



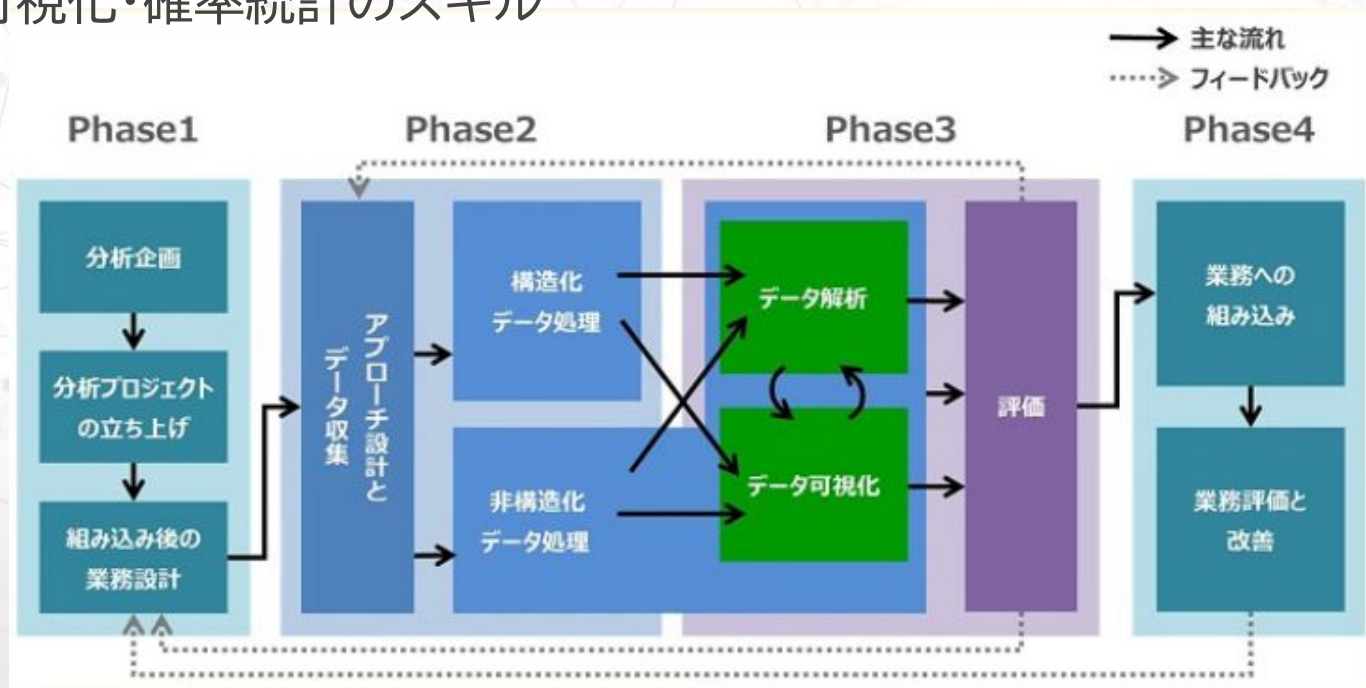
2022/5/17

スライド作成者: 中内

# GCI2022SUMMER WEEK4

Pythonによるデータ可視化  
記述統計・単回帰分析

# 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・確率統計のスキル



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

# 探索的データ分析(EDA)

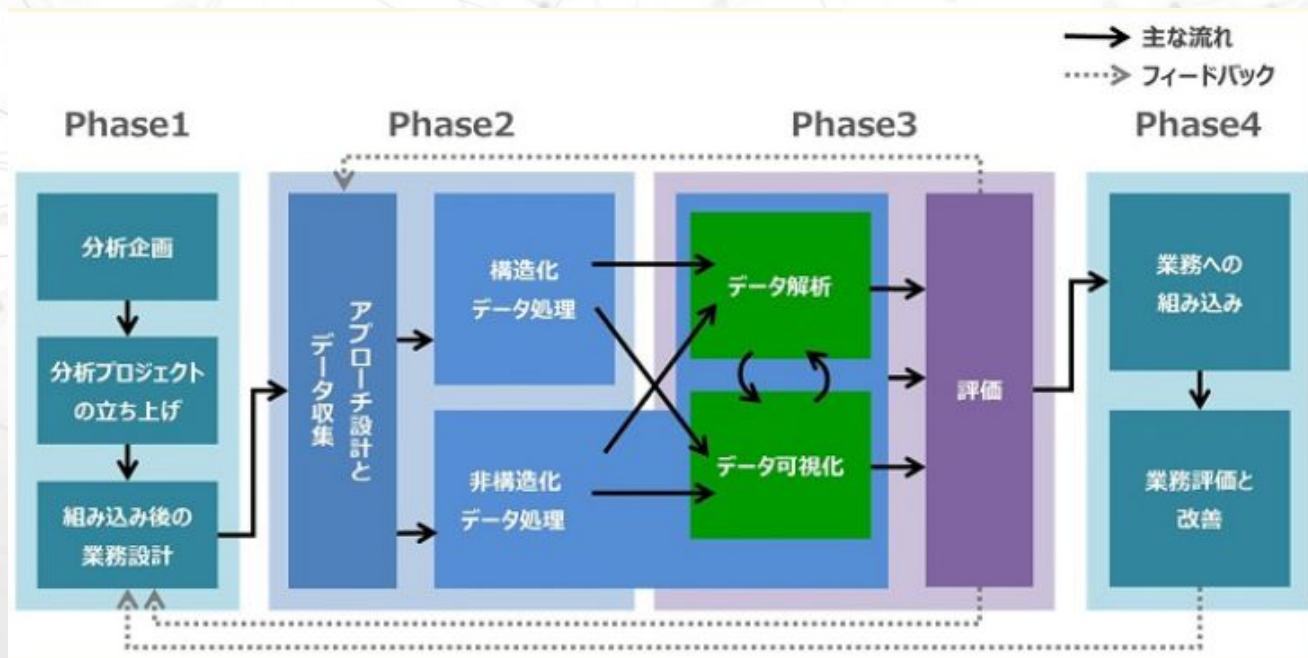
- データセットに適宜前処理を施しつつ様々な統計量を抽出して可視化し、そこに内在する特性・パターン・偏りについて探索的に仮説立案・検証を繰り返して分析すること
- 「探索的」であることが重要であり、必然的に「試行錯誤」を重ねることになる
- ①理論やドメイン知識などの事前知識を活用した仮説立案と、  
②バイアスを排した特徴観察の両方が重要(知的好奇心も大事)

# 探索的データ分析(EDA)

- データサイエンスの文脈では、データ可視化と確率・統計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」という文脈もあり、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

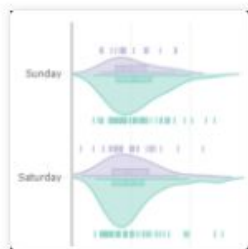


- 可視化することで、クライアント企業に提起すべき課題を自分が発見できる
- 可視化することで、クライアント企業に課題を伝えることができる
- 各フェーズでそれぞれの可視化が必要

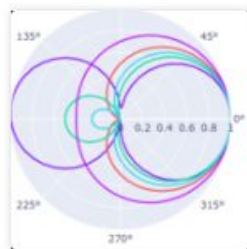


<https://www.ipa.go.jp/jinzai/itss/itssplus.html>

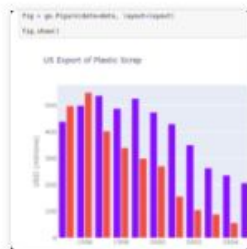
# データ可視化



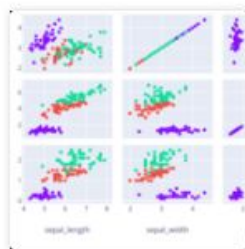
The Figure Data Structure



Creating and Updating Figures



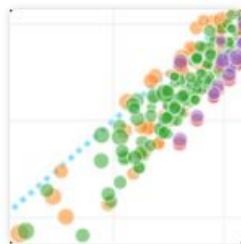
Displaying Figures



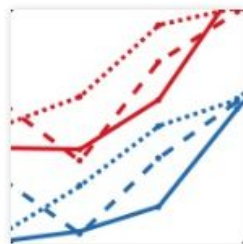
Plotly Express



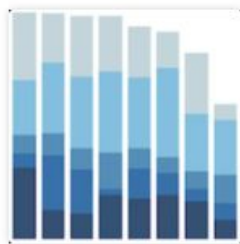
Analytical Apps with Dash



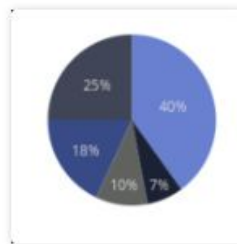
Scatter Plots



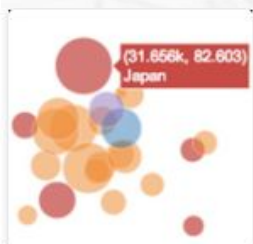
Line Charts



Bar Charts



Pie Charts



Bubble Charts

<https://plotly.com/python/>

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ースがある

## matplotlib(本体API)

### 【概要】

matplotlibの本体API

### 【特徴】

**オブジェクト指向**に従うインターフェースとなっている

### 【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】

最低限の利用である場合にも、pyplotよりも多くのパラメータを明示する必要がある

## matplotlib.pyplot

### 【概要】

**簡易版**的な位置づけ(MATLABに近似させたインターフェースという側面もある)

### 【特徴】

簡素で対話的なグラフ化に特化している

### 【長所】

**細かいパラメータの指定なしに短いコーディング**で使える

### 【短所】

図の細かい作り込みには一部制約あり  
(完全にはオブジェクト指向でないため)



# 最も手軽には数行でグラフ化できる(matplotlib.pyplot)

前提: 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
```

①描画したいデータを用意

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

③グラフ表示を指示  
(JupyterNotebookでは事前設定により省略可)

ミニマムには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



# 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]

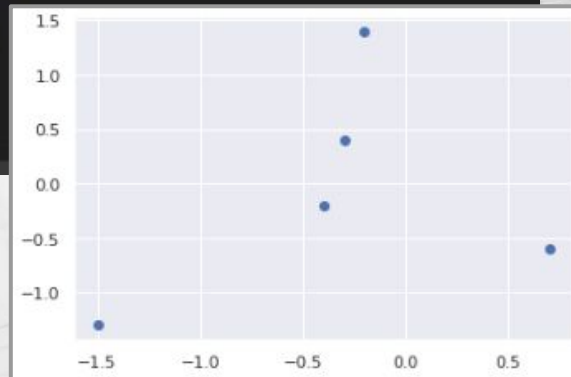
y = [-1.3, -0.6, -0.2, 1.4, 0.4]

# グラフを描画

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

# グラフを表示

plt.show()



# 最も手軽には数行でグラフ化できる(matplotlib.pyplot)

前提: 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]
```

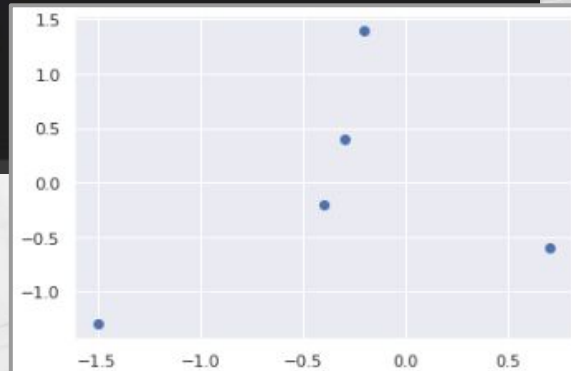
```
y = [-1.3, -0.6, -0.2, 1.4, 0.4]
```

# グラフを描画

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

# グラフを表示

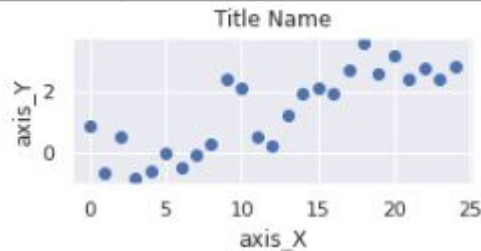
```
plt.show()
```



# 使用頻度の高いパラメータ(マーカーの変更)

3つ目の引数に指定する記号  
(`'-'`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

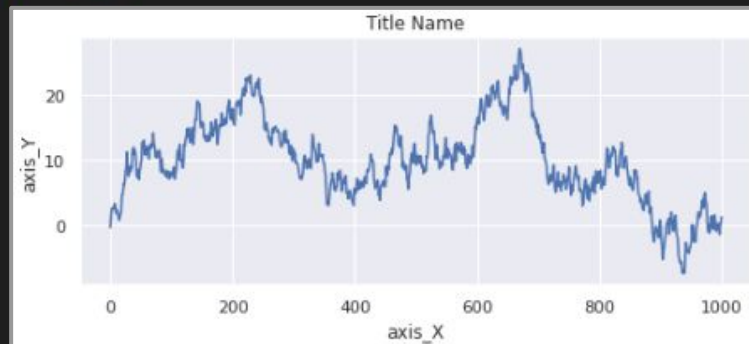


```
# 仮にこんなデータがx, yに入っていたとする
x = np.arange(1000)
y = np.random.randn(1000).cumsum()
```

```
# 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
7 (CARETDOWN)		caredown

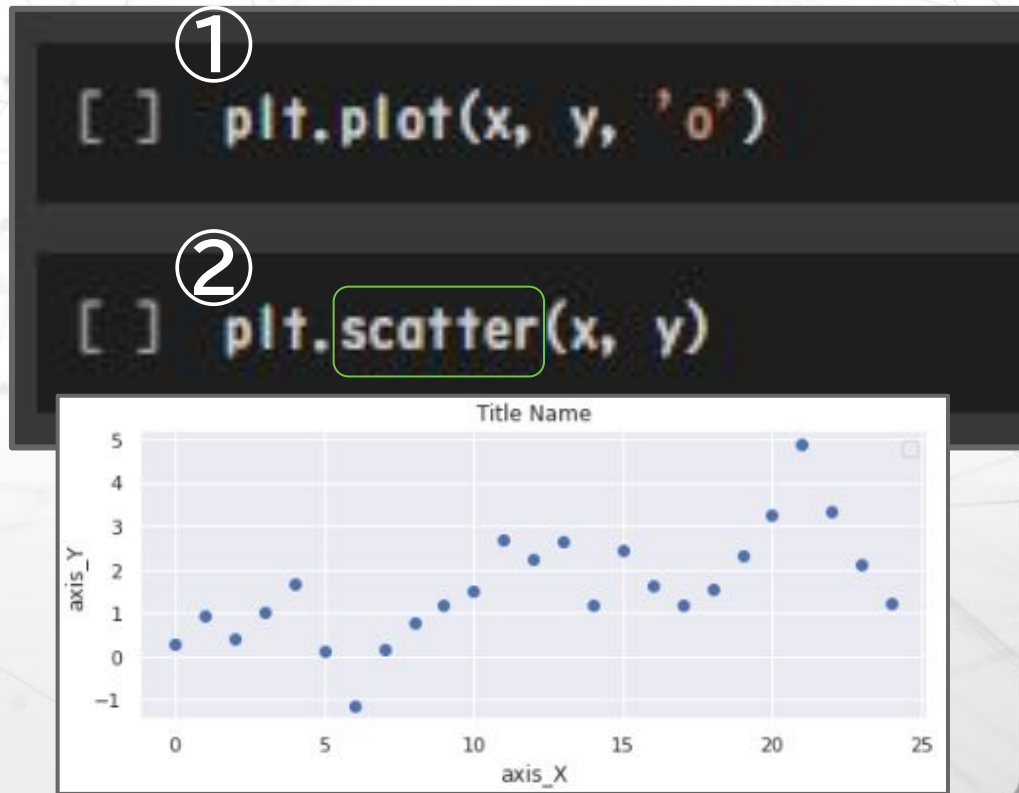
8 (CARETLEFTBASE)		caretleft (centered at base)
9 (CARETRIGHTBASE)		caretright (centered at base)
10 (CARETUPBASE)		caretup (centered at base)
11 (CARETDOWNBASE)		caredown (centered at base)
"None", " " or ""		nothing
'\$...\$'		Render the string using mattext. E.g "\$f\$" for marker showing the letter 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)

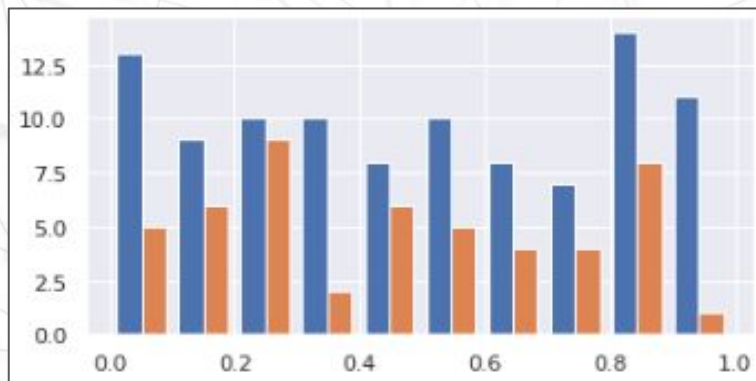


# 使用頻度の高いパラメータ(グラフ種類ごとの関数)

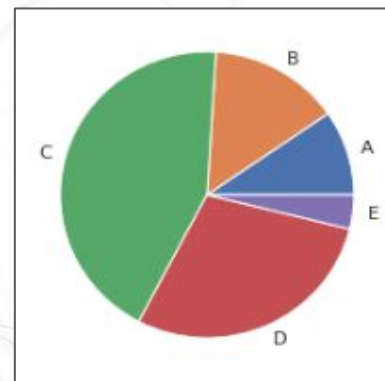
- 前のスライドでは右図①の方法でグラフの種類を指定した
- 更にさまざまなグラフ種類を試したい場合は、②の方法で指定することもできる
- ②の方法の場合、  
「**scatter**」の部分を経々なグラフの名前に置き換えることで多様なグラフを使い分けられる



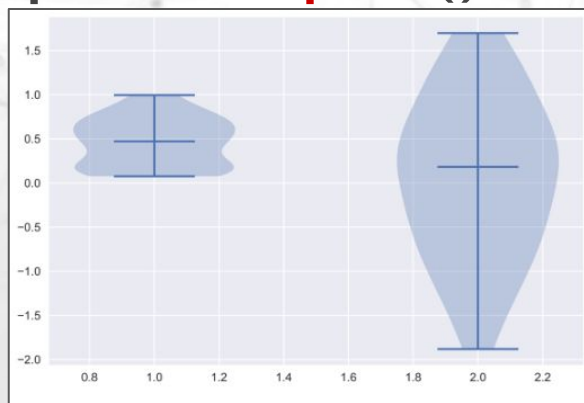
**plt.hist()**



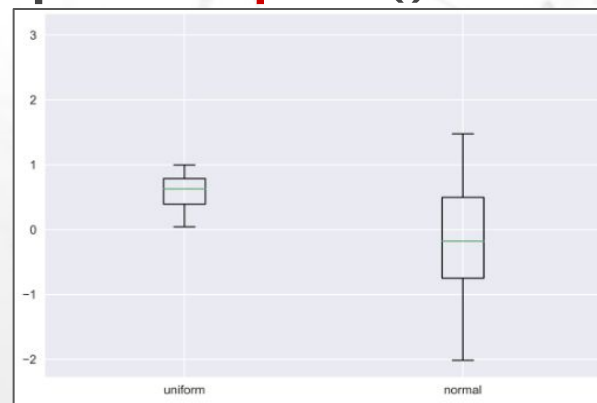
**plt.pie()**



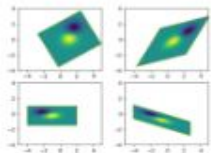
**plt.violinplot()**



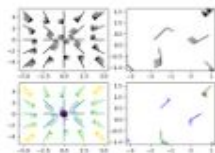
**plt.boxplot()**



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



Affine transform of an  
image



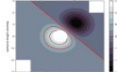
Wind Barbs



Barcode



Contour Label Demo



Contour Demo



Contourf Hatching



Hillshading



Anscombe's quartet



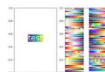
Hinton diagrams



Contour and log color  
scale



Contouring the  
solution space of  
optimizations



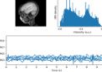
BboxImage Demo



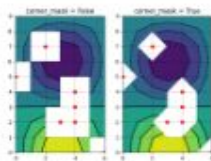
Left ventricle bullseye



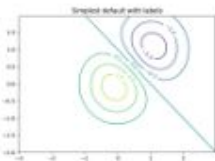
MRI



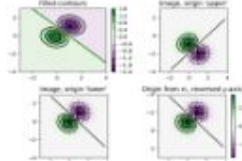
MRI With EEG



Contour Corner Mask



Contour Demo



Contour Image



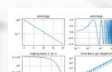
Nested pie charts



Labeling a pie and a  
donut



Bar chart on polar axis



Log Demo



Log Axis



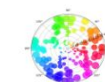
Logit Demo



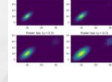
Polar plot



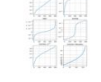
Polar Legend



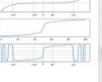
Scatter plot on polar



Exploring  
normalizations



Scales



Symlog Demo

# 使用頻度の高いパラメータ(タイトルラベルと軸ラベル)

①  
グラフに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()**



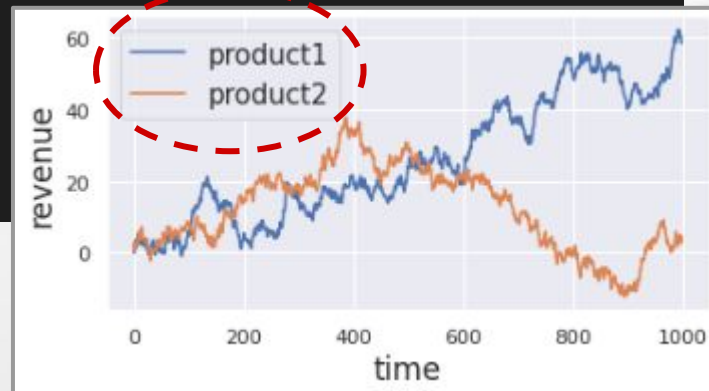
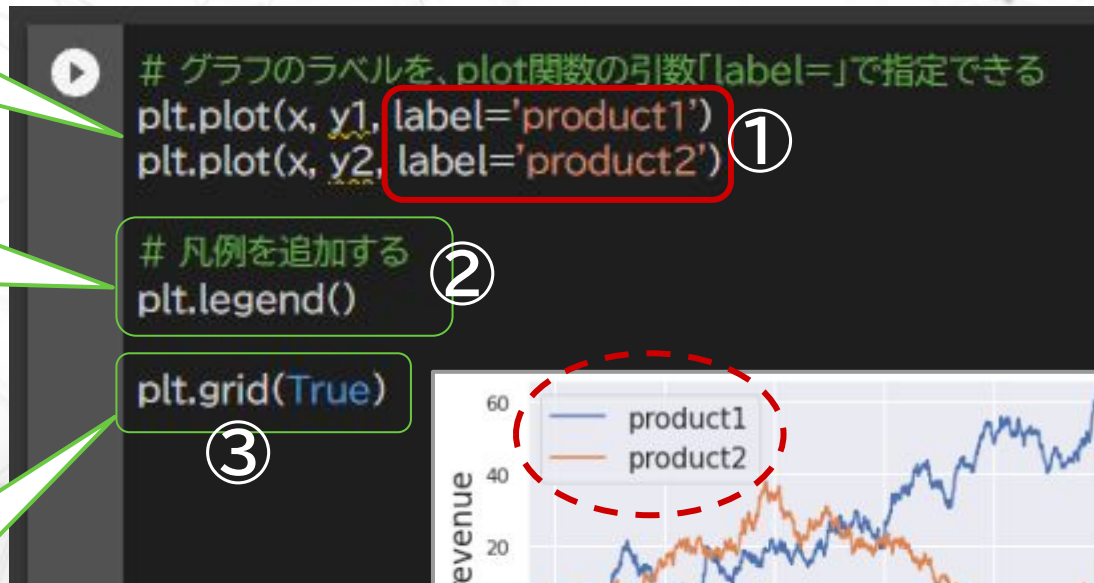


# 使用頻度の高いパラメータ(凡例とグリッド線)

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()**

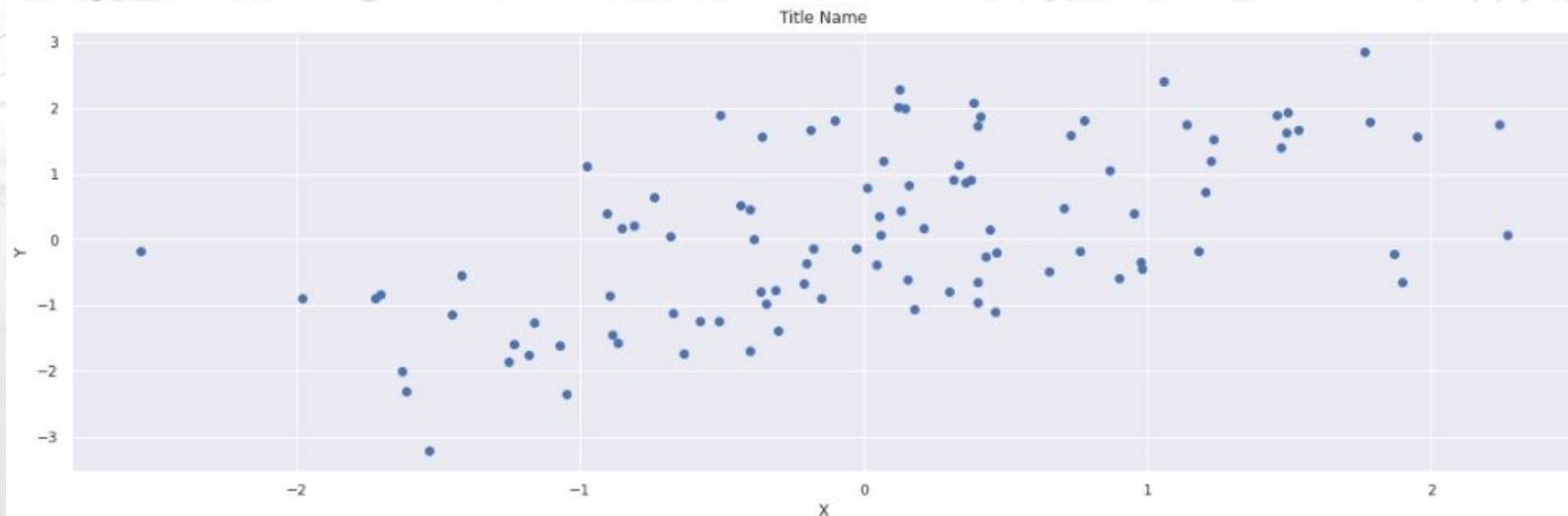
③グリッド線を追加できる  
**plt.grid()**  
引数はブール値(True or  
False)で渡す



# 使用頻度の高いパラメータ(グラフサイズの変更)

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

(例) `plt.figure(figsize=(20,6))`



# 使用頻度の高いパラメータ(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

もちろん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

気になったことは公式ドキュメント  
やチートシートで調べる。



```
# 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 two-step process for navigating the Matplotlib documentation. The top screenshot shows the main documentation page with the 'Reference' link in the top navigation bar circled in red and labeled with a red circle containing the number '1'. The bottom screenshot shows the 'API Reference' page, with the left sidebar containing a list of modules. In this sidebar, 'matplotlib.pyplot' is circled in red and labeled with a red circle containing the number '2', and 'matplotlib.pyplot' is also circled in red and labeled with a red circle containing the number '3'. A large grey arrow on the right side of the image points from the top screenshot to the bottom screenshot, indicating the transition between the two views.

matplotlib

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

1

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.path\_effects  
**matplotlib.pyplot**  
**matplotlib.pyplot**  
matplotlib.pyplot.colorbar

API Reference

公式ドキュメントトップページで「**Reference**」を選択し、遷移した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。



# 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というメソッドについて」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

その他様々なメソッド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**を参照するように記載されている。)

## 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)

テキストラベル系の関数を制御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「`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>	<b>Figure</b>
<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'}

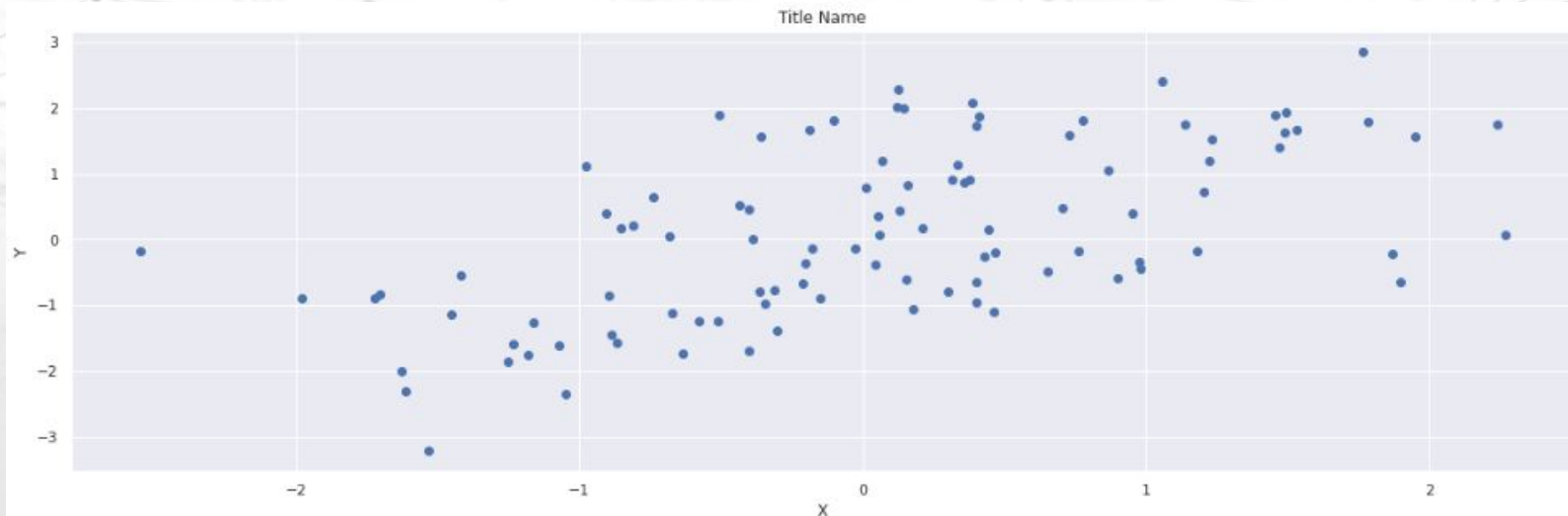
The background of the slide is a complex network diagram. It consists of numerous small, light-gray circular nodes scattered across the frame. These nodes are interconnected by a dense web of thin, light-gray lines, creating a mesh-like structure that resembles a molecular model or a data network. The overall aesthetic is technical and modern.

# Notebook^

# 使用頻度の高いパラメータ(グラフサイズの変更)

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

(例) `plt.figure(figsize=(20,6))`

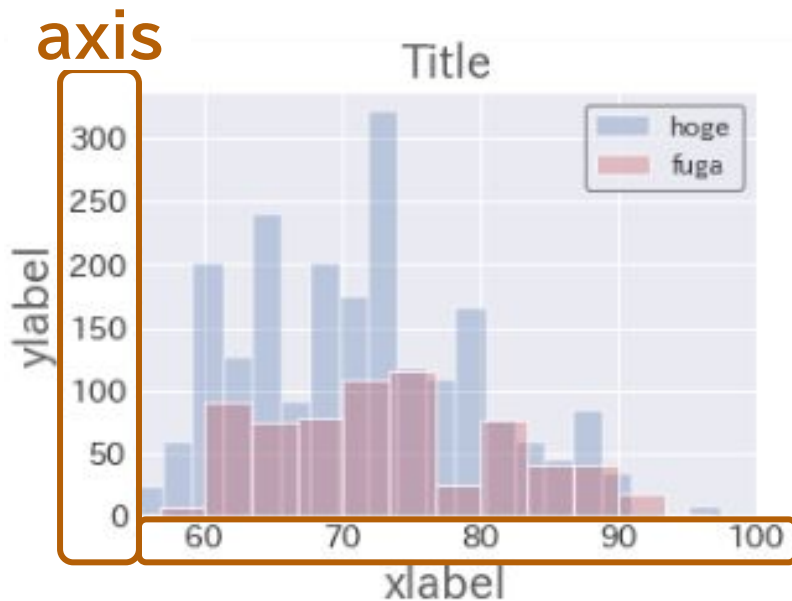




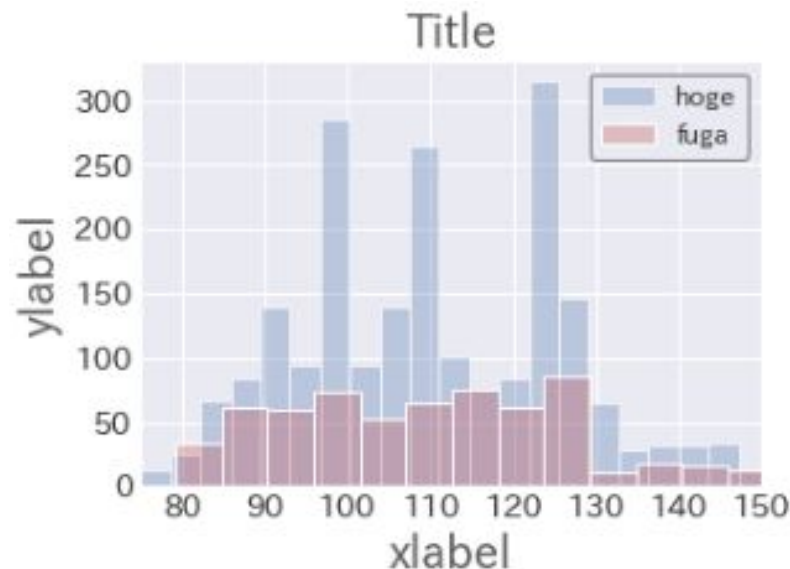
# matplotlibのインターフェイス

## figure

axes



axes



# 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のみで紹介したが、実はこの関数は「**Figureの新規作成**」という意味もある（pyplotで一つしかFigureを作らない場合は省略可能なのであまり意識しなくてもOK）

Figureのなかに複数のAxesを生成する関数  
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。

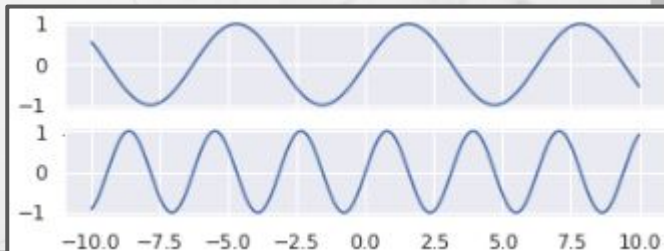
`plt.subplot( 2, 1, 2 )`

縦に **2行**、横に **1列** の Axes のうちの **2つ目**

① `# グラフの大きさを指定`  
`plt.figure(figsize=(20, 6))`

② `# 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))`

③ `# 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番号の指定の仕方

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



A complex network diagram with numerous nodes (dots) and connecting lines, creating a web-like structure. The nodes are of varying sizes and are connected by thin, light gray lines. The overall pattern is dense and fills the entire background.

# Notebook^



# 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

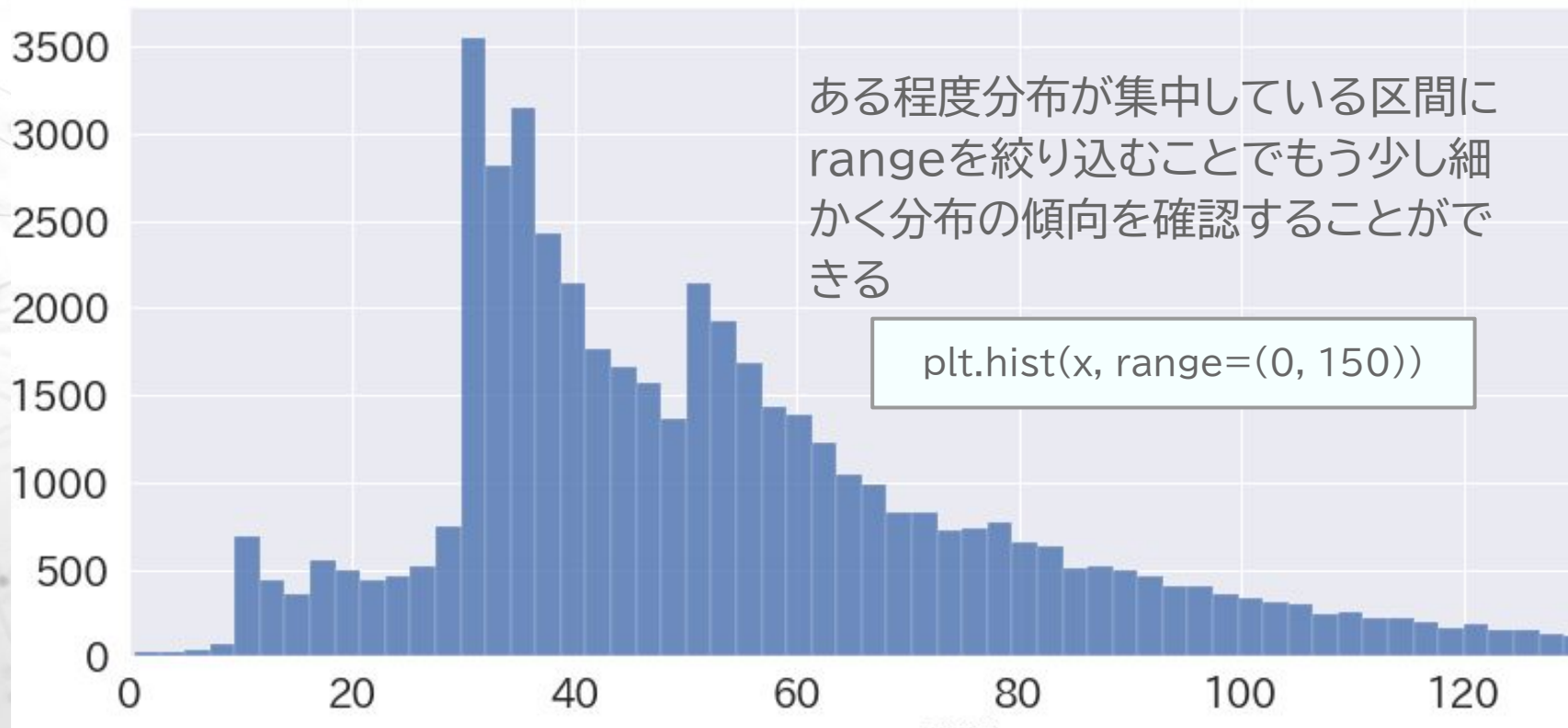
y軸のカラム名の指定

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)。

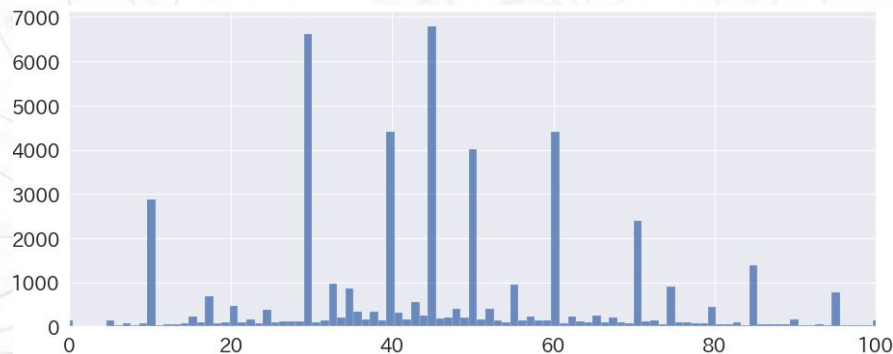
# 引数rangeの指定をしなかった例



# 引数rangeの指定をした場合

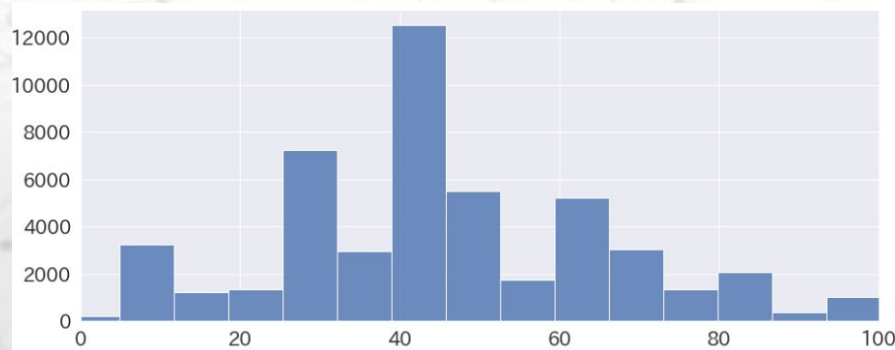


# ビン切りを変えてデータを観察する



## 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

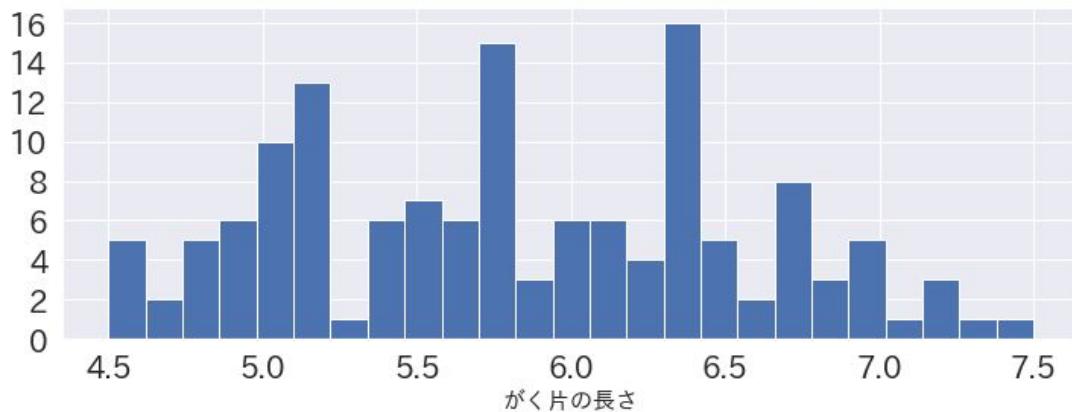


## ビンを大まかに切った場合

- 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)
```

これもほとんど使わない引数が大部分なので覚える必要はない

# 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]

①

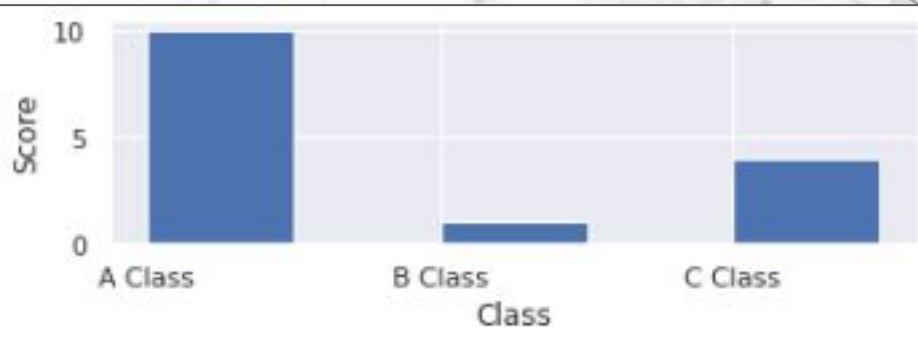
②

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

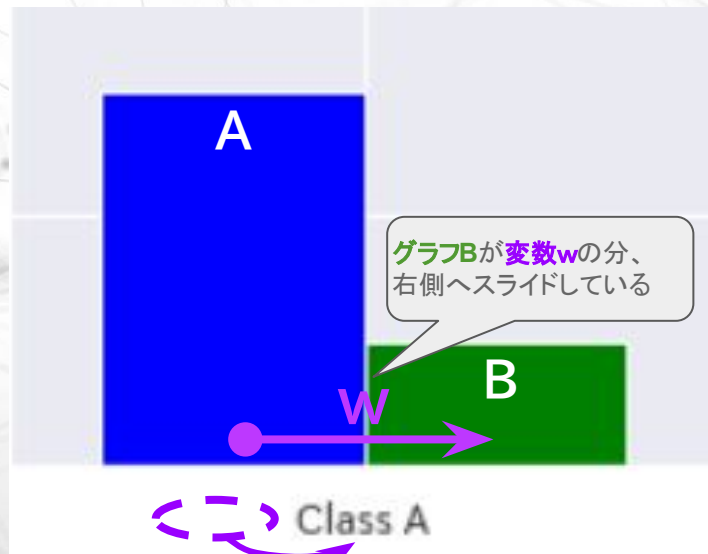
③

# 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



グラフBが変数 $w$ の分、  
右側へスライドしている

目盛りラベルが変数 $w \div 2$ の分、  
右側へスライドしている

$w = 0.4$

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: 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]

```
labels = ['Frogs',  
sizes = [15,  
colors = ['yellowgreen',  
explode = (0,
```

[1]

```
'Hogs',  
30,  
'gold',  
0.1,
```

[2]

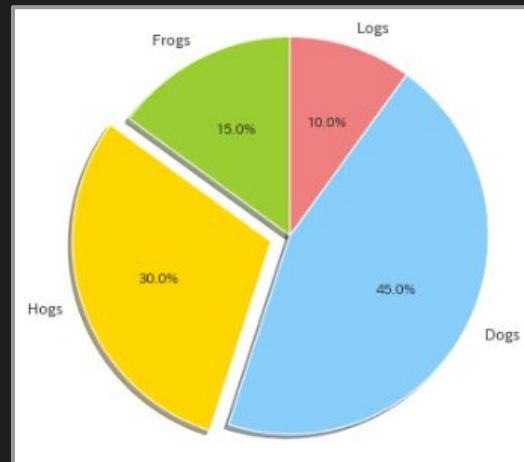
```
'Dogs',  
45,  
'lightskyblue',  
0,
```

[3]

```
'Logs']  
10]  
'lightcoral']  
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。

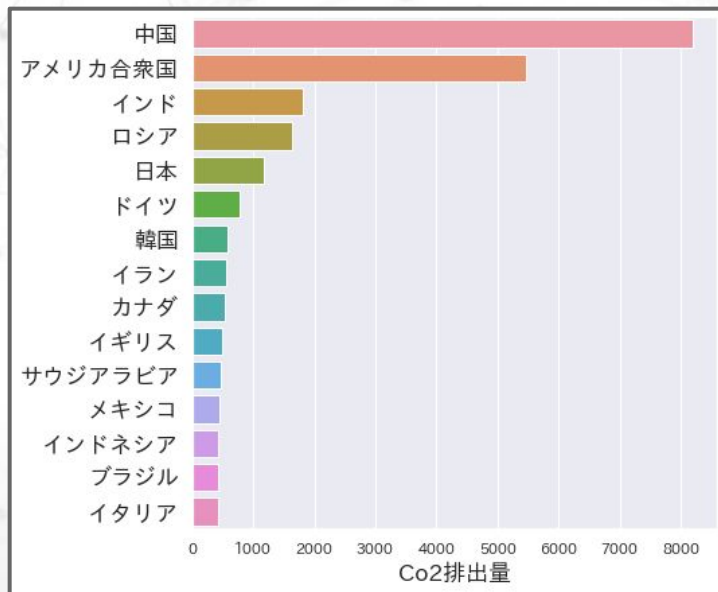
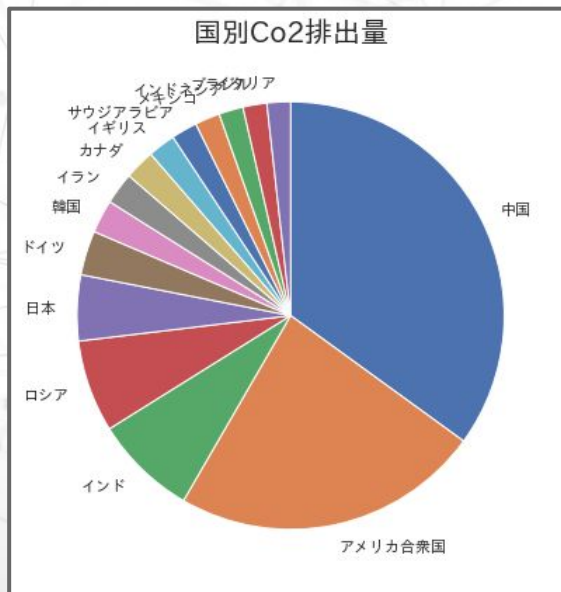
<https://jikitourai.net> > 生活・コラム ▾  
**円グラフを安易に使ってはいけない理由を実例を交えて説明し ...**  
2021/04/08 — 見た目がカッコいいからと3D円グラフを使っていないでしょうか？円グラフは作成が簡単なので使いがちなグラフですが、見る人の認識を誤らせやすい ...  
サンプルで見るダメな円グラフ・3D円グラフの問題点・平面の円グラフではどうか  
22/05/07 にこのページにアクセスしました。

<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。

- 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いておく
- 実務家からも「使うべきでない」という意見が上がっている(※)

※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

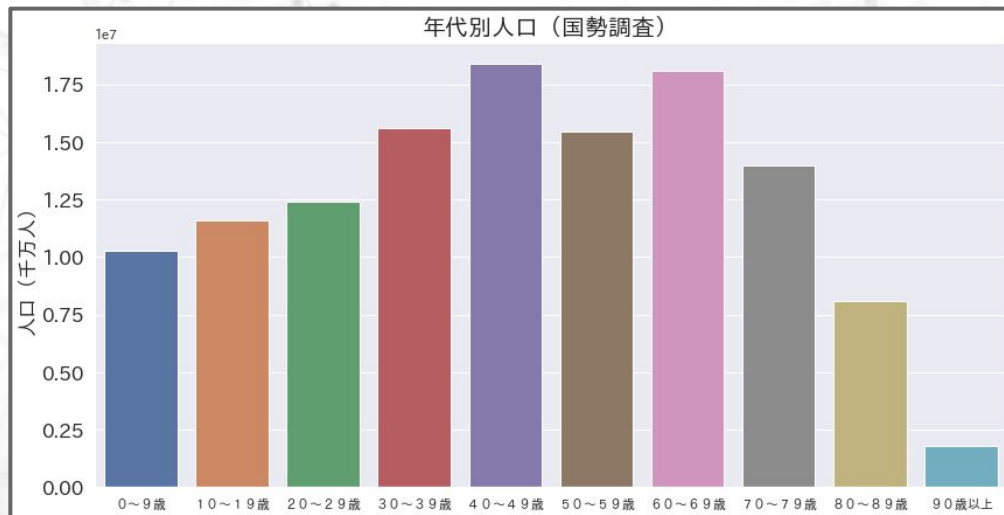
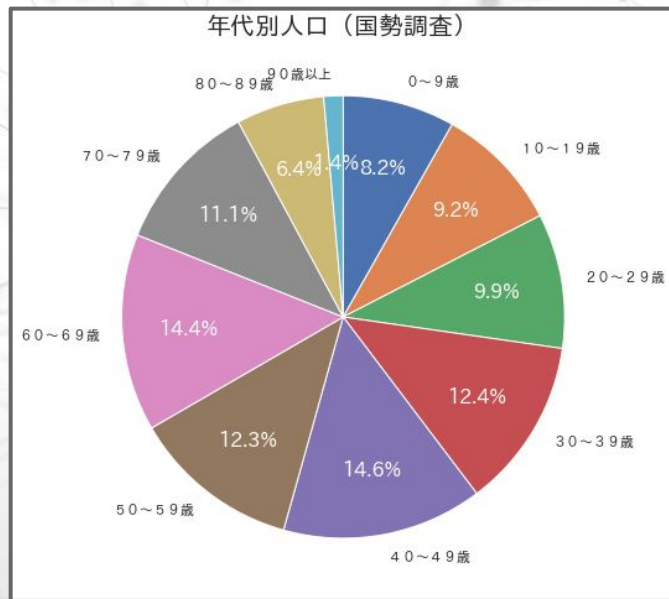
# 円グラフは使い方に注意が必要



- 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
- 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



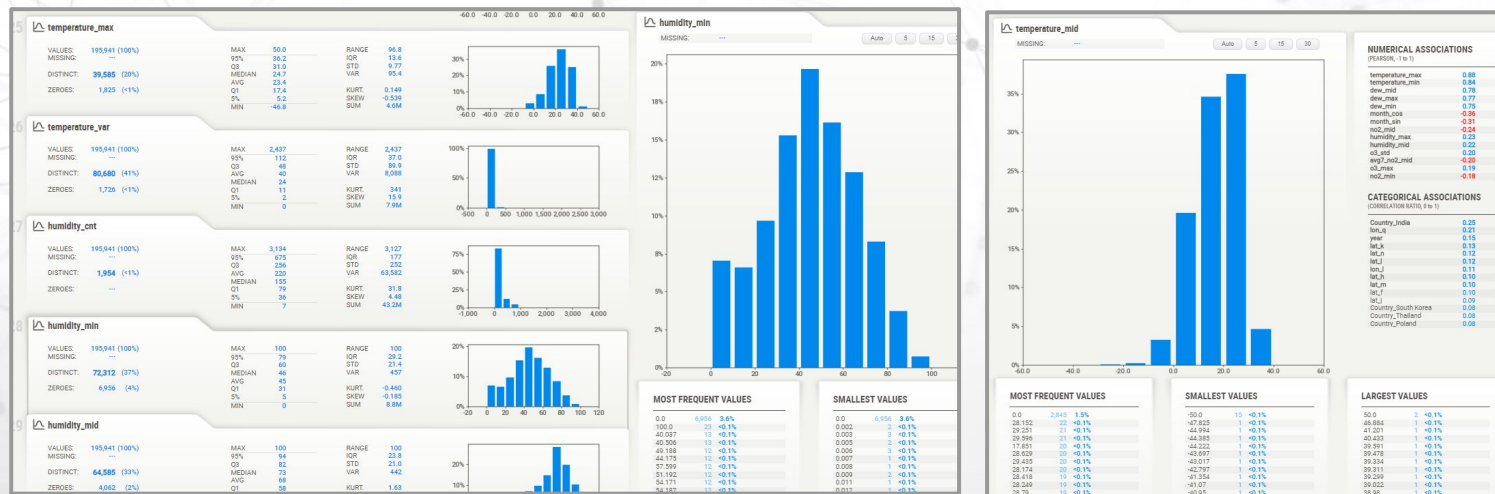
# 円グラフは使い方に注意が必要



- 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*

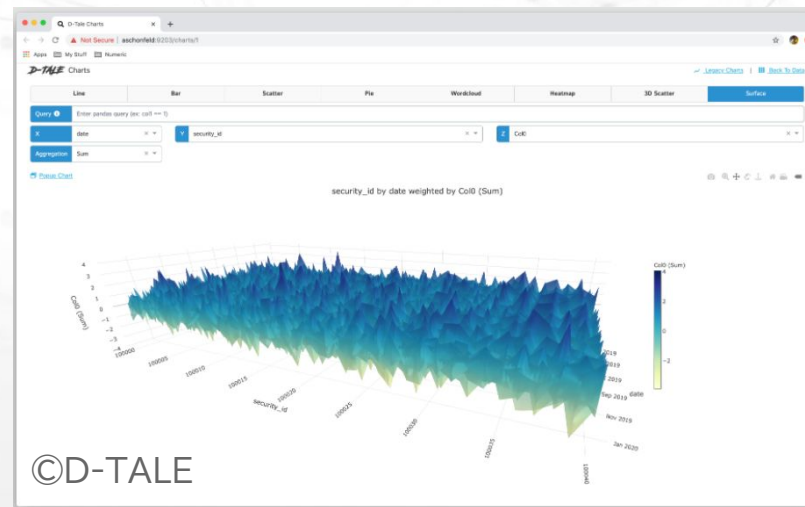
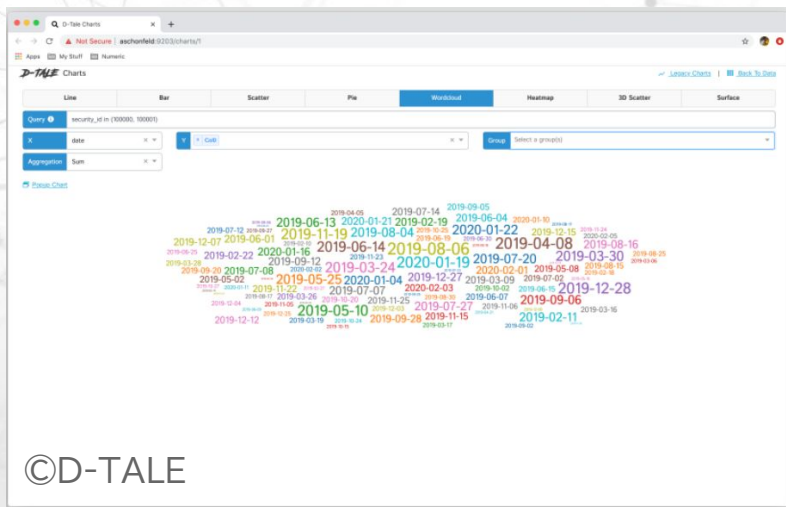
<https://github.com/fbdesignpro/sweetviz>



# EDA自動化ライブラリの例:

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

date	security_id	int_val	Col0	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col10	Col11
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
2019-09-13	100001	3227834703	0.81	-0.48	-2.25	-0.36	1.05	0.71	-0.01	0.40	0.63	-0.24	-1.62	1.99
2019-09-13	100002	1430132283	0.27	1.15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2019-09-13	100003	1053490088	0.34	-0.24	-0.07	-0.37	0.08	0.95	-0.05	-0.05	-0.05	-0.05	-0.05	-0.05
2019-09-13	100004	101747011	-0.66	-1.30	-0.11	1.02	0.44	1.81	-0.05	-0.05	-0.05	-0.05	-0.05	-0.05
2019-09-13	100005	2054840457	0.70	0.71	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2019-09-13	100006	2443483404	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2019-09-13	100007	9419927953	0.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2019-09-13	100008	7297483636	-0.03	0.29	0.06	-0.55	0.00	-0.03	0.00	0.00	-0.03	-0.03	-0.03	-0.03
2019-09-13	100009	2454840457	0.70	0.29	0.06	-0.55	0.00	-0.03	-0.03	-0.03	-0.03	-0.03	-0.03	-0.03
2019-09-13	100010	5572650089	-0.24	-0.04	0.29	1.63	-2.35	-0.19	0.96	0.32	-1.19	0.96	-1.02	0.43
2019-09-13	100011	7767970090	-0.41	0.34	-1.21	0.17	-0.14	-0.35	-1.54	-0.74	-0.81	-1.05	0.57	-0.20
2019-09-13	100012	3540402206	0.26	0.93	0.09	-0.11	0.77	-0.41	0.25	-1.92	1.06	-1.74	-0.38	-0.36



The background of the slide is a complex network diagram. It consists of numerous small, light-gray circular nodes scattered across the frame. These nodes are interconnected by a dense web of thin, light-gray lines, creating a mesh-like structure that resembles a molecular model or a data network. The overall aesthetic is technical and modern.

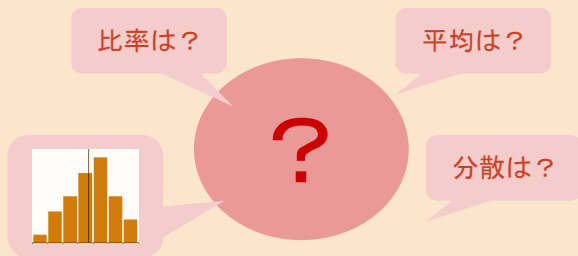
# 統計の基礎



# 記述統計と推計統計

## 記述統計

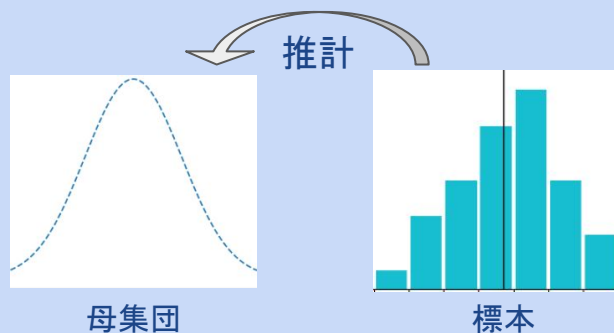
手元の標本データを把握して  
要約・記述する方法



ここにいる集団の性質を把握

## 推計統計

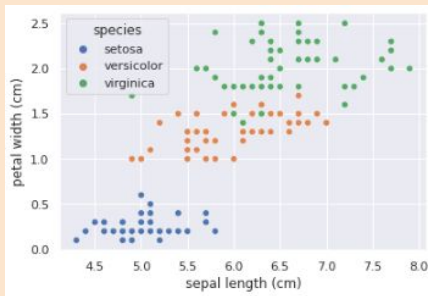
手元の標本データに基づき  
その背後にある母集団について  
推計する方法



# 量的データと質的データ

## 量的データ

四則演算が有効な数値で比率に  
意味がある(金額・時間・年齢etc.)



連続値も離散値も含む

## 質的データ

カテゴリや状態を表現するための  
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  
できないこともないが、分析上の意味はもたない

# 平均値と中央値のどちらを使うか

## 平均値

変量の総和を個数で割ったもの

$$(1 + 2 + 100) \div 3$$

平均値は「34.33」

## 中央値

分布の中央にくる値のこと

1, 2, 100

「中央値は2」



```
dataframe['columns_name'].mean()
```



```
dataframe['columns_name'].median()
```

平均値は外れ値の影響を受けやすいことに注意を要する

# 最頻値を併せて使う

## 最頻値とは

最も度数(頻度)が多い値のこと

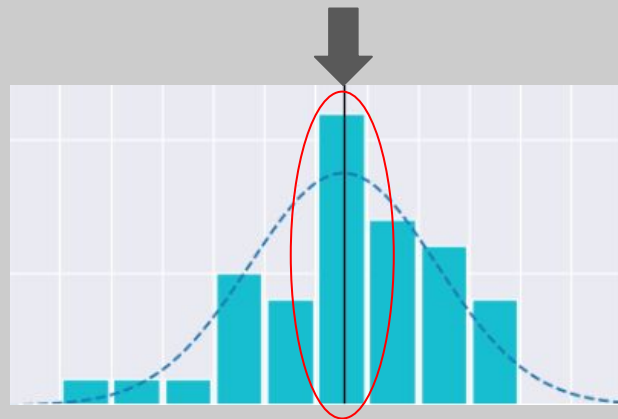
1, 2, 3, 3, 4  
= 最頻値は3

外れ値の影響は受けない



```
dataframe['columns_name'].mode()
```

ヒストグラムで一発でわかる

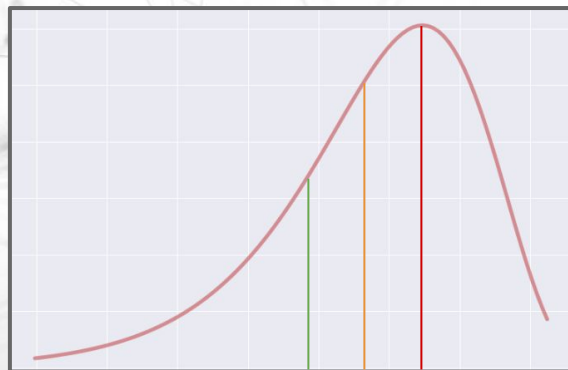


最頻値は山が複数あるような分布のデータでは注意が必要



# 平均値と中央値と最頻値の関係

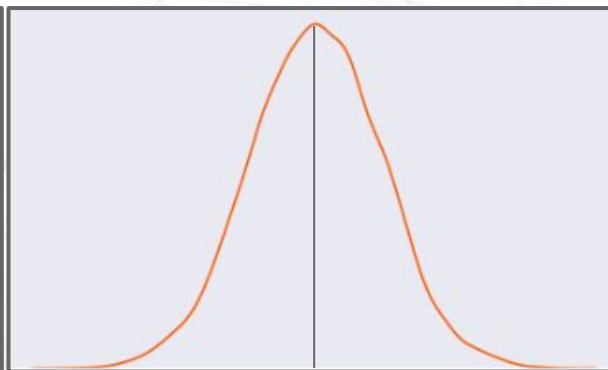
右に偏ったデータ



平均値 中央値 最頻値

平均値 < 中央値 < 最頻値

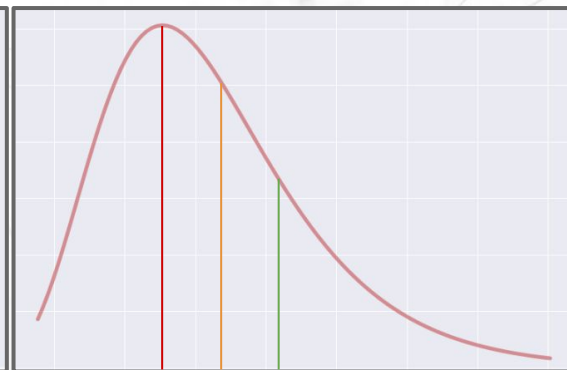
正規分布に近いデータ



最頻値 = 平均値 = 中央値

平均値 = 中央値 = 最頻値

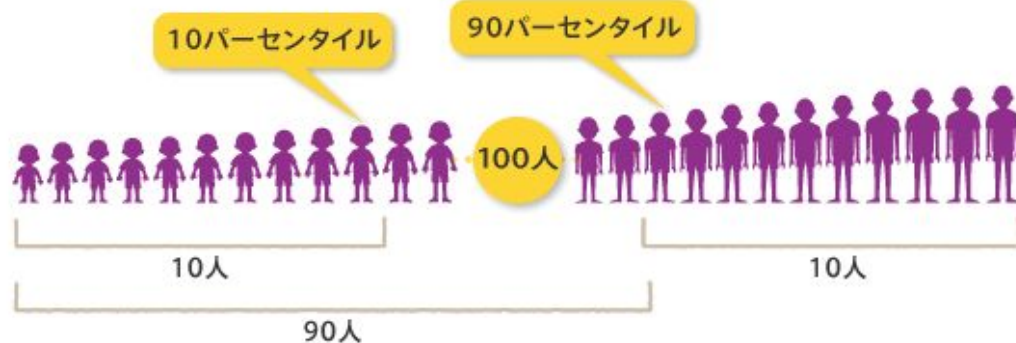
左に偏ったデータ



最頻値 中央値 平均値

最頻値 < 中央値 < 平均値

# 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何番になるのかを示す数値

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

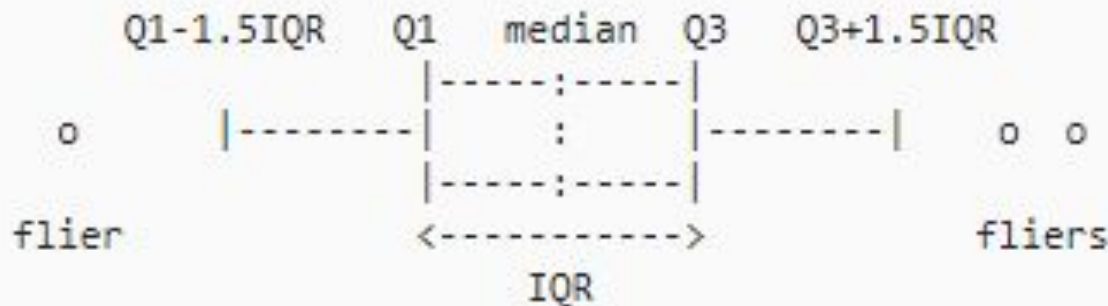
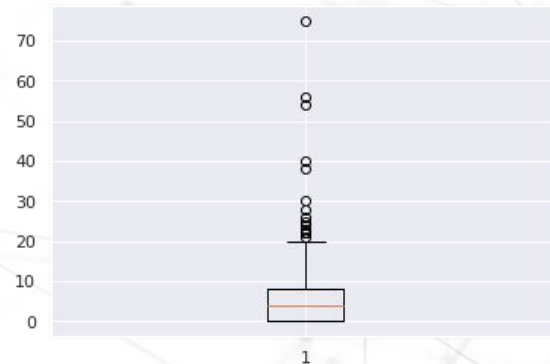
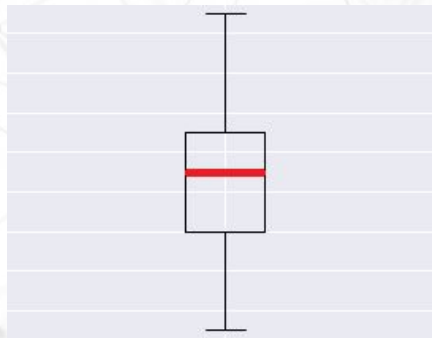
count	395.000000
mean	5.708861
std	8.003096
min	0.000000
25%	0.000000
50%	4.000000
75%	8.000000
max	75.000000
Name: absences, dtype: float64	

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パーセント
箱内部の赤い線	第2四分位 50パーセント 中央値
箱の上底	第3四分位 75パーセント
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

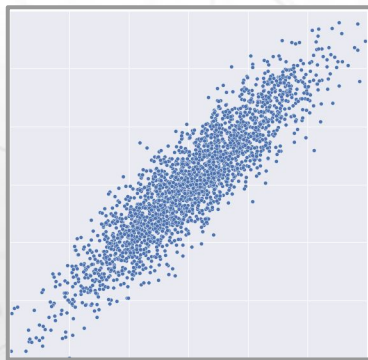


matplotlib公式ドキュメントより

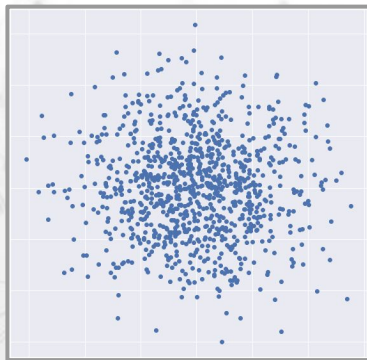
[https://matplotlib.org/stable/api/as\\_gen/matplotlib.pyplot.boxplot.html](https://matplotlib.org/stable/api/as_gen/matplotlib.pyplot.boxplot.html)

# 相関係数の性質

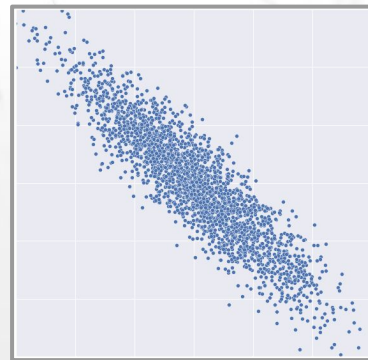
- 「-1」から「1」までの値を取る
- 正の相関が強いと相関係数が1に近づく
- 負の相関が強いと相関係数が-1に近づく
- 相関係数が「1」or「-1」のときは完全相関という
- 相関係数が0の付近は相関がないといえる
- 相関関係は因果関係とは異なる
- あくまで、どれだけ比例的な関係を持っているかを計るもの
- 「1」でも「-1」でも分析には等しく有効



正の相関



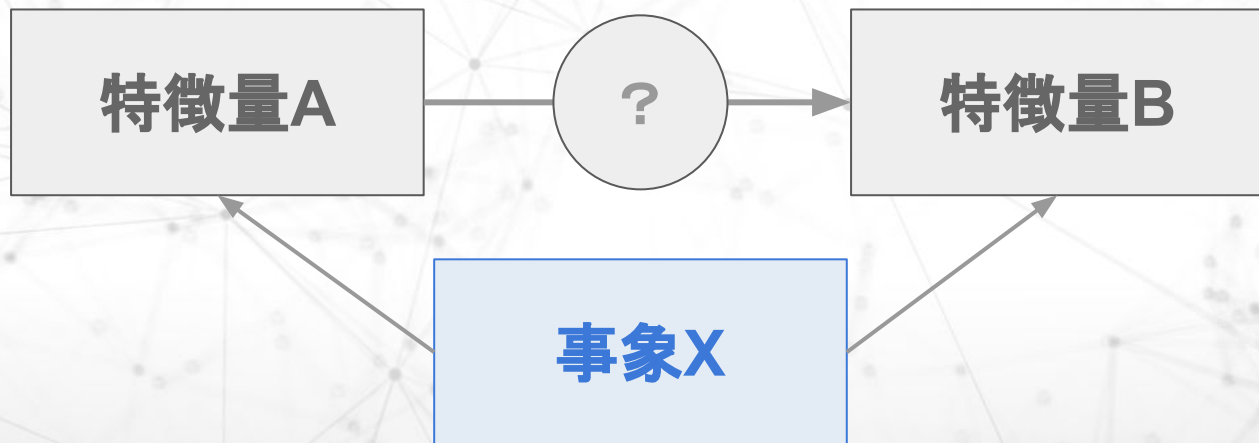
相関がない



負の相関

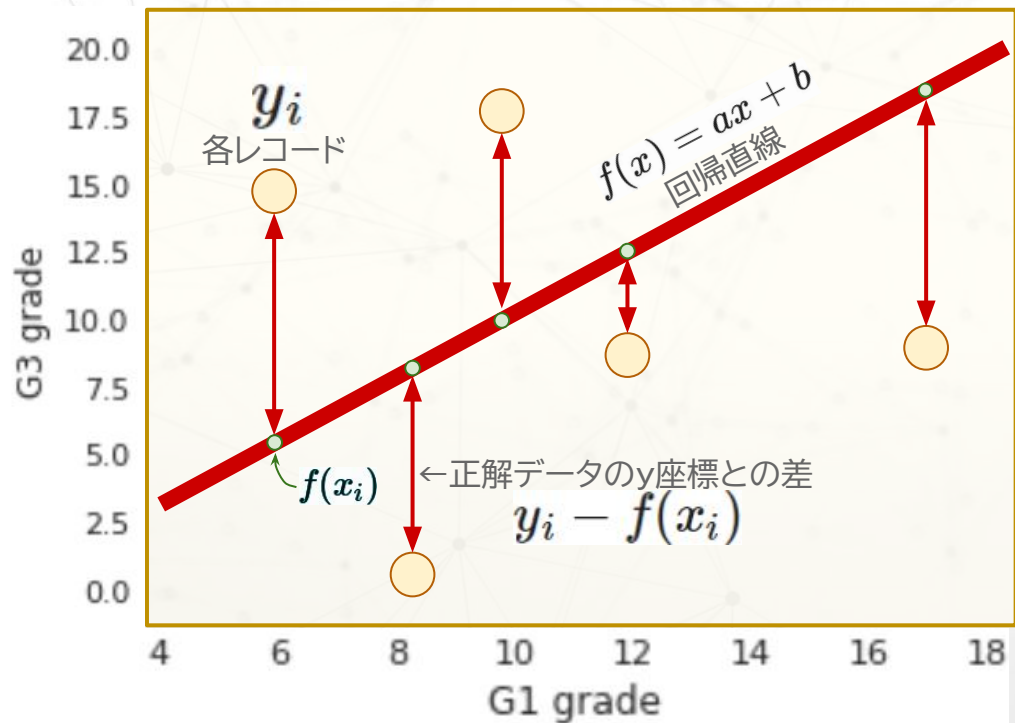


# 相関関係は因果関係とは異なる



- 特徴量AとBの間に高い相関係数があっても、第三の要因Xが双方に等しく原因として作用している可能性もある
- そもそも完全な偶然である場合も多々ありうる
- 因果関係は機械的に求められるものではなく、ドメイン知識と併せた丁寧な考察が必要

# 最小二乗法による単回帰分析



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

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。  
(コーディング上は略称の「sklearn」を使うので気を付ける。)

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、これらは次週で本格的に学ぶ。

```
[ ] 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

今回は モジュール名.関数 という形で呼び出している。

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

※モジュール名(numpyやpandasでいうと「np.」や「pd.」に相当する部分)

※以下のインポート方法により、緑色下線部分の表記を省く場合もある。

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

①機械学習モデルを作成するコンストラクタ関数  
(※クラスインスタンスを作成する関数のことをコンストラクタ関数という。)

②コンストラクタ関数により左辺の変数(ここでは「reg」)に機械学習モデルが格納される。





# 機械学習モデルはメソッドで操作する



## 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)
```



# 単回帰分析モデルの結果表示関係のメソッド

**.coef\_**

回帰係数を表示

**.score(X, Y)**

決定係数を表示



**.intercept\_**

切片を表示

その他にもいろいろ！  
(必要に応じて公式ドキュメントを参照する)

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。