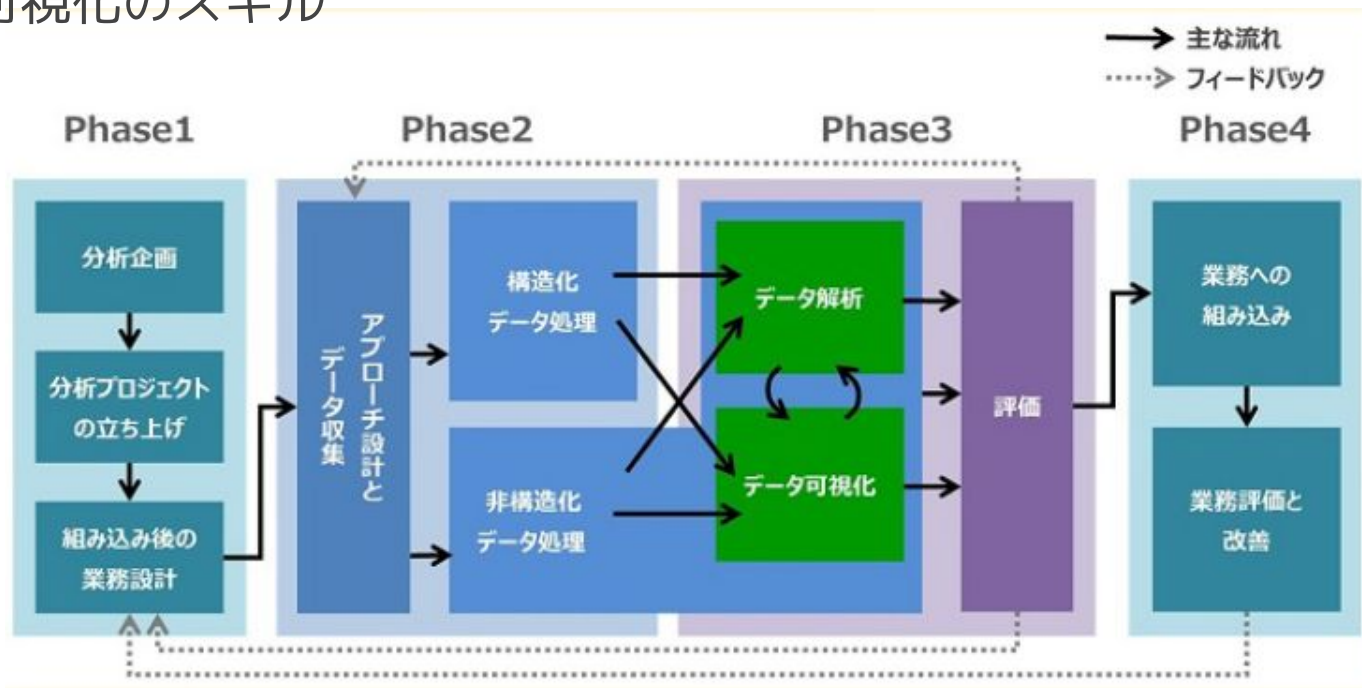


# Week4 Pythonによるデータ可視化 (Matplotlib)

講師・スライド作成：中内

今回学ぶのは探索的データ分析（EDA）のスキルセットの一部としてのデータ可視化のスキル



<https://www.ipa.go.jp/jinzai/itss/itssplus.html>

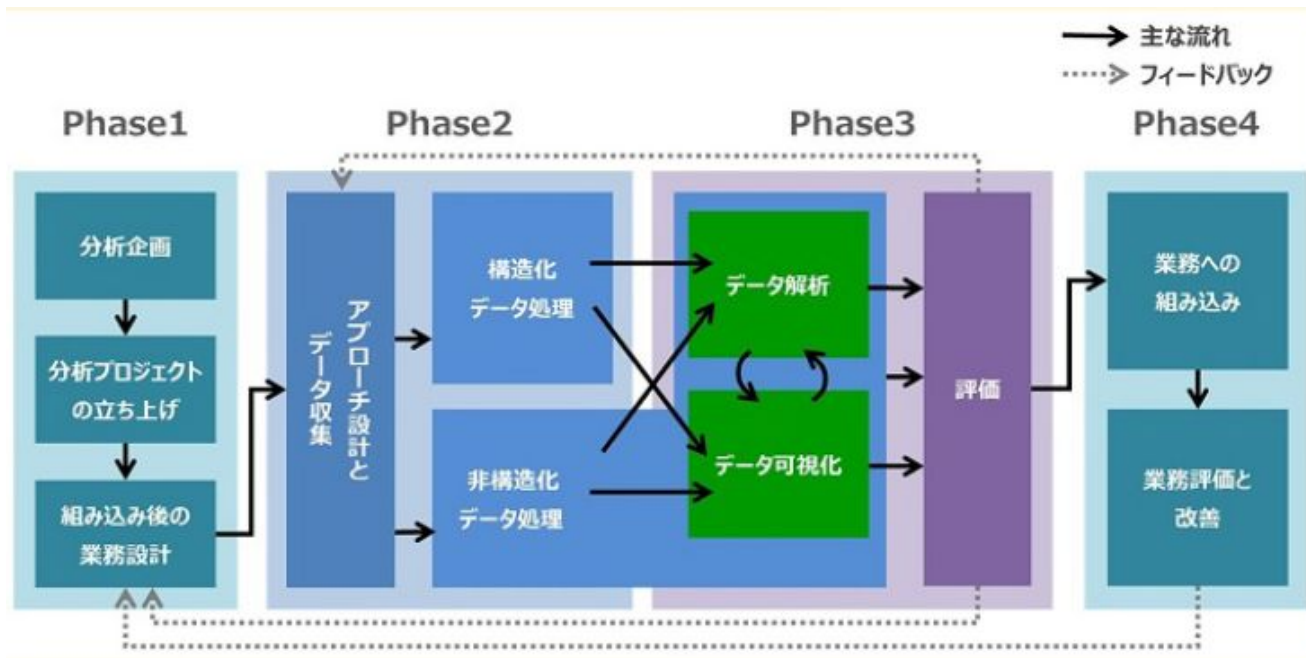
## 探索的データ分析（EDA）

- データセットに適宜前処理を施しつつ様々な統計量を抽出して可視化し、そこに内在する特性・パターン・偏りについて探索的に仮説立案・検証を繰り返して分析すること
- 「探索的」であることが重要であり、必然的に「試行錯誤」を重ねることになる
- ①理論やドメイン知識などの事前知識を活用した仮説立案と、②バイアスを排した特徴観察の両方が重要（知的好奇心も大事）

## 探索的データ分析（EDA）

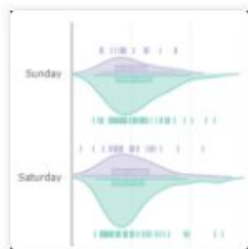
- データサイエンスの文脈では、データ可視化の知識は一義的にはEDAを効果的に行うためのスキル
- これとは別に「コミュニケーションスキルとしての可視化」という必要性もあるが、これと「EDAとしての可視化」では若干考え方や必要なスキル構成要素が異なることに注意すべき

- 可視化することで、クライアント企業に提起すべき課題を自分が発見できる
- 可視化することで、クライアント企業に課題を伝えることができる
- 各フェーズでそれぞれの可視化が必要

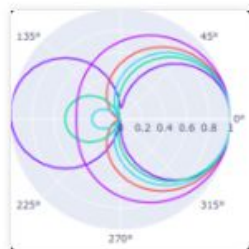


<https://www.ipa.go.jp/jinzai/itss/itssplus.html>

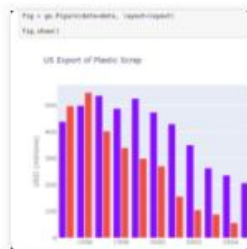
# データ可視化



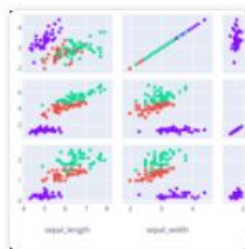
The Figure Data Structure



Creating and Updating Figures



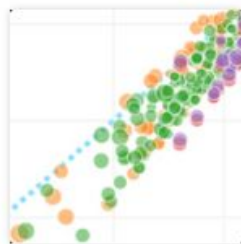
Displaying Figures



Plotly Express



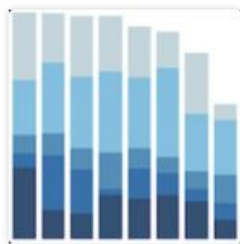
Analytical Apps with Dash



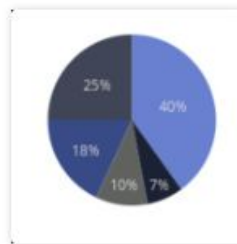
Scatter Plots



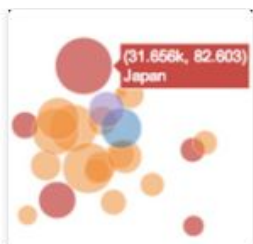
Line Charts



Bar Charts



Pie Charts



Bubble Charts

Pythonにおけるデータ可視化では  
主にMatplotlibというライブラリを使う



# matplotlibには2つの異なるインターフェイスがある

## matplotlib

### 【概要】

matplotlibの本体

### 【特徴】

グラフの各部位を逐一定義していく方法をとっている

### 【長所】

非常に細かいところまで図を調整できる  
図の逐次的な変更がしやすい

### 【短所】

pyplotよりも**多くのパラメータを指定する必要がある**

## matplotlib.pyplot

### 【概要】

元々は**簡易版的**な位置づけ（※MATLABに近似的なインターフェイスという側面もある）

### 【特徴】

試行錯誤的なグラフ作成に適している

### 【長所】

**細かいパラメータの指定なしに短いコード**でも使える

### 【短所】

極端に細かい作り込みをしたい時に制約が生じる可能性がある



# 最も手軽には数行でグラフ化できる(matplotlib.pyplot)

① 描画したいデータを用意

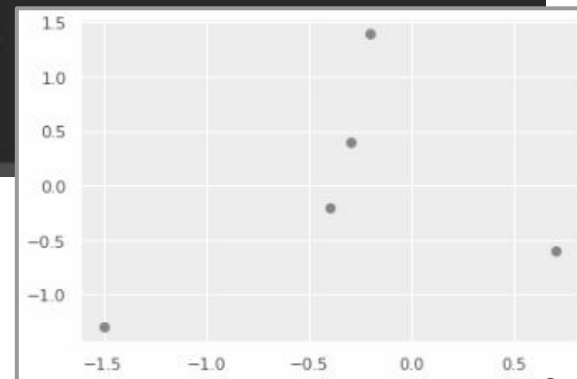
```
# x軸とy軸のそれぞれに表示したいデータ  
x = [-1.5, 0.7, -0.4, -0.2, -0.3]  
y = [-1.3, -0.6, -0.2, 1.4, 0.4]
```

② 各軸にどのデータを表示するか、グラフの種類は何か、の2点を最小限指定する

```
# グラフを描画  
plt.plot(x, y, 'o')
```

③ グラフ表示を指示  
(JupyterNotebookでは省略可)

```
# グラフを表示  
plt.show()
```



最小限ならたったこれだけで描画できる  
=グラフ作成は少しも面倒ではないという理解が大事

# 最も手軽には数行でグラフ化できる(matplotlib.pyplot)

次のようにモジュールをインポートしておく

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

グラフを描画する関数

`plt.plot()`

x軸, y軸のデータを引数に渡す

グラフを表示する関数

`plt.show()`



```
# x軸とy軸のそれぞれに表示したいデータ
```

```
x = [-1.5, 0.7, -0.4, -0.2, -0.3]
```

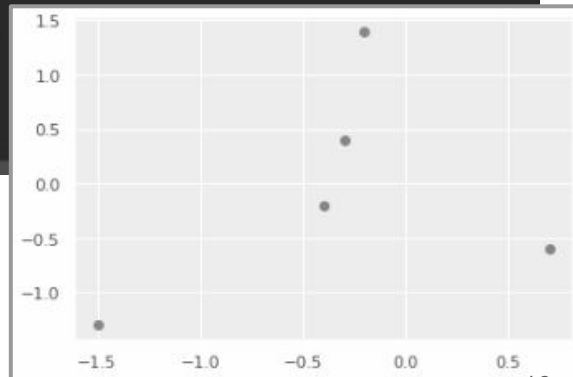
```
y = [-1.3, -0.6, -0.2, 1.4, 0.4]
```

```
# グラフを描画
```

```
plt.plot(x, y, 'o')
```

```
# グラフを表示
```

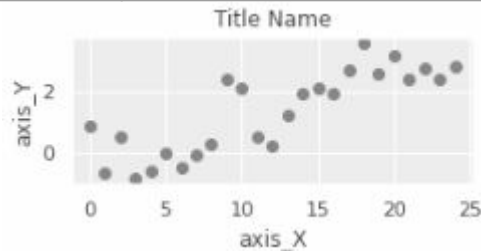
```
plt.show()
```



# マーカーを変更する

3つ目の引数に指定する記号  
(`'o'`の部分) によってグラフの  
マーカーを簡単に変更できる

<code>'o'</code>	散布図
<code>'-'</code>	折れ線グラフ
	その他数十種類

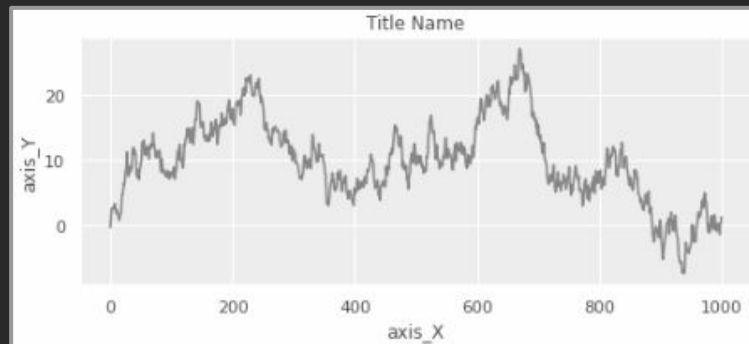


```
# 仮にこんなデータがx, yに入っていたとする
x = np.arange(1000)
y = np.random.randn(1000).cumsum()
```

```
# 前のスライドで説明した部分
plt.title('Title Name')
plt.xlabel('axis_X')
plt.ylabel('axis_Y')
```






```
# 3つ目の引数でグラフ種類を変更できる
plt.plot(x, y, '-')
```

```
plt.show()
```



marker	symbol	description
"."		point
"."		pixel
"o"		circle
"v"		triangle_down
"^"		triangle_up
"<"		triangle_left
">"		triangle_right
"1"		tri_down
"2"		tri_up
"3"		tri_left
"4"		tri_right
"8"		octagon
"5"		square
"p"		pentagon
"p"		plus (filled)

"s"		star
"h"		hexagon1
"H"		hexagon2
"+"		plus
"x"		x
"X"		x (filled)
"D"		diamond
"d"		thin_diamond
" "		vline
"_"		hline
0 (TICKLEFT)		tickleft
1 (TICKRIGHT)		tickright
2 (TICKUP)		tickup
3 (TICKDOWN)		tickdown
4 (CARETLEFT)		caretleft
5 (CARETRIGHT)		caretright
6 (CARETUP)		caretup
7 (CARETDOWN)		caredown

8 (CARETLEFTBASE)		caretleft (centered at base)
9 (CARETRIGHTBASE)		caretright (centered at base)
10 (CARETUPBASE)		caretup (centered at base)
11 (CARETDOWNBASE)		caredown (centered at base)
"None", " " or ""		nothing
'\$...\$'		Render the string using mathtext. E.g "\$f\$" for marker showing the letter f.
verts		A list of (x, y) pairs used for Path vertices. The center of the marker is located at (0, 0) and the size is normalized, such that the created path is encapsulated inside the unit cell.
path		A Path instance.
(numsides, 0, angle)		A regular polygon with numsides sides, rotated by angle.
(numsides, 1, angle)		A star-like symbol with numsides sides, rotated by angle.
(numsides, 2, angle)		An asterisk with numsides sides, rotated by angle.

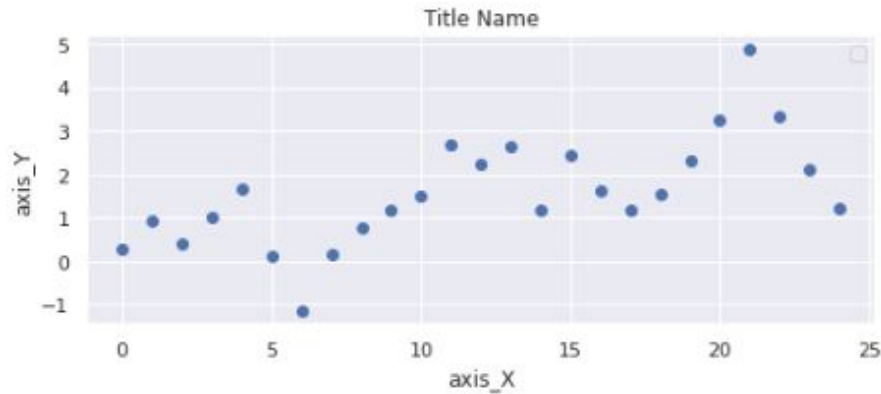
公式ドキュメントを参照すると無数に用意されてあるマーカーの種類を簡単に参照できる  
[https://matplotlib.org/stable/api/markers\\_api.html](https://matplotlib.org/stable/api/markers_api.html)

# グラフの種類ごとに用意された関数を使う

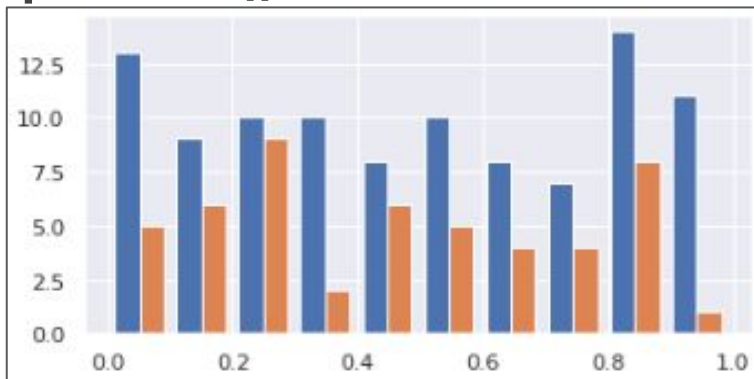
- 前のスライドでは右図①の方法でグラフの種類を指定した
- 更にさまざまなグラフ種類を試したい場合は、②の方法で指定することもできる
- ②の方法の場合、「**scatter**」の部分を様々なグラフの名前に置き換えることで多様なグラフを使い分けられる

① `[ ] plt.plot(x, y, 'o')`

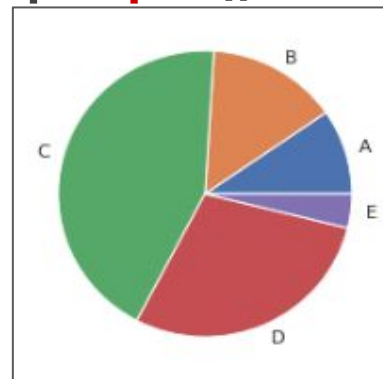
② `[ ] plt.scatter(x, y)`



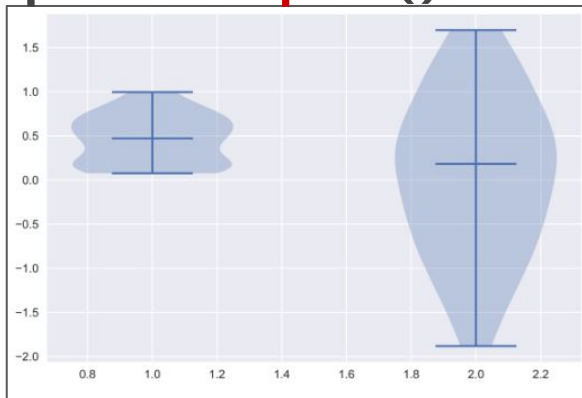
**plt.hist()**



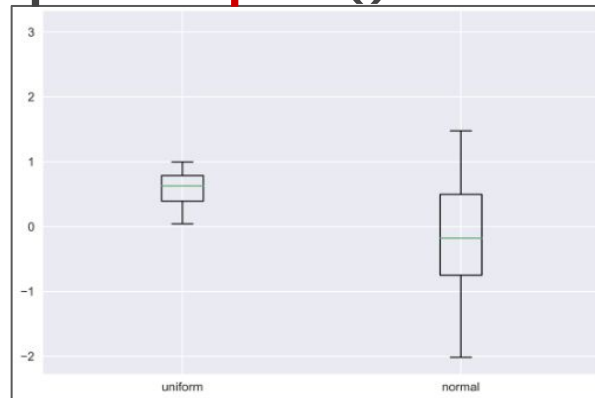
**plt.pie()**



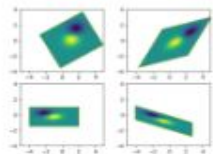
**plt.violinplot()**



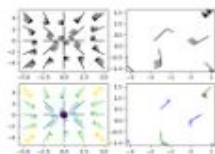
**plt.boxplot()**



# グラフの種類は無数にあるので公式ドキュメントなどを 適宜参照しながら使い分ける



Affine transform of an  
image



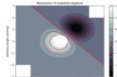
Wind Barbs



Barcode



Contour Label Demo



Contour Demo



Contour Hatching



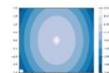
Hillshading



Anscombe's quartet



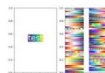
Hinton diagrams



Contourf and log color  
scale



Contouring the  
solution space of  
optimizations



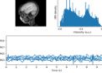
BboxImage Demo



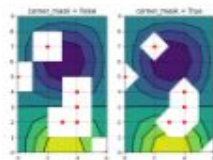
Left ventricle bullseye



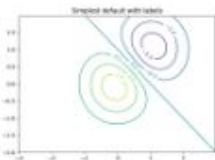
MRI



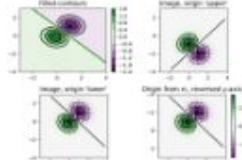
MRI With EEG



Contour Corner Mask



Contour Demo



Contour Image



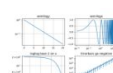
Nested pie charts



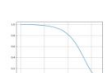
Labeling a pie and a  
donut



Bar chart on polar axis



Log Demo



Log Axis



Logit Demo



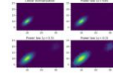
Polar plot



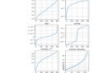
Polar Legend



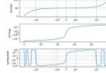
Scatter plot on polar



Exploring  
normalizations



Scales



Symlog Demo

# タイトルラベルと軸ラベルを追加する

①  
タイトルを追加できる  
**plt.title()**

②  
X軸とY軸のそれぞれに  
ラベルを追加できる  
**plt.xlabel()**  
**plt.ylabel()**

```
# 前のスライドで説明した部分  
x = [-1.5, 0.7, -0.4, -0.2, -0.3]  
y = [-1.3, -0.6, -0.2, 1.4, 0.4]  
plt.plot(x, y, 'o')
```

①  
# グラフタイトルを追加  
**plt.title('Title Name')**

②  
# Xの座標名を追加  
**plt.xlabel('axis\_X')**

# Yの座標名を追加  
**plt.ylabel('axis\_Y')**

**plt.show()**

タイトルラベル

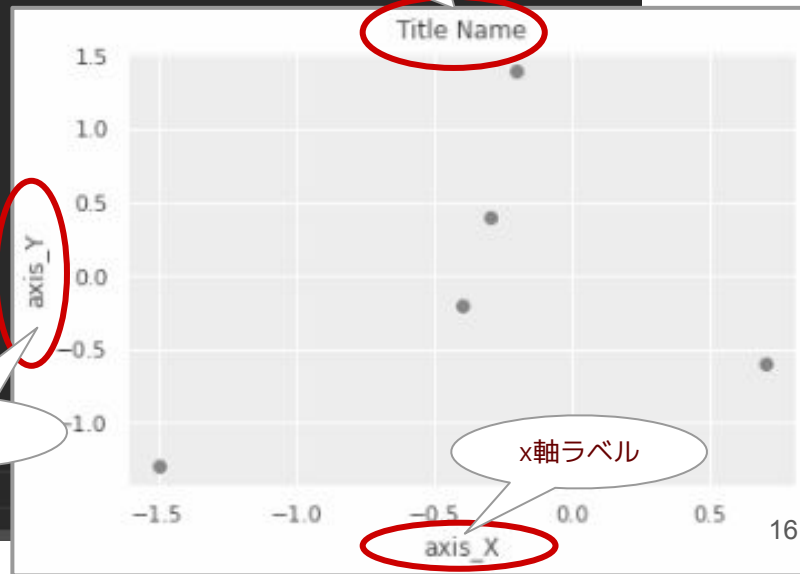
Title Name

y軸ラベル

axis\_Y

x軸ラベル

axis\_X



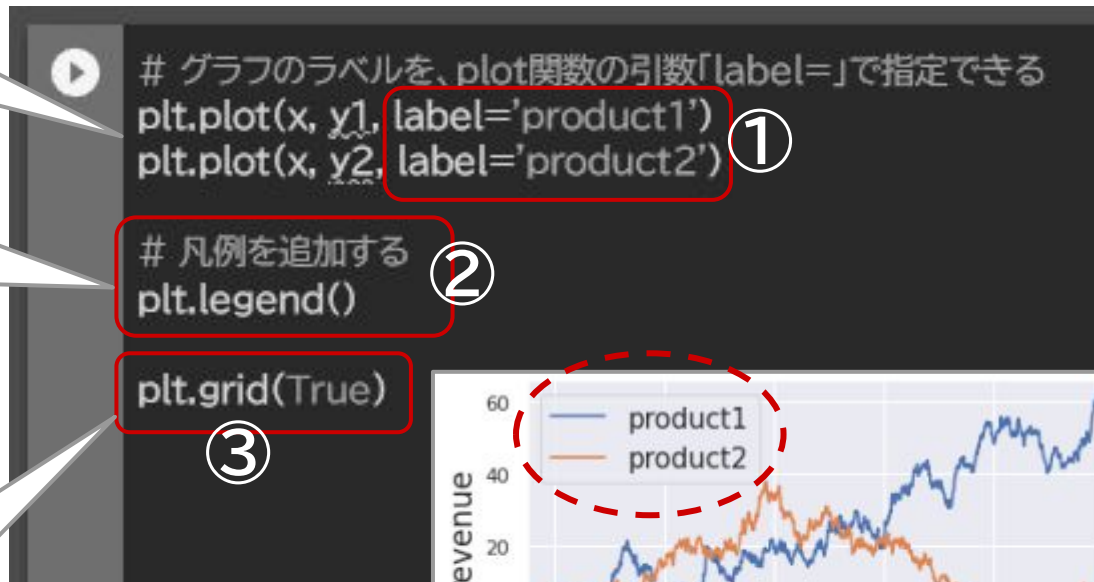


# 凡例とグリッド線を追加する

①plt.plotの引数「label=」で  
グラフラベルを指定した上で...

②凡例を追加できる  
**plt.legend()**

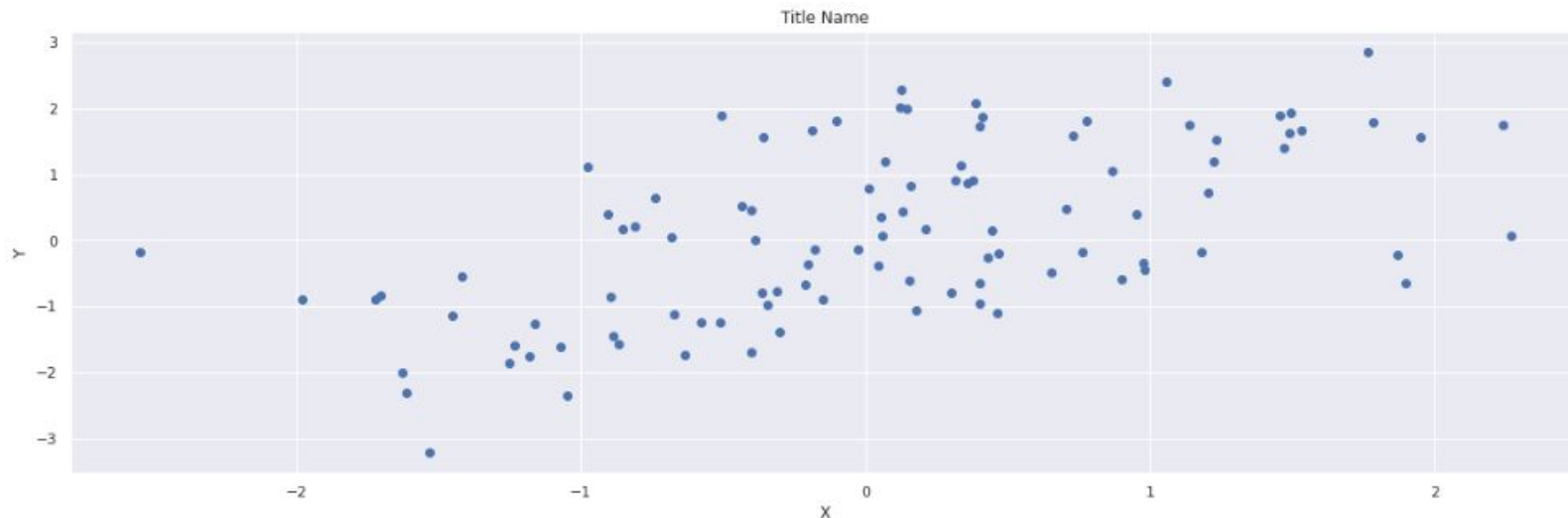
③グリッド線を追加できる  
**plt.grid()**  
引数はブール値(True or  
False)で渡す



# グラフの大きさを変更する

`plt.figure(figsize=({x軸の長さ}, {y軸の長さ}))`

(例) `plt.figure(figsize=(20, 6))`

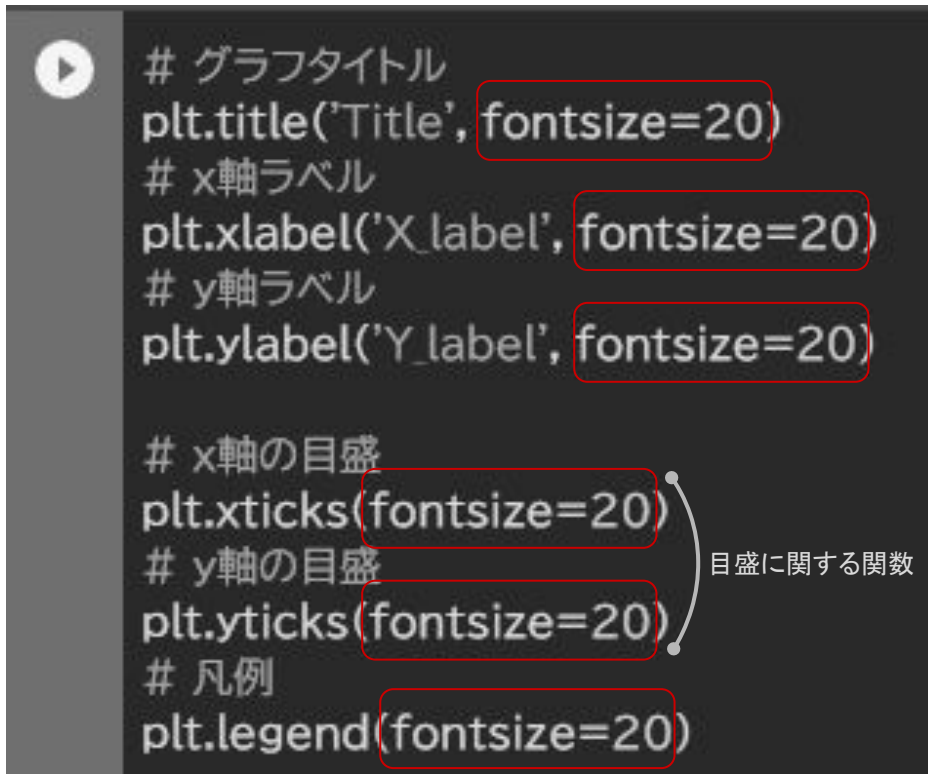


# 文字の大きさを変える（フォントサイズの変更）

各ラベルはいずれも引数に  
**fontsize**を指定することで文字  
の大きさを変更できる

フォントサイズだけではなく、  
色、書体、その他細かい調整も  
可能

細かい点は覚えるのではなく調べ  
方を身につける



```
# グラフタイトル
plt.title('Title', fontsize=20)
# x軸ラベル
plt.xlabel('X_label', fontsize=20)
# y軸ラベル
plt.ylabel('Y_label', fontsize=20)

# x軸の目盛
plt.xticks(fontsize=20)
# y軸の目盛
plt.yticks(fontsize=20)
# 凡例
plt.legend(fontsize=20)
```

目盛に関する関数

# 公式ドキュメントの見方

The image shows a screenshot of the Matplotlib 3.5.1 documentation website. The main navigation bar at the top includes links for Plot types, Examples, Tutorials, Reference, User guide, Develop, and Release notes. The 'Reference' link is circled in red and labeled with a red '1'. Below the navigation bar is a search bar and a sidebar with links for the latest stable release (3.5.1), the last release for Python 2 (2.2.5), and the development version. The main content area displays 'Matplotlib 3.5.1 documentation' and 'API Reference'. A list of modules is shown on the left, with 'matplotlib.pyplot' circled in red and labeled with a red '2'. Below it, 'matplotlib.pyplot' is circled in red and labeled with a red '3'. A large grey arrow points from the 'Reference' link to the 'API Reference' section. A text box on the right explains the navigation steps.

matplotlib

Plot types Examples Tutorials **Reference** User guide Develop Release notes

Search the docs ...

Matplotlib 3.5.1 documentation

Latest stable release  
3.5.1: docs | release notes

Last release for Python 2  
2.2.5: docs | changelog

Development version

matplotlib

Plot types Examples Tutorials Reference User guide

matplotlib.legend\_handler  
matplotlib.lines  
matplotlib.markers  
matplotlib.mathtext  
matplotlib.mlab  
matplotlib.offsetbox  
matplotlib.patches  
matplotlib.path  
matplotlib.patheffects  
**matplotlib.pyplot**  
**matplotlib.pyplot**  
matplotlib.pyplot.acorr

API Reference

公式ドキュメントトップページから  
「Reference」に入り、ページ左側にある見出しリストから使い方を調べたい関数を探し出す。

Example: We create a figure `fig` and `axes ax` then we  
axis labels and a figure title.

# 公式ドキュメントの見方

見出し。住所のように「matplotlibの中のpyplotの中のxlabelという関数について」という構成になっている

どんな引数を指定できるかが()の中に示されている

各引数の詳細説明領域

どんなデータ型で入力すべきかや、どのような選択肢が用意されているかなどが書かれている

その他様々な関数と共通のパラメータ。(フォントサイズのようなラベルの外観に関わるものはText propertiesを参照するよう記載されている。)

## matplotlib.pyplot.xlabel

```
matplotlib.pyplot.xlabel(xlabel, fontdict=None, labelpad=None, *,  
loc=None, **kwargs)
```

[source]

Set the label for the x-axis.

### Parameters:

**xlabel** : *str*

The label text.

**labelpad** : *float, default: rcParams["axes.labelpad"] (default: 4.0)*

Spacing in points from the Axes bounding box including ticks and tick labels. If None, the previous value is left as is.

**loc** : *{'left', 'center', 'right'}, default:*

*rcParams["xaxis.labellocation"] (default: 'center')*

The label position. This is a high-level alternative for passing parameters *x* and *horizontalalignment*.

**Other Parameters:** **\*\*kwargs** : *Text properties*

*Text properties* control the appearance of the label.

# Text properties

[https://matplotlib.org/stable/api/text\\_api.html#matplotlib.text.Text](https://matplotlib.org/stable/api/text_api.html#matplotlib.text.Text)

フォントを調整するための共通のパラメータが全て記載されている  
(膨大な項目数がある)

例えば前のスライドで示した引数の  
「fontsize」はここに説明を見つけることができる

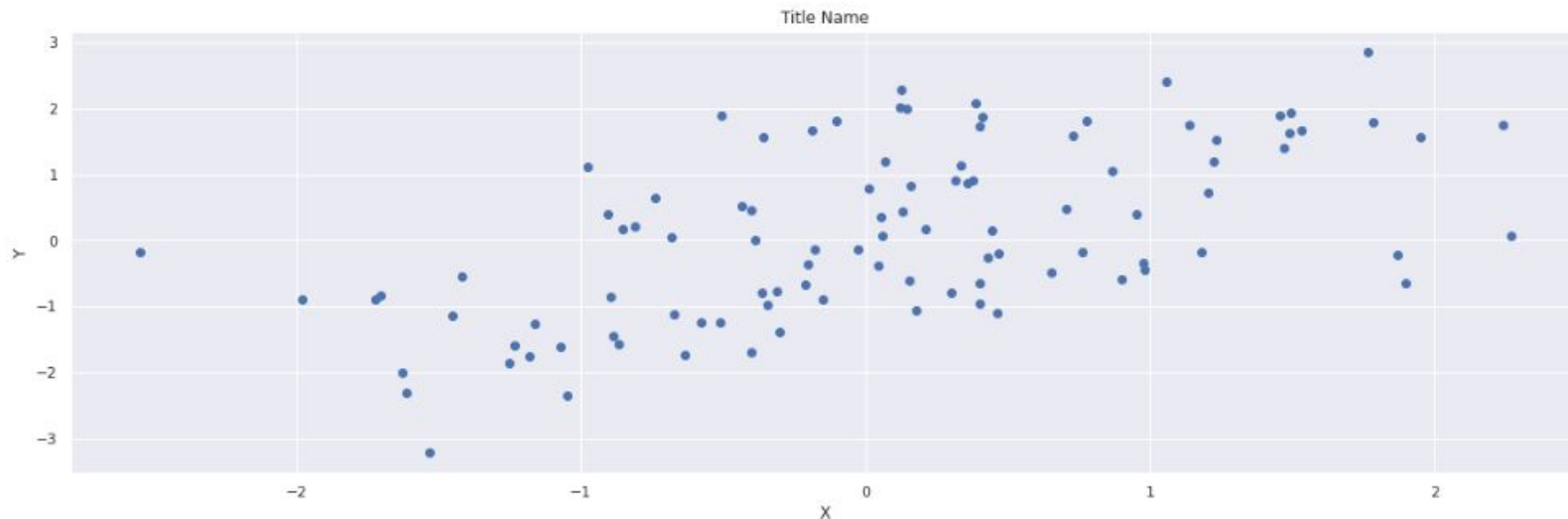
Property	Description
<code>agg_filter</code>	a filter function, which takes a (m, n, 3) float array and a dpi value, and returns a (m, n, 3) array
<code>alpha</code>	scalar or None
<code>animated</code>	bool
<code>backgroundcolor</code>	color
<code>bbox</code>	dict with properties for <code>patches.FancyBboxPatch</code>
<code>clip_box</code>	unknown
<code>clip_on</code>	unknown
<code>clip_path</code>	unknown
<code>color</code> or <code>c</code>	color
<code>figure</code>	Figure
<code>fontfamily</code> or <code>family</code>	{ <code>FONTNAME</code> , 'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}
<code>fontproperties</code> or <code>font</code> or <code>font_properties</code>	<code>font_manager.FontProperties</code> or <code>str</code> or <code>pathlib.Path</code>
<code>fontsize</code> or <code>size</code>	float or {'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'}

# グラフの大きさを変更する

再掲

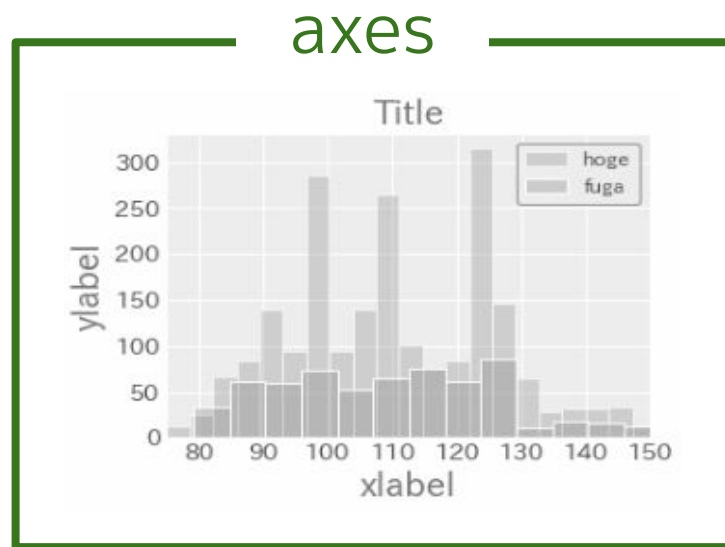
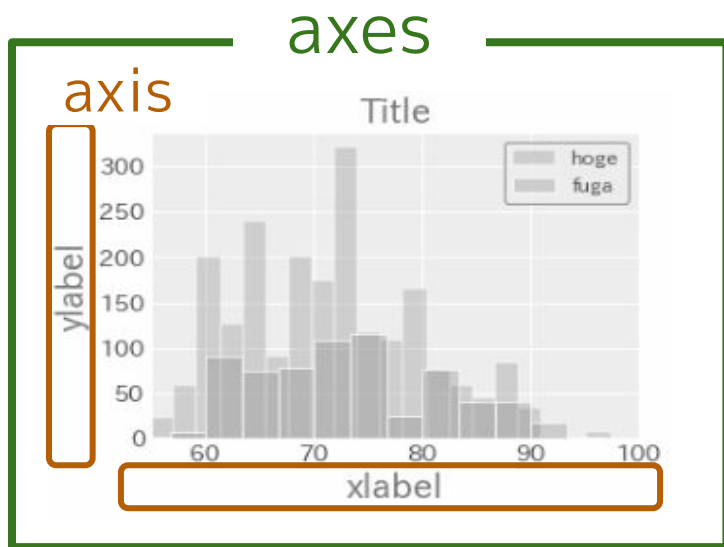
`plt.figure(figsize=({x軸の長さ}, {y軸の長さ}))`

(例) `plt.figure(figsize=(20, 6))`



# グラフを構成するfigureとaxesを理解する

figure





# 複数のaxesを持つグラフ

先ほどはグラフサイズの指定という文脈で紹介したが、この関数は「Figureの新規作成」が本来の機能（1つしかFigureを作らない場合は省略可）

Figureのなかに複数のAxesを作成する関数  
カッコの中に3つの引数が並んでいる。意味は次のとおり。

`plt.subplot( 2, 1, 2 )`

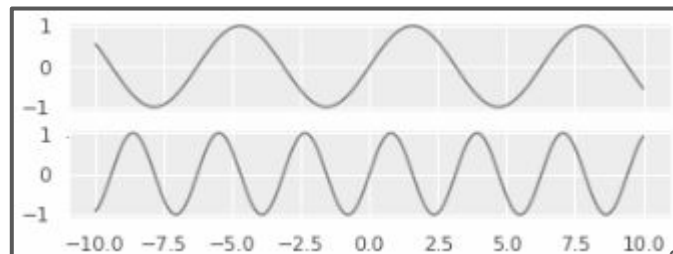
縦に2行、横に1列のAxesのうちの2つ目

```
# グラフの大きさを指定
plt.figure(figsize=(20, 6))

# 2行1列のグラフの1つ目
plt.subplot(2, 1, 1)
x = np.linspace(-10, 10, 100)
plt.plot(x, np.sin(x))

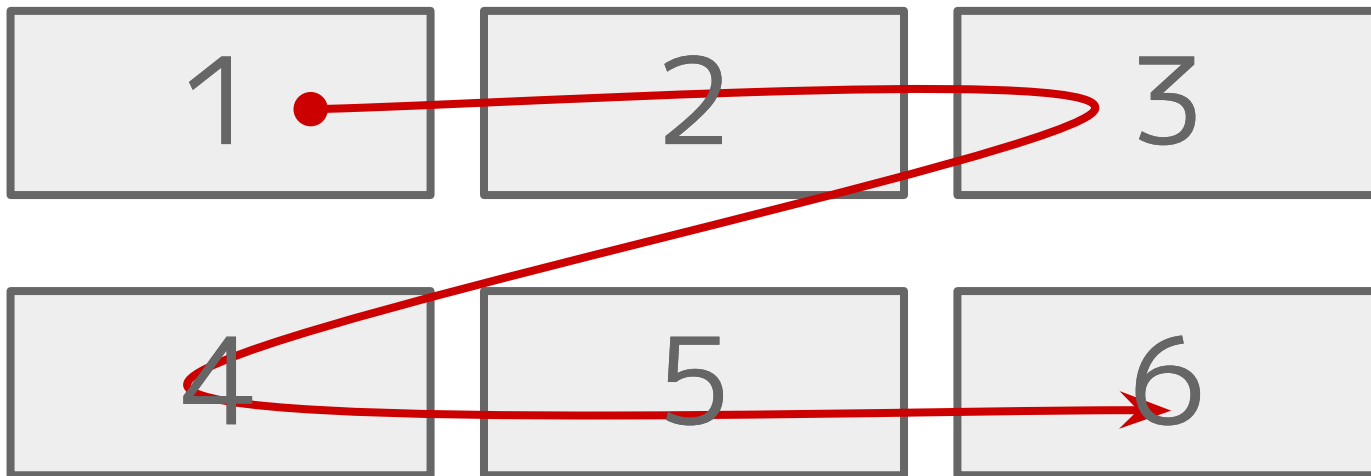
# 2行1列のグラフの2つ目
plt.subplot(2, 1, 2)
y = np.linspace(-10, 10, 1000)
plt.plot(y, np.sin(2*y))

plt.grid(True)
```



axesの番号と位置の並び方

`plt.subplot(2, 3, n)`



# MatplotlibとSeabornの関数の違い（データ指定の仕方）



【Matplotlibの書き方】(pyplotも同じ/一応Seabornもこの書き方でもOK)

```
plt.scatter(x = df['sepal_length'], y = df['petal_width'])
```

x軸のデータの指定

y軸のデータの指定

【Seabornの書き方】

```
sns.scatterplot(data=df, x='sepal_length', y='petal_width')
```

データセットの指定

x軸のカラム名の指定

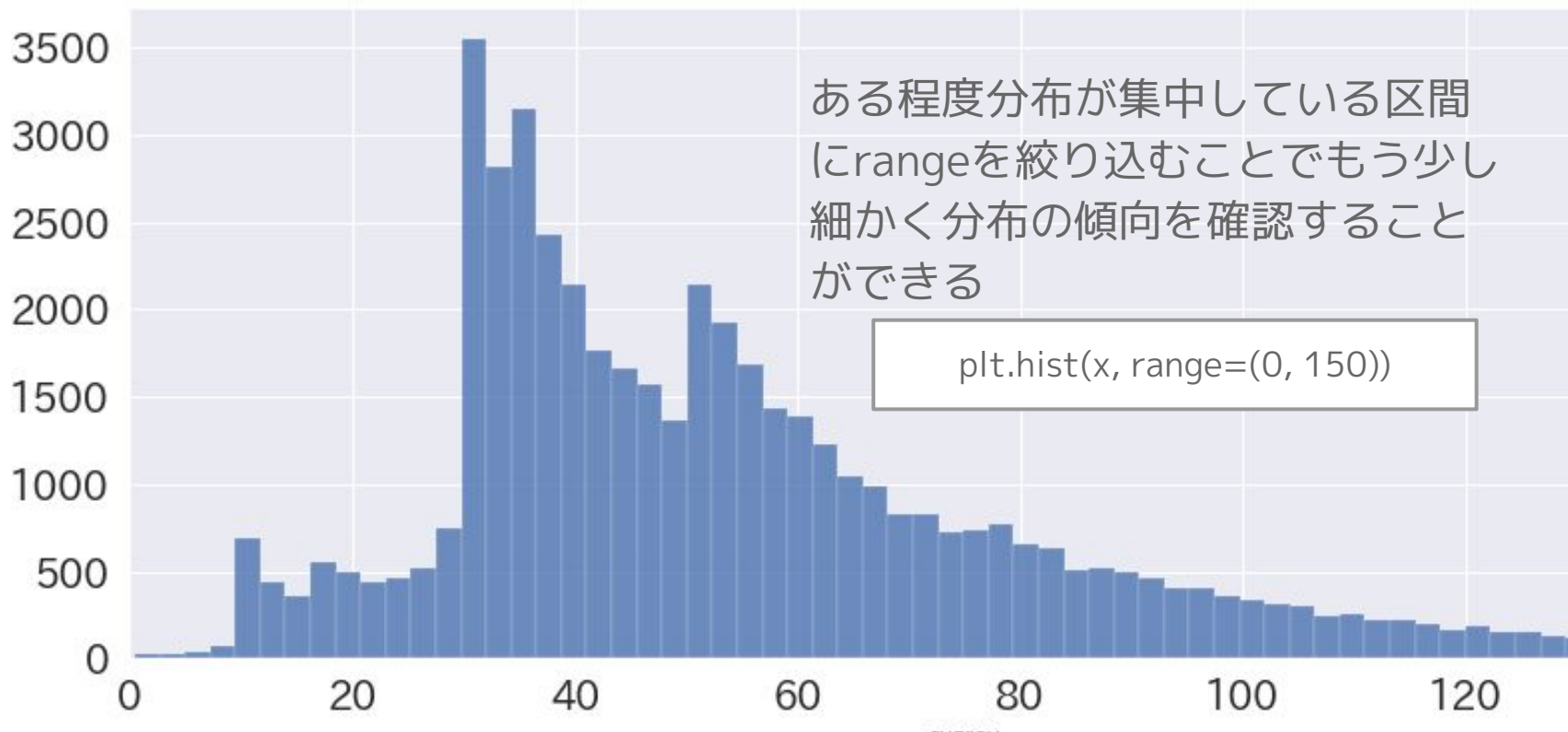
y軸のカラム名の指定

SeabornでもMatplotlibと同じ書き方が一応できるが、引数「data」にデータセットを指定したうえで、引数x・yにはカラム名だけを書くスタイルの方がSeabornではより一般的な書き方（こちらの方が応用が効きやすいため）。

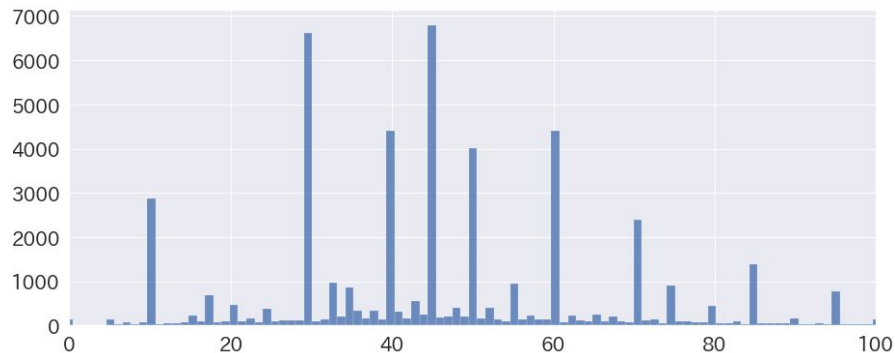
## 引数rangeの指定をしなかった例



## 引数rangeの指定をした場合

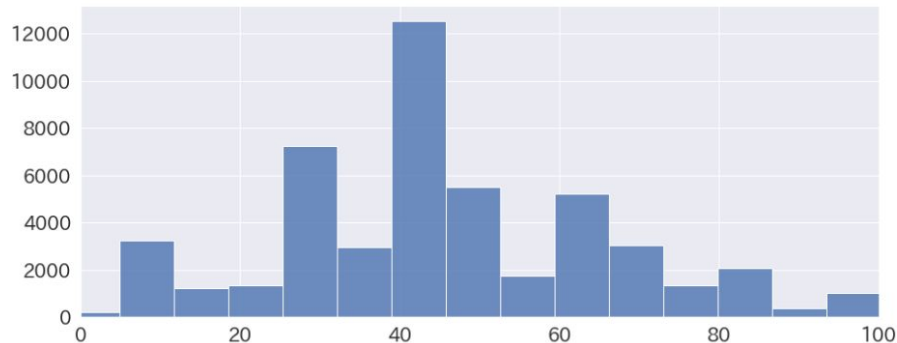


# ビン切りを変えてデータを観察する



ビンを細かく切った場合

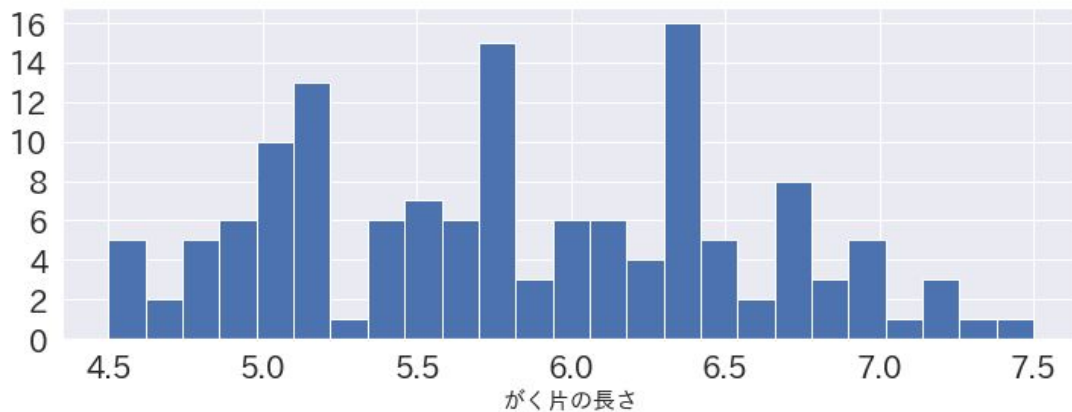
- 局所的に分布の集中している区間を発見できる場合がある
- 全体像としての傾向が掴みにくなる



ビンを大まかに切った場合

- 全体的な傾向を把握しやすい
- 局所的に生じている特異的な分布は見えなくなる

# ビン切りを変えてデータを観察する



引数「bins」に整数型の値を渡して、区間をいくつで切るか指定する。

`plt.hist(x, bins={ビン切りする数})`で指定する。

【例】

`plt.hist(x, bins=25)`

グラフごとに引数もたくさんあるので調べながら使う

## matplotlib.pyplot.hist

```
matplotlib.pyplot.hist(x, bins=None, range=None, density=False,  
weights=None, cumulative=False, bottom=None, histtype='bar', align='mid',  
orientation='vertical', rwidth=None, log=False, color=None, label=None,  
stacked=False, *, data=None, **kwargs)
```

全て覚える必要はどこにもなく、必要に応じて調べて使う



# 棒グラフをつくる関数 plt.bar()

plt.bar()で棒グラフを指定

引数のalignは棒グラフをX軸のどこに合わせるかを指定するもの

'center': バーの中心をxtickに合わせる

'edge': バーの左端をxtickに合わせる

※バーの右端をxtickに合わせる場合は'edge'を選択した上で、widthに負の数を指定する。

plt.xticks(), plt.yticks()

x,yの各軸の目盛りのパラメータを指定

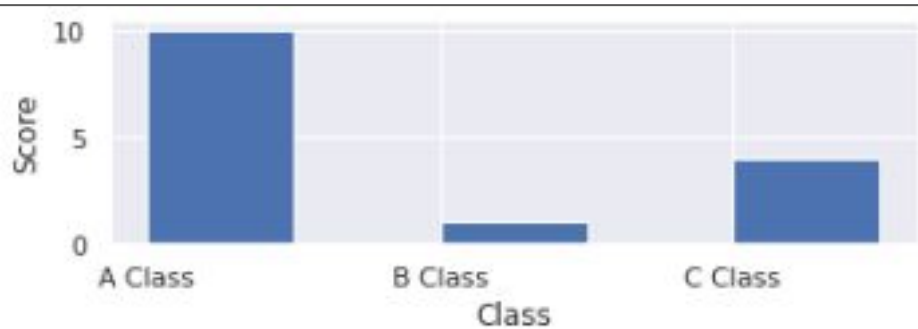
第一引数に目盛りを入れる位置の値のリスト、

第二引数に目盛りラベルをリストで指定

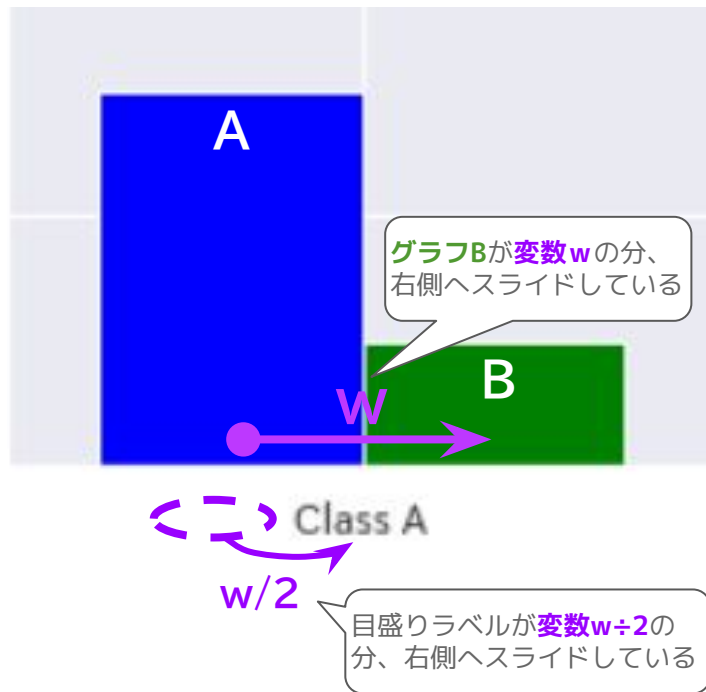
```
# 表示するデータ
x = [1, 2, 3]
y = [10, 1, 4]

plt.bar(x, y, align='edge', width = 0.5)

# 棒グラフそれぞれのラベル
plt.xticks(x, ['A Class', 'B Class', 'C Class'])
```



# 棒グラフの応用：比較棒グラフ



$w = 0.4$

①「棒グラフの太さ」に後で指定したい値を先に変数 $w$ に入れておく（コードをシンプルにするため）

②グラフAは通常通りに指定する

```
plt.bar(x, y1, width = w)
```

x軸方向の位置 y軸方向の位置

③グラフBは「x軸方向の位置」として「グラフAの太さ」である変数 $w$ を足した値を指定する

```
plt.bar(x + w, y2, width = w)
```

x軸方向の位置 y軸方向の位置 棒の太さ

④x軸の目盛りラベルの「x軸方向の位置」として「グラフAの太さ」である変数 $w$ の半分を足した値を指定する

```
plt.xticks(x + w / 2, ['Class A', 'Class B',
```

x軸方向の位置

# 棒グラフの応用：積み上げ棒グラフ



```
p1 = plt.bar( x, height1,
```

x軸方向の位置

棒グラフの高さ

p1の「棒グラフの高さ」の値をそのまま  
p2の「y軸方向の高さ」に指定している。

```
p2 = plt.bar( x, height2, bottom = height1,
```

x軸方向の位置

棒グラフの高さ

y軸方向の位置

引数「**bottom**」

棒グラフのy軸方向の高さを指定する引数。  
渡した値の分、お尻が持ち上がるイメージ。

# 円グラフの作成：plt.pie()

## 【使い方】

表示をしたい要素の順に値を入力したリストをそれぞれの引数に渡して使う。

## 【主な引数】

**x**

各要素の値

**label**

各要素に表示する文字列

**colors**

各要素の色

**explode**

各要素を中心からどの程度離して表示するか

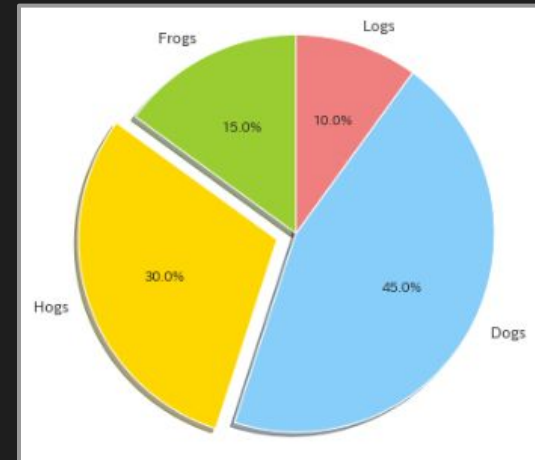
**startangle**

要素を並べる起点の位置を角度で指定する（「0」に指定すると3時の位置が起点になる）

	[0]	[1]	[2]	[3]
labels =	'Frogs',	'Hogs',	'Dogs',	'Logs'
sizes =	15,	30,	45,	10
colors =	'yellowgreen',	'gold',	'lightskyblue',	'lightcoral'
explode =	0,	0.1,	0,	0

# グラフを表示

```
plt.pie(x = sizes,  
        explode = explode,  
        labels = labels,  
        colors = colors,  
        autopct = '%1.1f%%',  
        shadow = True,  
        startangle = 90)
```



# 円グラフは使い方に注意が必要

Google

円グラフ メリット デメリット

× | 🔍

🔍 すべて 🖼️ 画像 📰 ニュース 🛍️ ショッピング 🎬 動画 ⋮ もっと見る ツール

約 3,860,000 件 (0.52 秒)

<https://matsuda-blog.info> > データ分析・統計・数学 ▾  
**円グラフを使ってはいけない3つの理由。代わりに棒グラフを ...**  
2021/11/14 — もちろん円グラフにも強みはありますが、デメリットの方が大きいのが事実です。実際、科学の世界では円グラフはほとんど用いられず、棒(帯)グラフを ...

<https://dataviz.hatenablog.com> > entry > 2019/02/05 ▾  
**それでもまだ円グラフを使いますか？ - データ可視化の ...**  
2019/02/05 — 今日のテーマは円グラフ (パイ チャート) です。みんな大好き円グラフ。レポートや白書、報告書、そういったものには必ず 1 つ以上円グラフが入ってい ...  
22/05/07 にこのページにアクセスしました。

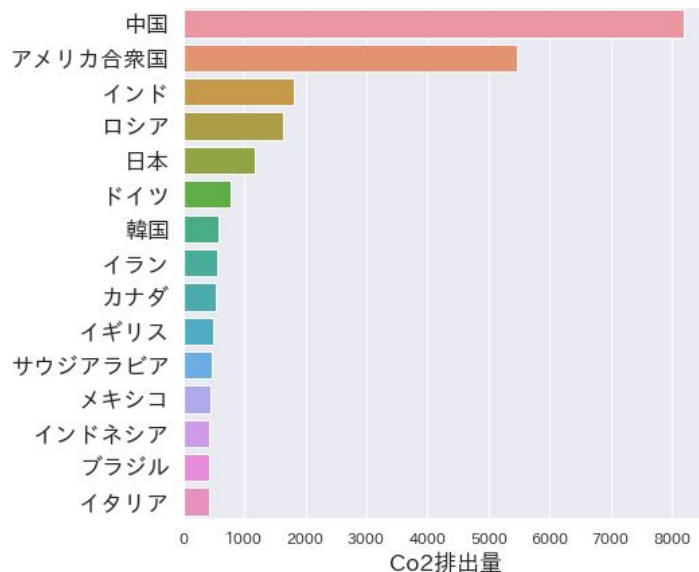
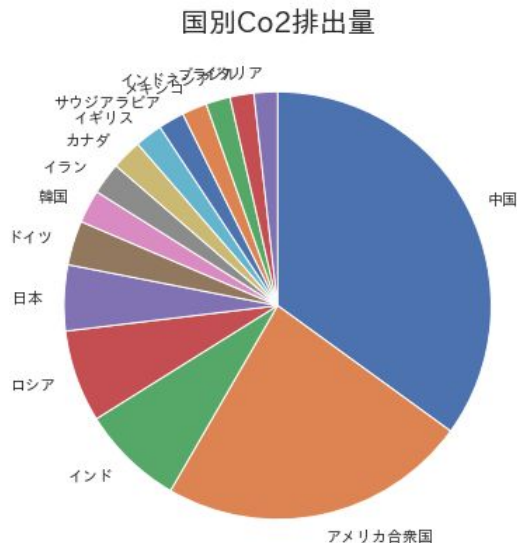
<https://jikitourai.net> > 生活・コラム ▾  
**円グラフを安易に使ってはいけない理由を実例を交えて説明し ...**  
2021/04/08 — 見た目がカッコいいからと3D円グラフを使っていないでしょうか？円グラフは作成が簡単なのでつい使いがちなグラフですが、見る人の認識を誤らせやすい ...  
サンプルで見るダメな円グラフ・3D円グラフの問題点・平面の円グラフではどうか  
22/05/07 にこのページにアクセスしました。

<https://sigma-eye.com> > エクセル ▾  
**実は使えない円グラフ 2つの理由を紹介します【代替案も ...**  
2020/01/07 — データの解析をする上で、グラフというものは必要不可欠です。棒グラフ、線グラフ、レーダーチャート・・・様々なも。

- 円グラフを使うことそれ自体に対して賛否両論があることだけは頭の片隅に置いておく

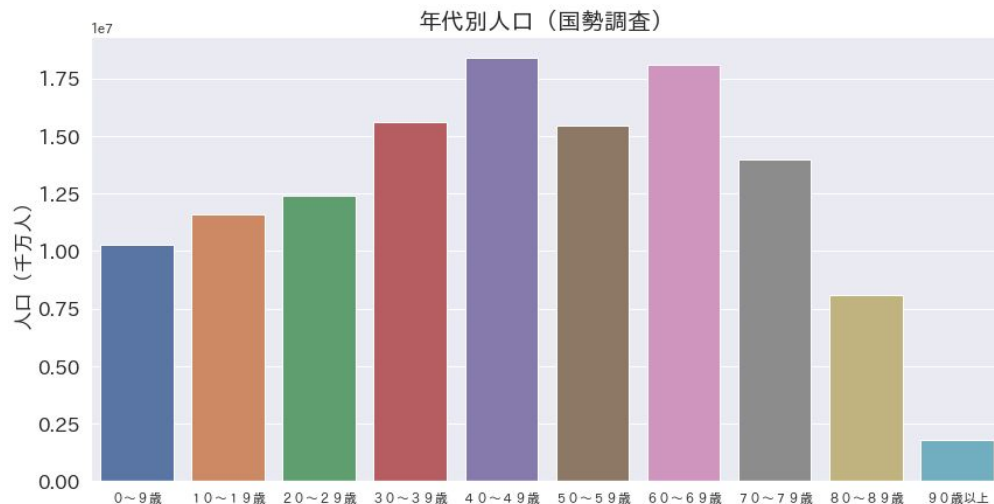
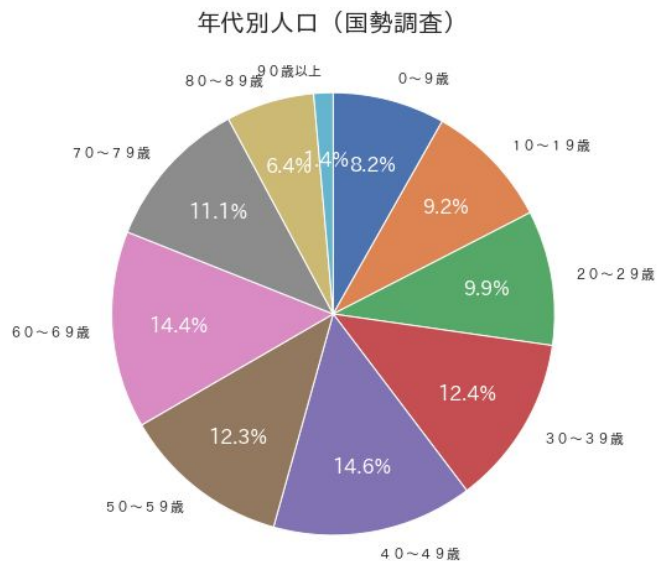
※「使うべきでない」という指摘は、例えば  
松本健太郎『グラフをつくる前に読む本』、技術評論社、2017など

# 円グラフは使い方に注意が必要



- 支配的なシェアを持つ要素が存在することを目立たせる効果はある
- でも棒グラフでも同じことは把握できる（プレゼンならともかくEDAなら棒グラフの方が簡単に作れて良い）
- 円グラフはちょっと要素数が多いだけでラベルが潰れるなどして調整に無駄な手間がかかる

# 円グラフは使い方に注意が必要



- 円グラフは支配的なシェアの要素がない場合は漫然とした印象になりやすい。
- 棒グラフに比べて円グラフでは要素間の差が直感的に把握しにくい。

# EDA自動化ライブラリの例: *Sweetviz*

<https://github.com/fbdesignpro/sweetviz>

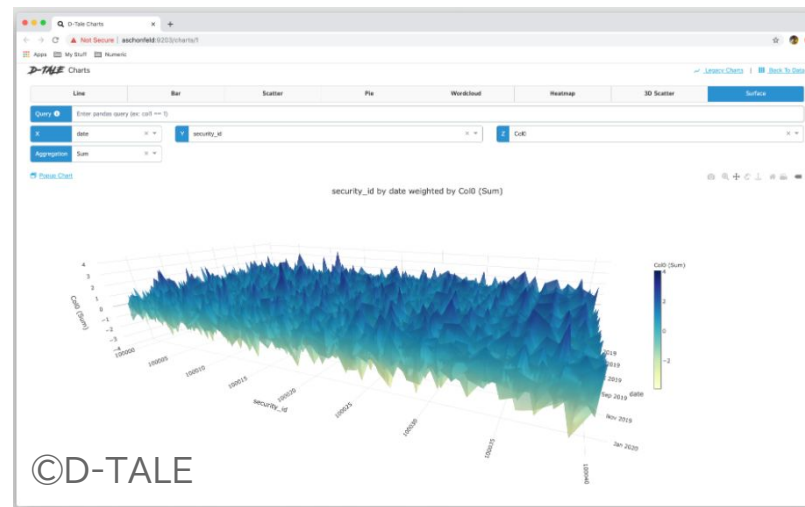
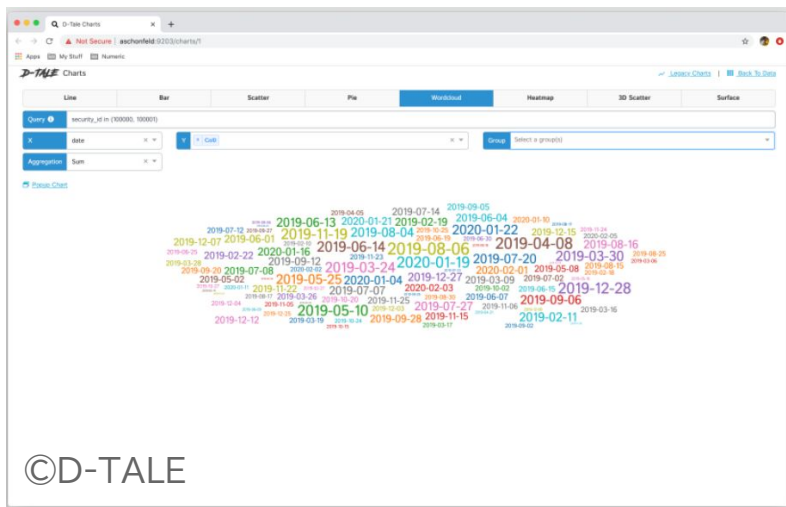




# EDA自動化ライブラリの例 : D-TALE

<https://github.com/man-group/dtale>

	date	security_id	int_val	Col0	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col10	Col11
0	2019-09-13	100000	1370356066	1.05	1.68	-0.09	0.68	2.04	1.40	-0.74	-0.15	-0.40	-0.07	0.16	0.90
1	2019-09-13	100001	3273834703	0.81	-0.48	2.75	-0.38	1.05	0.71	-0.01	0.40	0.63	0.24	-1.62	1.99
2	2019-09-13	100002	1430102282	0.27	1.15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	2019-09-13	100003	3553690088	0.34	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	2019-09-13	100004	101747017	0.66	-1.30	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
5	2019-09-13	100005	3273834703	0.81	-0.48	2.75	-0.38	1.05	0.71	-0.01	0.40	0.63	0.24	-1.62	1.99
6	2019-09-13	100006	344383406	0.08	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	2019-09-13	100007	3419927953	0.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
8	2019-09-13	100008	72974836	-0.03	0.26	0.55	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
9	2019-09-13	100009	245480467	0.78	0.29	0.68	-0.85	0.11	0.57	-0.12	-0.29	-0.21	0.40	0.07	0.41
10	2019-09-13	100010	5572650089	-0.24	-0.04	0.29	1.63	-2.35	0.19	0.32	-1.19	0.96	-1.02	0.43	1.40
11	2019-09-13	100011	7367070080	-0.41	0.34	-1.21	0.17	-0.14	-0.35	-1.54	-0.74	-0.81	-1.05	0.57	-0.20
12	2019-09-13	100012	3540402206	0.26	0.93	0.09	-0.11	0.77	-0.41	0.25	-1.92	1.05	-1.74	-0.38	-0.36

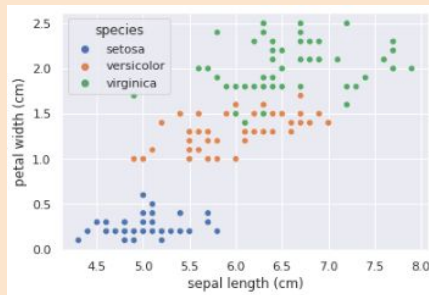


# 量的データと質的データ

## 量的データ

四則演算に意味がある数値データ

(金額・時間・年齢etc.)



連続値も離散値も含む

## 質的データ

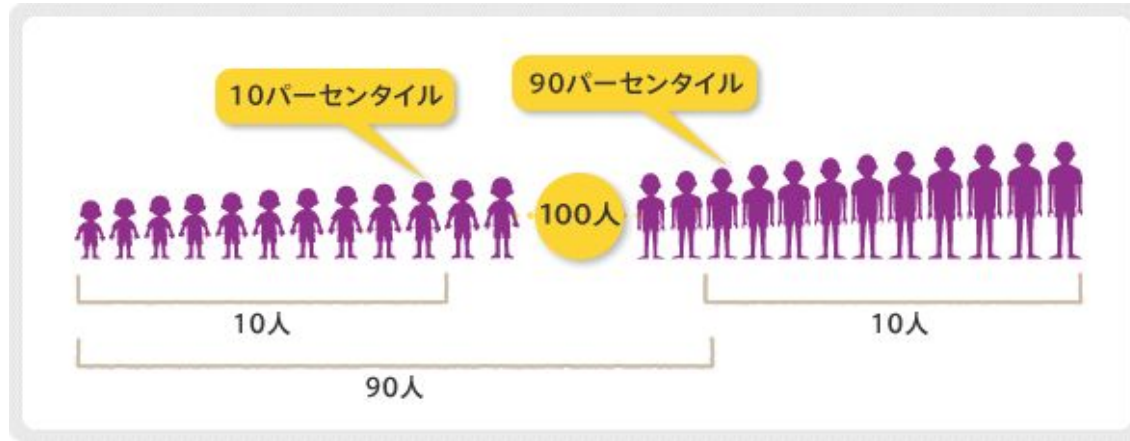
カテゴリや性質を表現するための  
データで四則演算に意味がない

(性別・状態区分・ある事柄への該当有無etc.)

address	famsize	Pstatus	Medu	Fedu
U	GT3	A	4	4
U	GT3	T	I	I
U	LE3	T	I	I

「Medu(母親の学歴区分)」などは数値型なので四則演算できないこともないが、分析上の意味はもたない

# パーセンタイルと要約統計量の把握



全体を100として小さい方から数えて  $n\%$  目にくる値

```
[ ] # 要約統計量
student_data_math['absences'].describe()
```

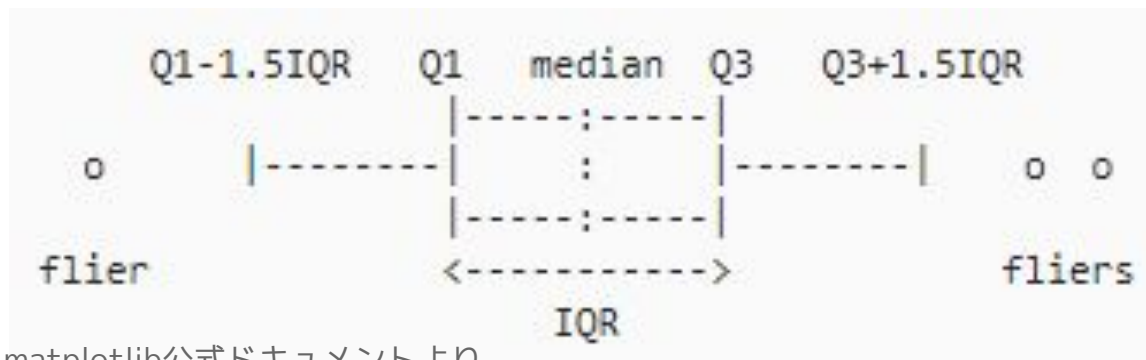
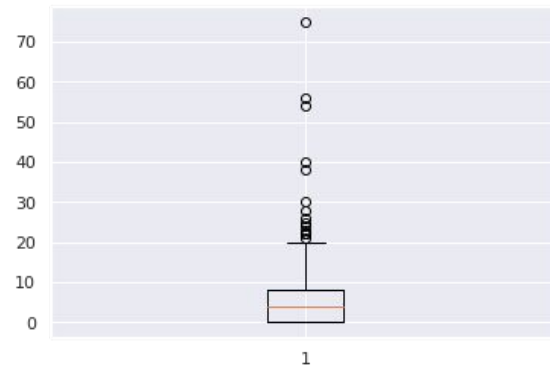
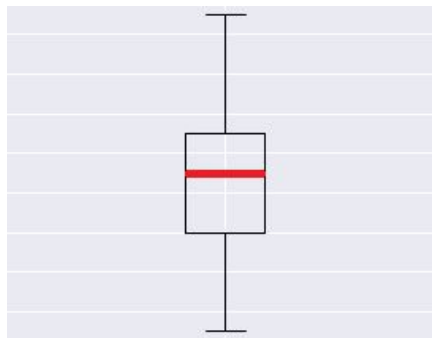
count	395.000000
mean	5.708861
std	8.003096
min	0.000000
25%	0.000000
50%	4.000000
75%	8.000000
max	75.000000
Name: absences, dtype: float64	

Pandasの`describe()`メソッドで他の統計量と併せて一括算出できる

参照URL : [http://ghw.pfizer.co.jp/comedical/evaluation/images/img\\_relation\\_01.gif](http://ghw.pfizer.co.jp/comedical/evaluation/images/img_relation_01.gif)

# 四分位数と箱ひげ図

箱の下底	第1四分位 25パーセンタイル
箱内部の赤い線	第2四分位 50パーセンタイル 中央値
箱の上底	第3四分位 75パーセンタイル
ひげの端	四分位範囲(IQR) の1.5倍
フライヤーポイント	四分位範囲(IQR) の1.5倍を超えた 外れ値

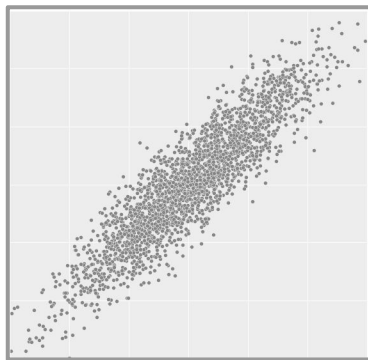


matplotlib公式ドキュメントより

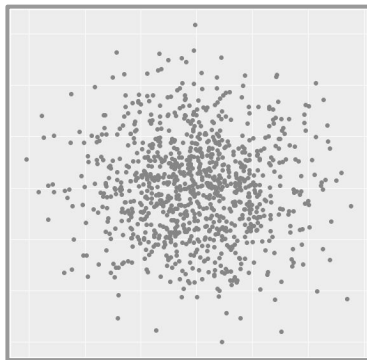
[https://matplotlib.org/stable/api/as\\_gen/matplotlib.pyplot.boxplot.html](https://matplotlib.org/stable/api/as_gen/matplotlib.pyplot.boxplot.html)

# 相関係数の性質

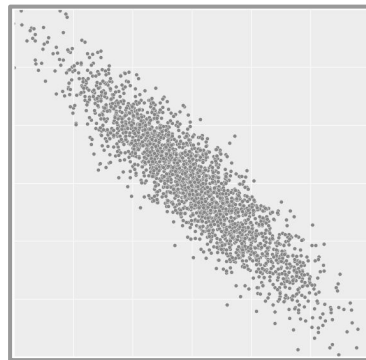
- $-1 \sim 1$ までの値を取る
- 正の相関が強いと  $1$ に近づく
- 負の相関が強いと  $-1$ に近づく
- $0$ に近いほど相関が弱い
- 一方の値が変化すると他方の値も変化するような連動性を表現する係数
- 因果関係があるかないかは定義外



正の相関

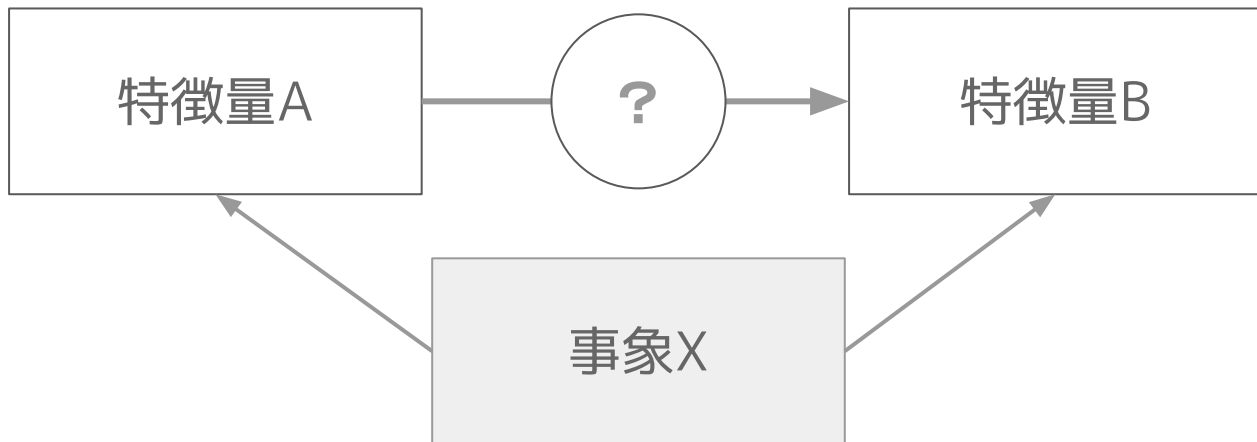


相関がない



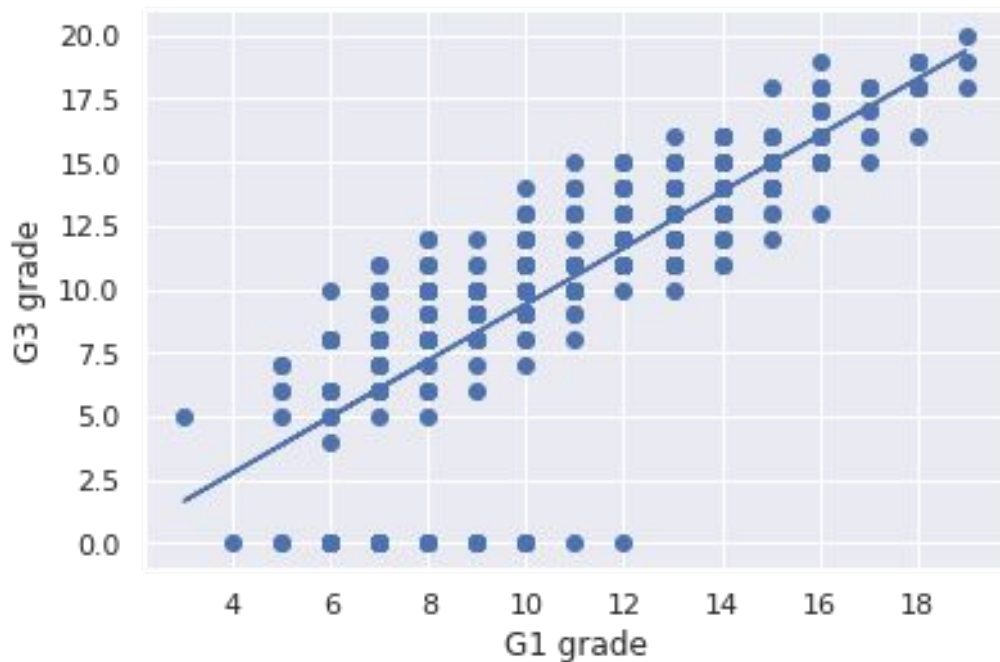
負の相関

# 相関関係は因果関係とは異なる



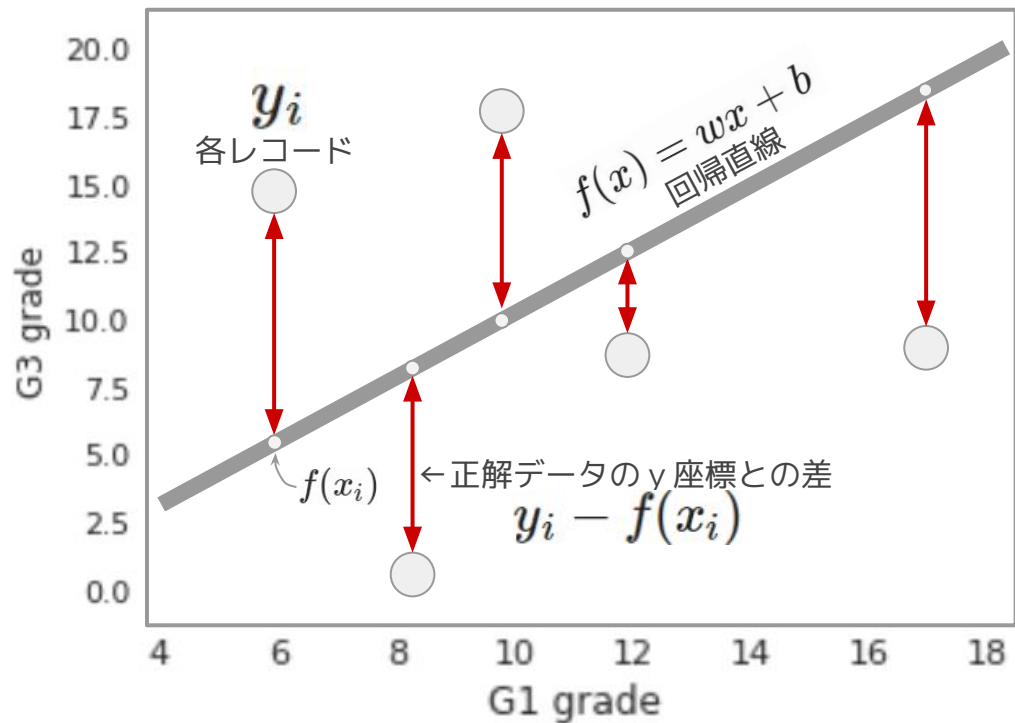
- 特徴量AとBの間に強い相関があっても、別の要因が双方に作用している可能性もある
- 偶然である場合もある
- 因果関係は係数で手軽に求められるものではなく、ドメイン知識も使いながらの丁寧な考察が必要

# 最小二乗法による単回帰分析



$$\overset{\text{目的変数}}{y} = \overset{\text{説明変数}}{w} \overset{\text{係数}}{x} + \overset{\text{切片}}{b}$$

# 最小二乗法による単回帰分析



$$f(x) = wx + b$$

誤差が最小になる  
係数aと切片bを  
求める (計算は自動)

$$\sum_{i=1}^n (y_i - f(x_i))^2$$

二乗誤差





# Scikit-learnによる単回帰分析

①まず、Scikit-learnパッケージから、使いたいモデルが属するモジュールをインポートする。（コード上では「sklearn」という名前なので注意）

②Scikit-learnには線形モデル以外にも様々な種類もモデルが用意されており、これらは次週詳しく学ぶ。

```
[ ] from sklearn import linear_model  
  
# 線形回帰のインスタンスを生成  
reg = linear_model.LinearRegression()
```



# Scikit-learnによる単回帰分析

```
[ ] from sklearn import linear_model  
  
# 線形回帰のインスタンスを生成  
reg = linear_model.LinearRegression()
```

この部分はモジュール名。この配下に複数の関数を用意されており、ドット以降に使いたい関数名を書く。

※以下のようにモジュールではなく使いたい関数を直接importして使っても良い。

```
from sklearn.linear_model import LinearRegression  
reg = LinearRegression()
```

①機械学習モデルを作成する関数(コンストラクタ)

②コンストラクタにより左辺の変数(ここでは「reg」)に機械学習モデルのインスタンスが格納される。

③後続する各種処理はこのインスタンスに対して行う。

# インスタンスはメソッドやインスタンス変数で活用する



## Step1

モデル別のコンストラクタ関数で機械学習モデルを作成

```
# 線形回帰のインスタンスを生成  
reg = linear_model.LinearRegression()
```



## Step2

**fit**メソッドで**学習**  
説明変数と目的変数を引数に渡す（ここではXとY）

```
# 予測モデルを計算、ここでa,bを算出  
reg.fit(X, Y)
```

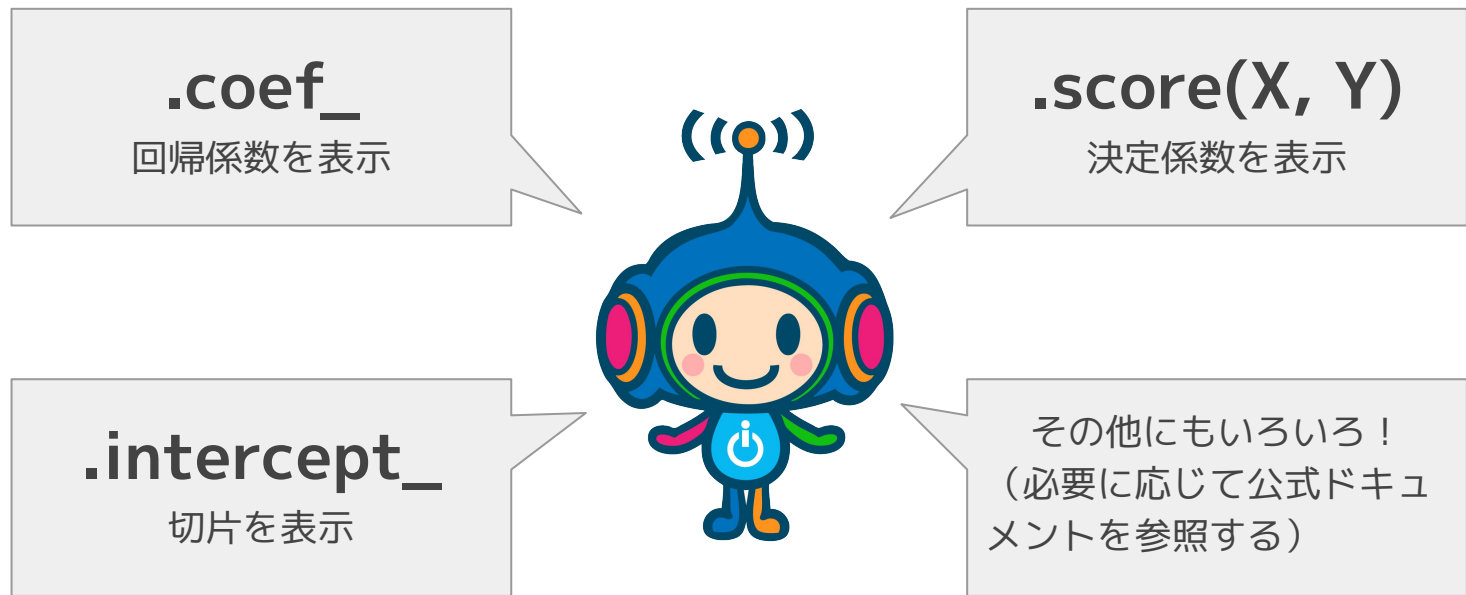


## Step3

**predict**メソッドで**予測**  
（予測させたいデータを引数に渡す）

```
# 予測したいデータを読み込ませて予測させる  
reg.predict(test_data)
```

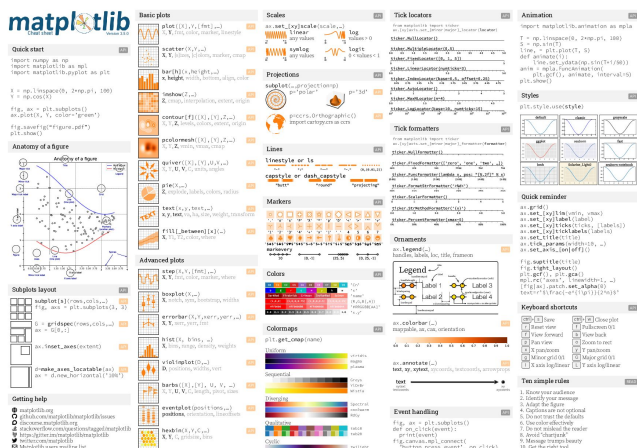
# 単回帰モデルの主なメソッド & インスタンス変数



Scikit-learnは各ライブラリの中でも特に公式ドキュメントが見やすい。  
→積極的に公式の情報を参照するのがおすすめ。

# 番外編：チャートシートを活用しよう

## Cheatsheets



## Handouts

### Matplotlib for beginners

Matplotlib is a library for making 2D plots in Python. It is designed with the philosophy that you should be able to create simple plots with just a few commands.

**1 Initialize**

```
import numpy as np
import matplotlib.pyplot as plt
```

**2 Prepare**

```
X = np.linspace(0, 4*np.pi, 1000)
Y = np.sin(X)
```

**3 Render**

```
fig, ax = plt.subplots()
ax.plot(X, Y)
plt.show()
```

**4 Observe**

**Choose**

Matplotlib offers several kind of plots (see Gallery):

```
X = np.random.uniform(0, 1, 100)
Y = np.random.uniform(0, 1, 100)
ax.scatter(X, Y)
```

```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linestyle="--")
```

```
X = np.linspace(0, 10, 100)
Y = np.random.uniform(0, 1, 100)
ax.bar(X, Y)
```

```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, marker="o")
```

**Organize**

You can plot several data on the same figure, but you can also split a figure in several subplots (named Axes):

```
X = np.linspace(0, 10, 100)
Y1, Y2 = np.sin(X), np.cos(X)
ax.plot(X, Y1, X, Y2)
```

```
ax1, ax2 = plt.subplots(2, 1)
ax1.plot(X, Y1, color="C1")
ax2.plot(X, Y2, color="C2")
```

```
ax1, ax2 = plt.subplots(1, 2)
ax1.plot(Y1, X, color="C1")
ax2.plot(Y2, X, color="C2")
```

**Label (everything)**

```
ax.plot(X, Y)
fig.suptitle("A Sine wave")
ax.set_title("A Sine wave")
ax.plot(X, Y)
ax.set_xlabel("None")
ax.set_ylabel("Time")
```

**Explore**

Figures are shown with a graphical user interface that allows to zoom and pan the figure, to navigate between the different views and to show the value under the mouse.

**Save (bitmap or vector format)**

```
fig.savefig("my-first-figure.png", dpi=300)
fig.savefig("my-first-figure.pdf")
```

Matplotlib 3.5.0 Handbook for beginners. Copyright (c) 2021 Matplotlib Development Team. Released under a CC BY 4.0 International License. Supported by NumFOCUS.

→ <https://matplotlib.org/cheatsheets/>

多数の関数やメソッドのうち、重要なものを見やすくまとめたシート  
たくさん出回っているので手元に置いておくとも便利