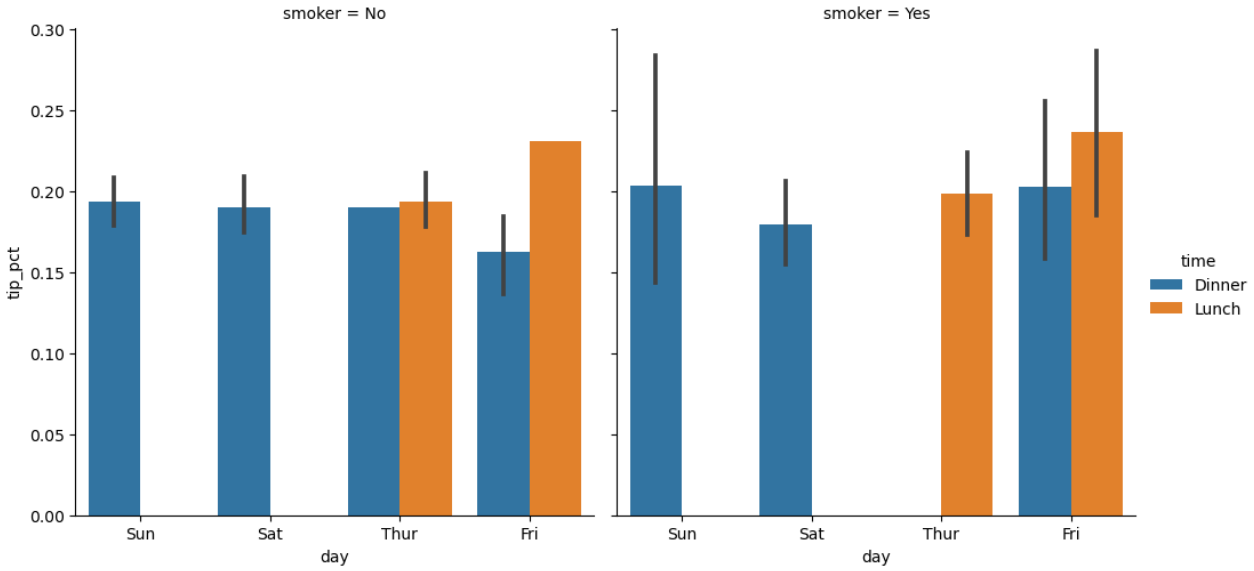
In [313…

```
plt.rcdefaults()
```

In [314…

```
# factorplotはcatplotに変わった
# ここでは2つのファセット「smoker=No」「smoker=Yes」を表示し
# それぞれのファセット内で棒の色を変えることで時間帯（'time'）をまとめている
sns.catplot(x='day', y='tip_pct', hue='time', col='smoker', \
                    kind='bar', data=tips[tips.tip_pct < 1])
```

Out[314… <seaborn.axisgrid.FacetGrid at 0x7ff9a871acd0>



In [316…

```
# 時間帯（'time'）に代わり、'time'の値ごとに行を追加すると
sns.catplot(x='day', y='tip_pct', row='time',\
            col='smoker', kind='bar', data=tips[tips.tip_pct < 1])
```

Out[316… <seaborn.axisgrid.FacetGrid at 0x7ff9a87b93a0>