10. 作図の基本

プログラミング・データサイエンス I

2022/6/30

1　今日の目的

今日の目的

- 作図の基本
  - 一枚の図を描く
  - 複数の図を描く
- 図形を描く

Python は、データ分析などで活用事例の多いプログラミング言語です。分析結果を図示することで、データから読み取れることを説明することができます。今日は、作図の基本について説明します。`matplotlib` という作図ライブラリを使うことにします。マニュアルは以下の URL にあります。

`https://matplotlib.org/contents.html`

はじめに、一枚の図を描く方法を紹介します。データを点で描く、折れ線で結ぶなどの基本的な方法から始め、図の大きさ、フォントサイズ、作図範囲の指定などの書式設定までを扱います。指定できることは非常に多岐にわたります。上記 URL のマニュアルを見るのが基本ですが、インターネット上にも沢山の例があります。

次に複数の図を描く方法を紹介します。異なる量を横軸や縦軸を揃えて見せたい時に便利です。一枚の図を描く方法とは、コマンド名が少し異なることに注意が必要です。

最後に、図形の描き方を紹介します。

今日のサンプルプログラムをダウンロードしてください。

`https://github.com/first-programming-saga/plotSample`

2　データを一枚の図に作図する

データを一枚の図に作図する

- 折れ線
- 散布図
- 図の書式設定
  - タイトル
  - 軸の名前と範囲
  - 凡例
- 図の表示と保存

2.1　作図の基本

作図の技術的内容の前に、基本的なことを確認しておきましょう。

図でデータを示すのは、それを使って読者に何かを伝えることが目的です。不適切な方法で図示すると、誤った情報や印象を伝えることになります。見た目を良くする工夫をしすぎたグラフを作成することで、特定の部分を強調してしまうこともあります。

また、図の種類は、データの性質に応じて選択するべきです。例えば、人口の時間的推移を表すならば横軸に時間を縦軸に人口をとって、折れ線グラフとするべきでしょう。あるいは、企業の従業員数と職員の平均年収の関係を示したいならば、データを点で表示する散布図を使います。各県の人口を比較したいならば、横軸を県名、縦軸を人口とした棒グラフを使います。

縦軸や横軸の範囲の選び方も重要です。人口の変化を表す際に、縦軸の範囲を狭くとると、大きく変動した印象を与えます。範囲を広くとると、変化がほとんどないように見えます。

もちろん、図にはタイトルを付け、横軸と縦軸にはラベルをつけなければいけません。単位がある量ならば、単位を示さなければいけません。見やすいように、色、線の太さ、点の大きさを調整することも必要です。

2.2 簡単なグラフ

それでは、simplePlot.ipynb を開いてください。最初の部分でライブラリをインポートしています (ソースコード 2.1)。matplotlib が最初に紹介した作図用ライブラリです。japanize_matplotlib は、図の中で日本語を使えるようにするライブラリです。

ソースコード 2.1　ライブラリのインポート

```
1 import matplotlib.pyplot as plt
2 import japanize_matplotlib
```

ソースコード 2.2　作図を行う

```
1  def drawData(xList:list[float], yList:list[float], zList:list[float]):
2      """
3      xlist-ylist を折れ線で、xlist-zlist を点で作図する
4      """
5      plt.figure(figsize = (15, 10), facecolor = 'white')#図の大きさ
6      plt.rcParams["mathtext.fontset"] = 'cm'
7      plt.rcParams['mathtext.default'] = 'it'
8      plt.rcParams["font.size"] = 28#フォントサイズ
9      plt.title('グラフタイトル')
10     plt.xlim(-1, 6)#x 軸の範囲
11     plt.ylim(0, 8)#y 軸の範囲
12     plt.xlabel('$x$')#x 軸のラベル
13     plt.ylabel('$y$')#y 軸のラベル
14
15     plt.plot(xList, yList, label = '理論', linewidth = 2)#折れ線
16     plt.scatter(xList, zList, label = 'データ',
17         color = 'red', marker = 's', linewidths = 5)#散布図
18
19     plt.legend(loc = 'upper right')#凡例の位置
20     plt.savefig('tmp.pdf')#ファイルへ出力
21     plt.show()
```

ソースコード 2.2 を見てください。5 行目から 13 行目までが、図の書式を定めている部分です。上から、図の大きさ、フォントサイズ、図のタイトル、x 軸の範囲、y 軸の範囲、x 軸のラベル、そして y 軸のラベルです。xは、数式として x を表現することを示しています。6 行目と 7 行目は、数式用のフォントの設定です。

15 行目は、xList というデータリストを x 軸の値に、yList というデータリストを y 軸の値として、折れ線でデータを結びます。label という変数を指定すると、凡例にその文字列を表示します。linewidth は線の太さです。

16 行目は、データを点で表します。color は点の色、marker は点の形、linewidth は点の大きさです。使える色と点の形は、以下の URL にあります。

color　https://matplotlib.org/3.1.0/gallery/color/named_colors.html
marker　https://matplotlib.org/3.1.0/api/markers_api.html?highlight=list%20markers

19 行目は、凡例の位置を指定しています。英語で凡例を legend と言います。loc='best'とすると、勝手に場所を決めてくれます。

作図した図をファイルに保存し (20 行目)、表示しています (21 行目)。ファイルへの保存の際には、拡張子に合わせた形式になります。ここでは、PDF という形式で保存しています。ファイルに出力した結果を図 1 に示します。

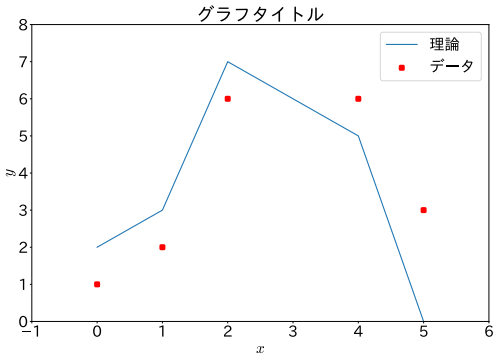

図 1　ファイルへの出力結果

課題 1　ソースコード 2.2 において、以下の変更を行ってみなさい。

1. y 軸の範囲を変更
2. 折れ線の色を設定
3. 散布図の点の形を変更

3　複数の図を描く

複数の図を描く

- 一つの図に、複数のグラフを描く
- x 軸や y 軸を共有する

3.1　軸を共有した複数の図の例 1

複数の図を縦軸、あるいは横軸を共有しながら描きたいことがあります。その場合は、`subplots()` を使用します。戻り値のうち `ax` は `Axes` という型のリストで、各要素が一つの図の領域を表します。

ソースコード 3.1　二つの図を左右に並べる

```
1  def drawData(xList:list[float], yList:list[float], zList:list[float]):
2      fig, ax = plt.subplots(1, 2, figsize = (20, 10),
3          sharey = True, facecolor = 'white')
4      plt.rcParams["mathtext.fontset"] = 'cm'
5      plt.rcParams['mathtext.default'] = 'it'
6      plt.rcParams['axes.titlesize'] = 35#フォントサイズ
7      plt.rcParams['axes.labelsize'] = 35#フォントサイズ
8      plt.yticks([-1 + .5 * i for i in range(6)])#y 軸の目盛り
9      #二つの図に共通の設定
10     for a in ax:
11         a.set_xlim(-5, 5)
12         a.set_xlabel('$x$')
13         a.tick_params(axis = 'both', length = 15)
14
15     #最初の図
16     ax[0].set_title('$\sin(x)$')
17     ax[0].plot(xList, yList)
18     #2 番目の図
19     ax[1].set_title('$\cos(x)$')
20     ax[1].plot(xList, zList)
21
22     plt.savefig('multiPlot.pdf')
23     plt.show()
```

それでは、`multiPlot.ipynb` を見ましょう。作図をしている部分がソースコード 3.1

です。subplot の最初の二つの引数 1,2 が一行二列の作図領域を作ることを表しています。つまり、横に 2 つ図が並びます。作図領域は ax として返ってきます。今回は、ax[0] と ax[1] です。最後の引数 sharey = True が y 軸を共有することを示しています。つまり、縦軸は共通の値となります。

4 行目と 5 行目は、数式フォントの設定です。7 行目は、y 軸の目盛り (tick と言います) とラベル設定です。-1 から 0.5 刻みで、1 まで目盛りを作ります。

10 行目から 13 行目は、二つの図に共通の要素を指定しています。ax がリストであったことを思い出してください。x 軸の範囲、x 軸のラベル、そして目盛りの大きさです。

16 行目と 17 行目で ax[0] に、19 行目と 20 行目で ax[1] に、それぞれタイトルを付け、データを折れ線で描いています。

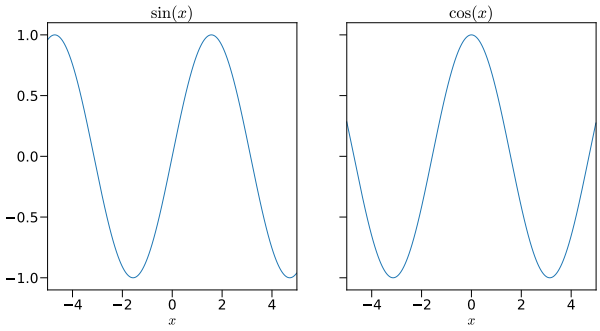

図 2　二つの図を描く

作図結果を図 2 に示します。変数名 x がきれいなイタリック体で、三角関数の名前がローマン体で表示されています。また、y 軸は、左の図だけに数値が入っています。

3.2　軸を共有した複数の図の例 2

次は、4 つの図を、x 軸と y 軸を共通として描く例です。作図結果は図 3 のようになります。multiPlot2.ipynb を開けてください (ソースコード 3.2)。

3 行目の最初の二つの引数 2,2 で、2×2 という四つの図を作ることを指定しています。最後の二つの引数が x 軸と y 軸を共通として描くことの指定です。結果として返ってくる ax も 2×2 のリストです。

四つの図に対する共通的設定を 13 行目から 19 行目で行っています。13 行目の最初の for は、行に関する繰り返しです。14 行目の内側の for は、列に関する繰り返しです。繰り返し方が違いますが、理解できますか。18 行目では、行の番号を見て、一番最後の行

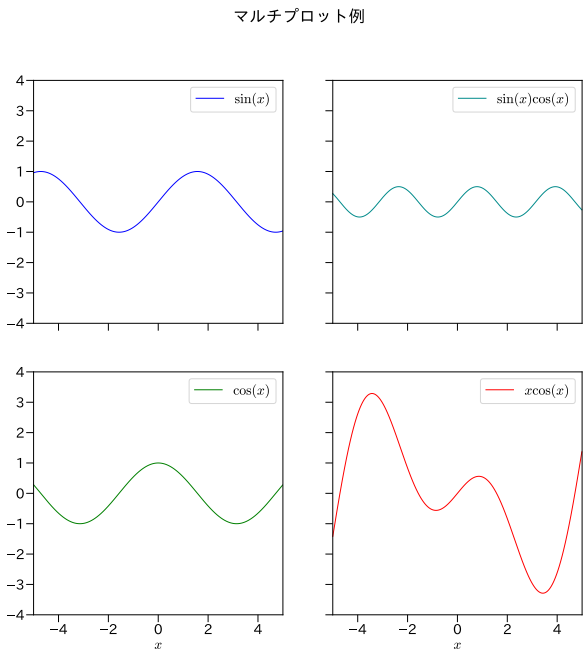

図 3　四つの図を描く

に対してだけ、x 軸のラベルを描いています。

21 行目から、4 つの図を折れ線で描いています。最後に、27 行目からの二重 `for` ループで、凡例を作っています。

4　その他の作図

その他の作図

- 棒グラフ
- 円グラフ
- ヒストグラム

4.1　棒グラフ

`matplotlib` では、様々な型のグラフを描くことができます。ここでは、三種類を示します。

最初は、棒グラフです。`plotBars.ipynb` を見てください (ソースコード 4.1)。例とし

ソースコード 3.2　4 枚の図を描く

```
def draw(x:list[float], y1:list[float], y2:list[float],
        y3:list[float], y4:list[float]):
    #xy 軸のスケールを揃える
    fig, ax = plt.subplots(2, 2, figsize = (20, 20),
        sharex = True, sharey = True, facecolor = 'white')
    plt.rcParams["mathtext.fontset"] = 'cm'
    plt.rcParams['mathtext.default'] = 'it'
    plt.rcParams['font.size'] = 35#フォントサイズ
    plt.rcParams['axes.titlesize'] = 35#フォントサイズ
    plt.rcParams['xtick.labelsize'] = 35

    fig.suptitle('マルチプロット例')
    #全ての図に共通の設定
    for i in range(len(ax)):
        for b in ax[i]:
            b.set_xlim(-5, 5)
            b.set_ylim(-4, 4)
            b.tick_params(axis = 'both', length = 15)
            if i == 1:#2 行目だけ x 軸ラベルを表示
                b.set_xlabel('$x$')

    ax[0,0].plot(x, y1, label = '$\sin(x)$', color = 'blue')
    ax[1,0].plot(x, y2, label = '$\cos(x)$', color = 'green')
    ax[0,1].plot(x, y3, label = '$\sin(x) \cos(x)$', color =
    ↪  'darkcyan')
    ax[1,1].plot(x, y4, label = '$x \cos(x)$', color = 'red')

    #全ての図の凡例
    for a in ax:
        for b in a:
            b.legend(loc = 'best')

    plt.savefig('multiPlot2.pdf')
    plt.show()
```

て、九州各県の人口を棒グラフで表しましょう。

1 行目のリスト population は、九州各県の 2015 年の人口です。棒グラフの高さに相当します。2 行目のリスト prefecture は、九州各県の名前です。棒グラフの横軸に相当します。

9 行目から 12 行目に書けて、軸のラベルや目盛りを設定しています。

14 行目の bar() の最初の引数が横軸、二番目の引数が縦軸です。ただし、横軸は、リスト population の番号を示しているに過ぎません。リスト x は 4 行目で生成していま

す。横軸の名前は tick_label として渡しています。出力結果が図 4 です。

ソースコード 4.1　棒グラフ

```
population = [5120, 847, 1413, 1818, 1191, 1136, 1691]
prefecture = ['福岡', '佐賀', '長崎', '熊本',
              '大分', '宮崎', '鹿児島']
x = [i for i in range(len(population))]

plt.figure(figsize = (15, 15), facecolor = 'white')
plt.rcParams['font.size'] = 28

plt.title('九州の人口，2015', fontsize = 28)
plt.xlabel('県', loc = 'right')
plt.ylabel('人口（千人）', loc = 'top')
plt.tick_params(axis = 'both', length = 15)

plt.bar(x, population, tick_label = prefecture)

plt.savefig('plotBar.pdf')
plt.show()
```

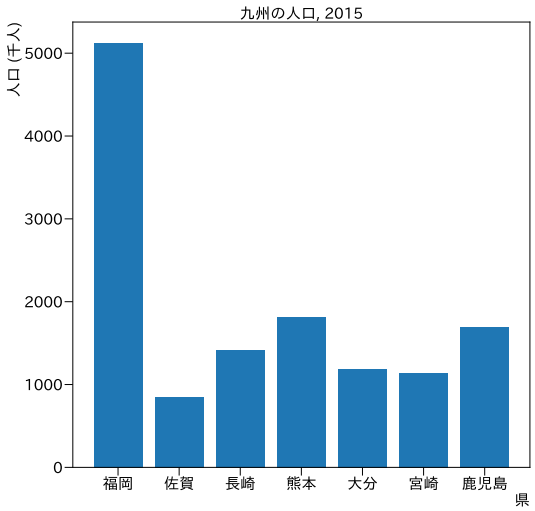

図 4　棒グラフ

4.2　円グラフ

次の例は、円グラフです。英語では、パイグラフと言います。お菓子のパイのことです。piPlot.ipynb を開けてください (ソースコード 4.2)。佐賀県の人口構成の変化を表すグラフです。人口を年少 (14 歳以下)、生産年齢 (15 歳以上、65 歳未満)、そして老年 (65 歳以上) に分けて、1995 年と 2015 年で比較します。

ソースコード 4.2　円グラフ

```
1  data = [
2          [160307, 566671, 157329],#1995
3          [116122, 483019, 229335] # 2015
4  ]
5  label = ['年少', '生産年齢', '老年']
6  color = ['y', 'c', 'r']
7  year = [1995,2015]
8
9  fig,ax = plt.subplots(1, 2, figsize = (20, 10), facecolor='white')
10 plt.rcParams['font.size'] = 24 #文字の大きさ
11 fig.suptitle('佐賀県の人口') #全体のタイトル
12
13 for k in range(2):
14         ax[k].pie(data[k], labels = label, colors = color,
15         startangle = 90, counterclock = False, autopct = '%1.1f%%')
16         ax[k].set_title(str(year[k]))
17
18 plt.savefig('piPlot.pdf')
19 plt.show()
```

リスト data に、1995 年と 2015 年の、三つの分類に相当する人口のリストを記述しています。リスト years は、タイトルに使う 2 つの年です。9 行目で、一行二列の図を作っています。

13 行目からの for ループで、二つの図を描いています。14 行目が円グラフを描く命令です。最初の引数が描くデータ、二番目が各データに対応するラベルです。一番最後が、数値を描く書式の指定で、小数以下一桁まで百分率で表すことを指定しています。

結果を図 5 に示します。matplotlib の円グラフのメソッド pie() は、データを与えると、自動的に百分率を計算してくれていることにも注意してください。

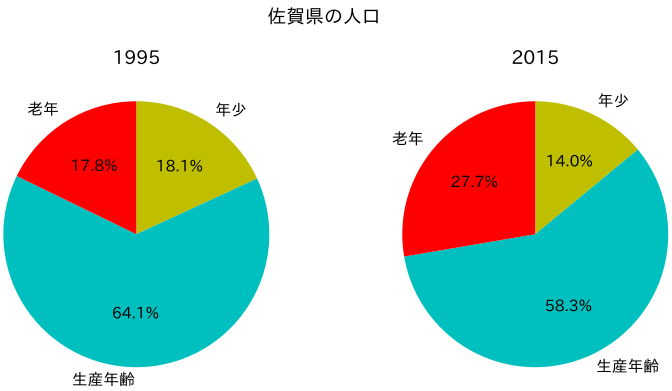

図 5　円グラフ

4.3　ヒストグラム

最後の例はヒストグラムです。histogram.ipynb を開いてください (ソースコード 4.3)。ヒストグラム (histogram) とは、頻度を表す図のことです。

ソースコード 4.3　ヒストグラム

```
#元データ
data = [2.5, 4., 5.5, 9.2, 0.7, 8.8, 6.1, 7.1, 4.3, 1.6,
        3.4, 1.4, 7.7, 4.4, 5.5, 9.6, 6.8, 8.6, 2.1,
        8.9, 2.9, 1.1, 2.1, 3.7, 2.0, 1.4, 0.5, 5.7]
plt.figure(figsize = (15, 15), facecolor = 'w')
plt.rcParams['font.size'] = 24
plt.title('ヒストグラムの例')

numBins = 10 #bin の数
rWidth = 0.9 #描く幅は、等間隔の幅に対して 0.9 倍
plt.hist(data, range=(0, 10), bins = numBins,
        rwidth = rWidth, color = 'cyan')

plt.savefig('histogram.pdf')
plt.show()
```

1 行目の data には、0 以上 10 未満の小数のデータが入っています。このデータを 0.1 刻みの区間 (bin という) に分けて、各 bin に何個入っているかを図示しましょう。通常で

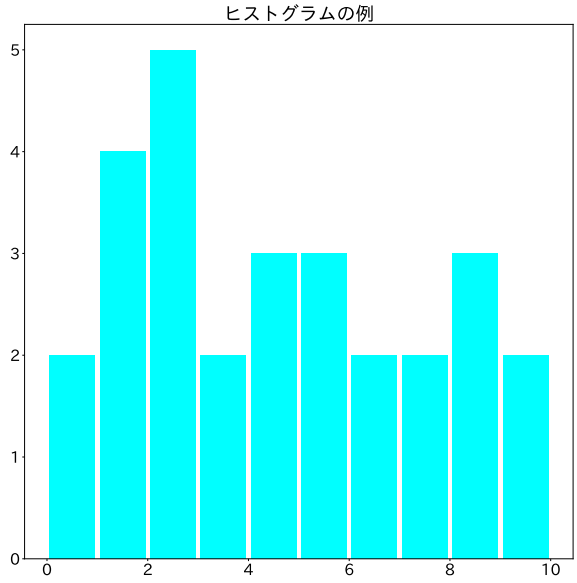

図 6　ヒストグラム

は、各 bin に入っている数を数えるプログラムを書く必要がありますが、`matplotlib` を使えば、自動的に数えてくれます。

9 行目の `hist()` メソッドを見てください。最初から、データ、データの区間、bin の数を指定します。`rwidth` は、作図する際に bin の間隔に対する棒グラフの相対的な大きさです。

結果は、図 6 のようになります。

5　図形を描く

図形を描く

- 既定の図を描く
- 点列から図を描く

作図の最後に、図を描くことを考えましょう。`drawShapes.ipynb` を開けてください。

図を描くには、二つの方法があります。一つ目は、既存の図形を描く方法です。ソースコード 5.1 の 6 行目から 11 行目は、円、四角形、正多角形という既存の図形を描いてい

ます。既存の図形の描き方は以下の URL にあります。

https://matplotlib.org/api/patches_api.html

ソースコード 5.1　図形を作図

```
1  def drawPatches(ax:plt.Axes):
2      """
3      図形を生成
4      """
5      patches=[]
6      #円
7      patches.append(pts.Circle((2,2),1,color='red'))
8      #四角
9      patches.append(pts.Rectangle((4,7),4.,2.,color='blue'))
10     #正多角形
11     patches.append(pts.RegularPolygon((7,3),5,2,color='yellow'))
12     #任意の点の列
13     verts = [(2.,8.),(3,4),(4,9),(6,2),(8,3)]
14     verts.append(verts[0])
15     path = createPath(verts)
16     patches.append(path)
17     #図形を追加
18     for p in patches:
19         ax.add_patch(p)
```

もう一つの方法は、座標の列を使って定義する方法です。ソースコード 5.1 の 13 行目で、座標の列を与えています。注意が必要なのは、最後に出発点の座標を追加しておく必要があることです (11 行目)。

ソースコード 5.2　自分で点列を定義する

```
1  def createPath(vertList:list[tuple[float,float]])->pts.Patch:
2      """
3      頂点の列から多角形を生成
4      """
5      n = len(vertList)
6      #codes は、各座標への操作
7      codes = [Path.MOVETO]#最初は移動
8      for i in range(n-2):#残りは線を引く
9          codes.append(Path.LINETO)
10     codes.append(Path.CLOSEPOLY)#最後に閉じる
11     path = Path(vertList,codes)#座標と操作を与える
12     return pts.PathPatch(path,facecolor='orange',lw=2)
```

図を定義する過程を説明します。少しイメージしにくいかも知れません。昔、プリンターで図が描けなかった頃、図を描く専用のプロッター (plotter) という機材がありました。

https://ja.wikipedia.org/wiki/%E3%83%97%E3%83%AD%E3%83%83%E3%82%BF%E3%83%BC

プロッターの操作は、

moveto　ペンを、線を描かずに指定した座標に移動する
lineto　ペンを線を引きながら、指定した座標に移動する
close　指定した座標に向かって図形を閉じる

という動作から構成します。これは、今でも、作図用のライブラリで共通に使われています。最後の動作は、最初の点に向けて図形を閉じるというものが多いのですが、matplotlib では、最後の点も指定する必要があります。

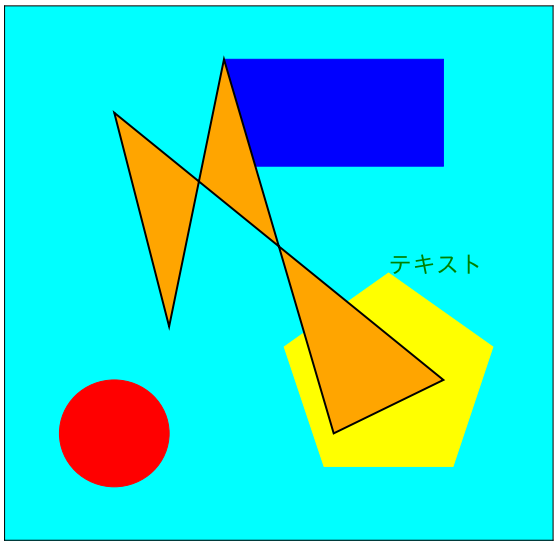

図 7　作図

ソースコード 5.1 を見てください。点を結んで図形を描くために、13 行目で点のリストを定義します。14 行目では、始点を最後に追加しています。

このリストを使って図形を、ソースコード 5.2 で定義します。最初の点へ MOVETO した

あと、順次 LINETO し、最後に CLOSEPOLY します。

6　次回

今回は、データをリストなどで作成し、作図を行いました。Excel では、内部で作図をできますね。Pandas のデータでも、matplotlib の機能と連携して、作図をすることができます。次回のテーマです。