2019「先進ゲノム支援」情報解析講習会（10月10日）

matplotlib & seaborn によるデータの視覚化

孫 建強

農業・食品産業技術総合研究機構
農業情報研究センター

視覚化

pandas

$y_i t = \beta' x_{it} + \mu_i + \epsilon_{it}$

matplotlib をベースとしている視覚化ライブラリーである。pandas データ構造をグラフ化できる。ただし、複雑なグラフや細かい部分の調整が難しい。

ggplot

R の ggplot2 とほぼ同じような使い方で、ほぼ同じような仕上がりとなる。The grammar of graphics と呼ばれる文法に従って記述する必要がある。

matplotlib

matplotlib は初期から存在する視覚化ライブラリーである。使用者が多いために、情報量も多い。複雑なグラフや細かい調整などが可能である。

plotly

ウェブベースのインタラクティブなグラフを作成できる。matplotlib に比べて勉強しやすいと言われている。

seaborn

matplotlib をベースとしている。matplotlib を補完する位置付けである。最近にリリースされたライブラリーである。複雑なグラフも簡単に作成できる。

Bokeh

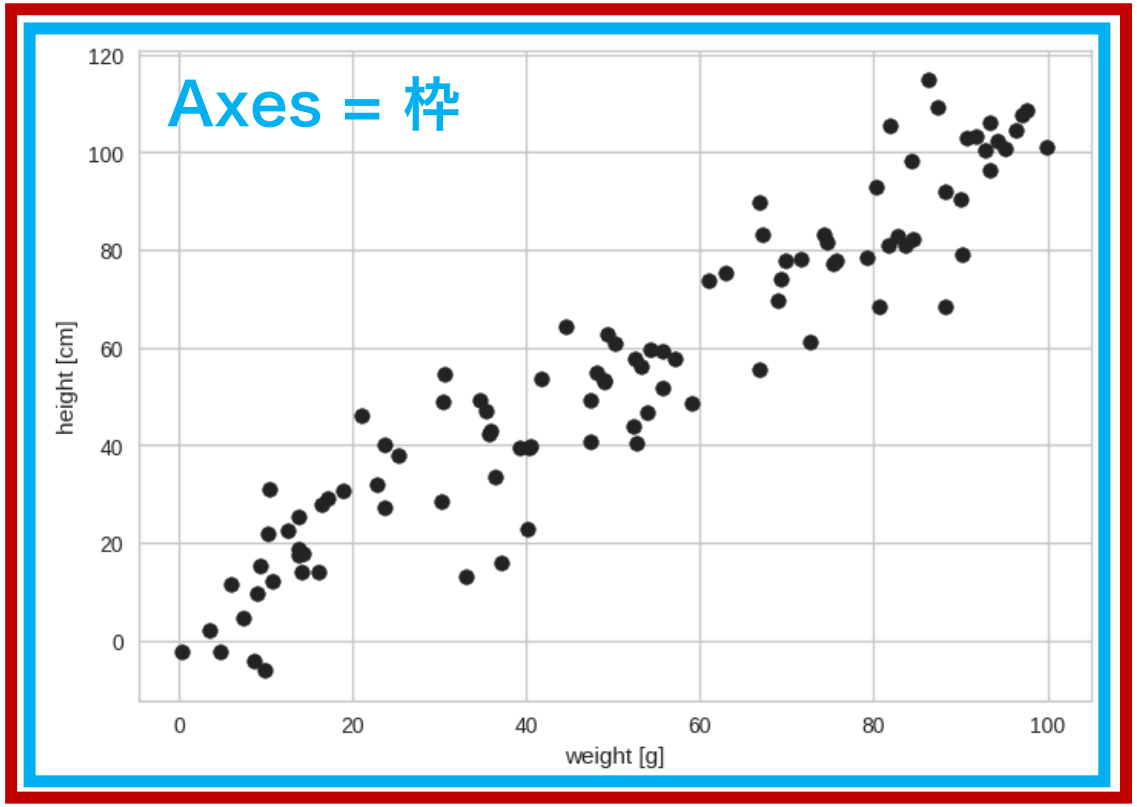
ウェブベースのインタラクティブなグラフを作成できる。The grammar of graphics と呼ばれる文法に従って記述する必要がある。

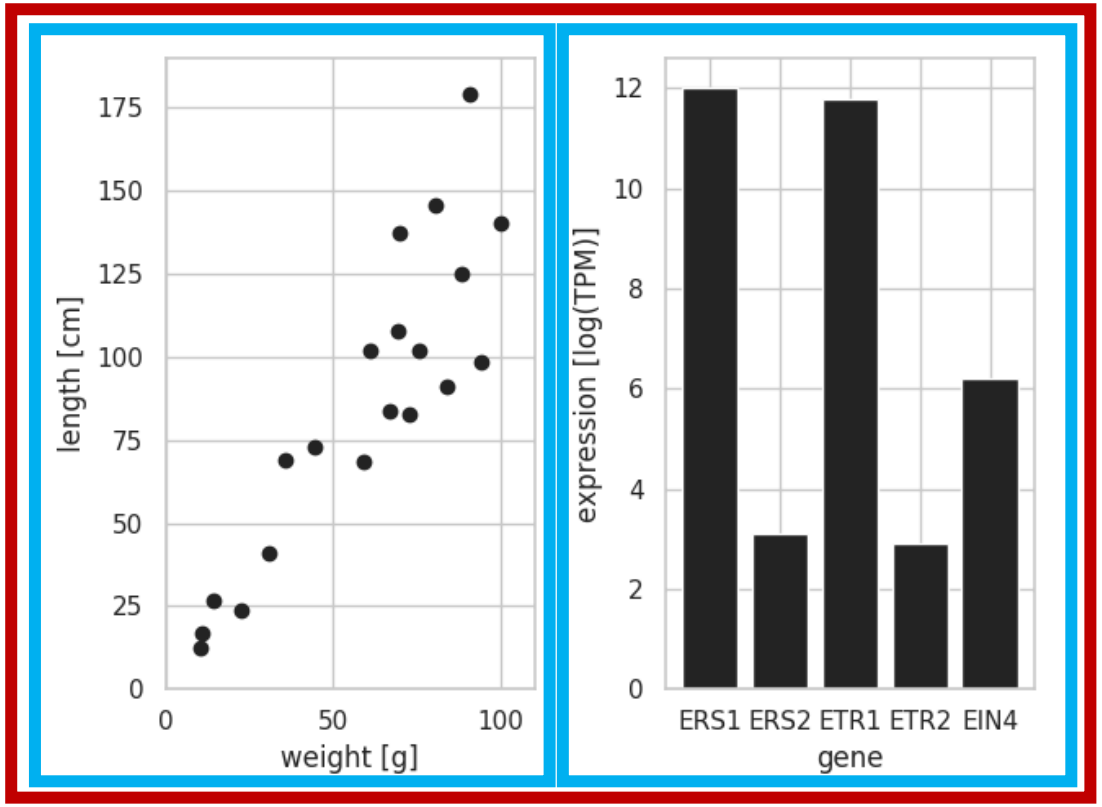
matplotlib グラフ作成

matplotlib グラフ作成

Pyplot = 落書き帳

Figure = ページ

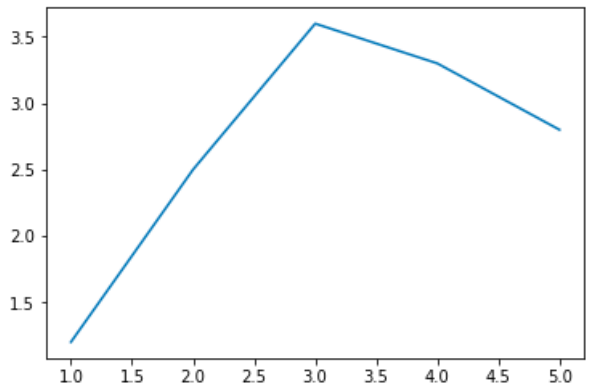




matplotlib グラフ作成

matplotlib を利用したグラフ作成は、pyplot モジュール中のメソッドを使用する。

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
fig.show()
```



matplotlib グラフ作成

matplotlib を利用したグラフ作成は、pyplot モジュール中のメソッドを使用する。

1. matplotlib の機能を呼び出す。pyplot 描画領域が用意される。

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
fig.show()
```

matplotlib グラフ作成

matplotlib を利用したグラフ作成は、pyplot モジュール中のメソッドを使用する。

1. matplotlib の機能を呼び出す。pyplot 描画領域が用意される。
2. Figure クラスのオブジェクトを用意する。

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
fig.show()
```

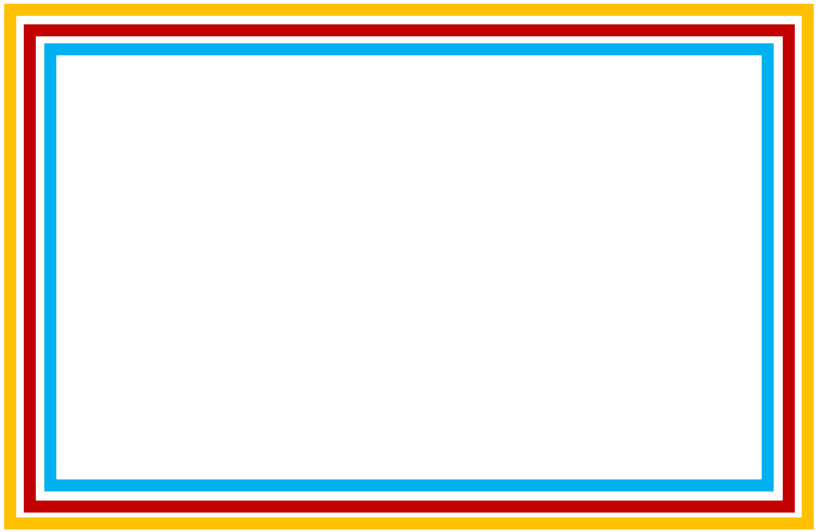
matplotlib グラフ作成

matplotlib を利用したグラフ作成は、pyplot モジュール中のメソッドを使用する。

1. matplotlib の機能を呼び出す。pyplot 描画領域が用意される。
2. Figure クラスのオブジェクトを用意する。
3. Figure 領域を1行1列に分割し、分割領域の1番目の領域でAxesクラスのオブジェクトを作成する。

▶

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
fig.show()
```

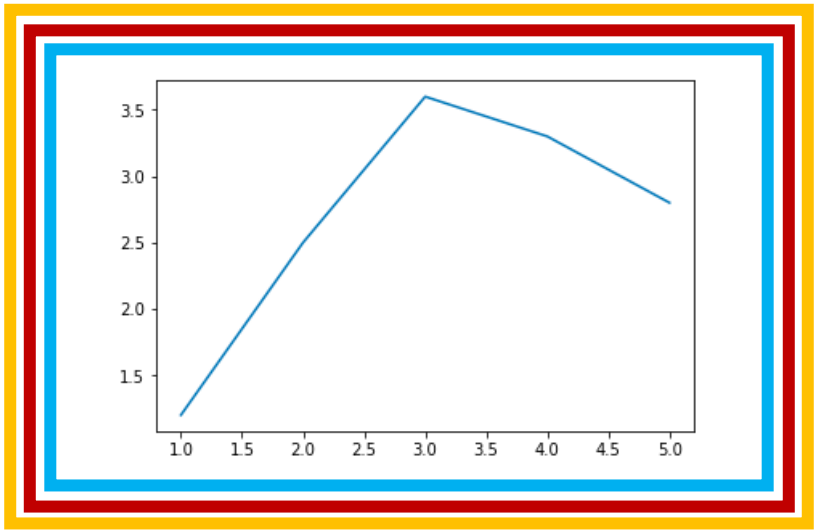


matplotlib グラフ作成

matplotlib を利用したグラフ作成は、pyplot モジュール中のメソッドを使用する。

1. matplotlib の機能を呼び出す。pyplot 描画領域が用意される。
2. Figure クラスのオブジェクトを用意する。
3. Figure 領域を1行1列に分割し、分割領域の1番目の領域でAxesクラスのオブジェクトを作成する。
4. 1番目のAxesオブジェクトに線グラフを描く。

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
fig.show()
```

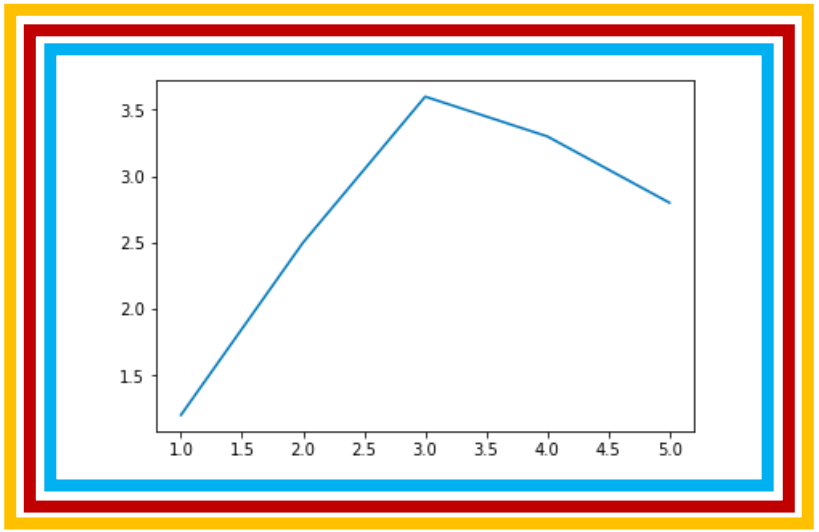


matplotlib グラフ作成

matplotlib を利用したグラフ作成は、pyplot モジュール中のメソッドを使用する。

1. matplotlib の機能を呼び出す。pyplot 描画領域が用意される。
2. Figure クラスのオブジェクトを用意する。
3. Figure 領域を1行1列に分割し、分割領域の1番目の領域でAxesクラスのオブジェクトを作成する。
4. 1番目のAxesオブジェクトに線グラフを描く。
5. これまでに描いたグラフをディスプレイ上に表示させる。

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
fig.show()
```

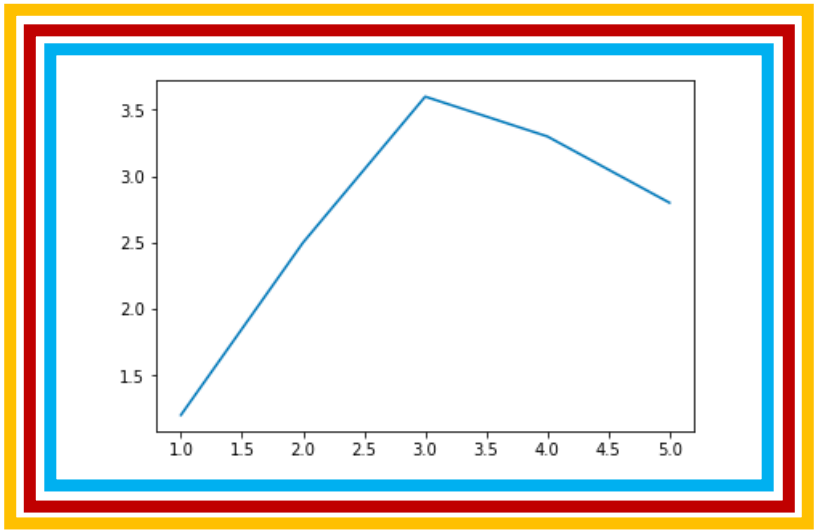


matplotlib グラフ保存

matplotlib を利用したグラフ作成は、pyplot モジュール中のメソッドを使用する。

1. matplotlib の機能を呼び出す。pyplot 描画領域が用意される。
2. Figure クラスのオブジェクトを用意する。
3. Figure 領域を1行1列に分割し、分割領域の1番目の領域でAxesクラスのオブジェクトを作成する。
4. 1番目のAxesオブジェクトに線グラフを描く。
5. これまでに描いたグラフをファイルにに保存する。

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
fig.savefig('fig1.png', format='png')
```



matplotlib グラフ保存

グラフを画像ファイルへ書き出すとき、show メソッドの代わりに savefig メソッドを使用する。画像ファイルのフォーマットは format オプションで指定する。PNG の他に PDF、PS、EPS、SVG を指定できる。

グラフのメモリ軸などに使う文字のフォントサイズなどを調整する場合は、set_xlabel などのメソッドを使う。また、画像ファイルのサイズや解像度などを調整したい場合は、描画デバイスを呼び出す際に行う。

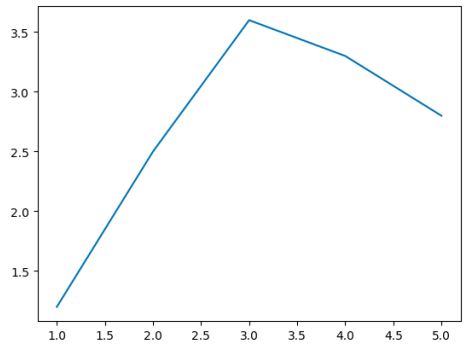
```
import numpy as np
import matplotlib.pyplot as plt


x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])

fig = plt.figure(figsize=[8, 6], dpi=300)
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)


ax.set_xlabel("xlabel", fontsize=18)
ax.set_ylabel("ylabel", fontsize=18)
ax.tick_params(labelsize=18)


fig.savefig('fig1.png', format='png')
```

基本グラフ

この資料と同様なグラフを描くには、matplotlibをインポートするほかに、seabornもインポートして、背景およびスタイルを変更を行なってください。

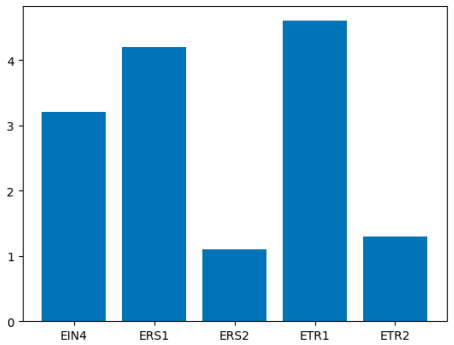
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
sns.set_style('whitegrid')
sns.set_palette('gray')
```

基本グラフ



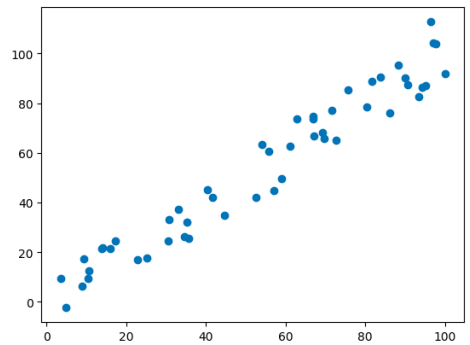
line chart

データの時系列的な変化傾向を視覚化するとき目的で使われる。



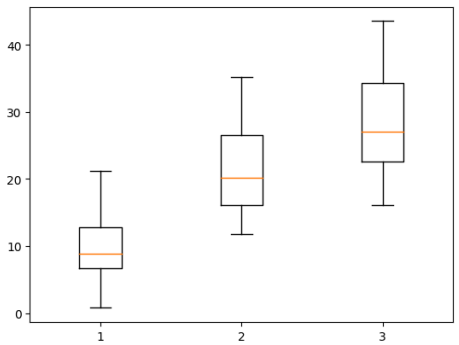
bar chart

カテゴリ毎に分類されるデータを視覚化する目的で使われる。



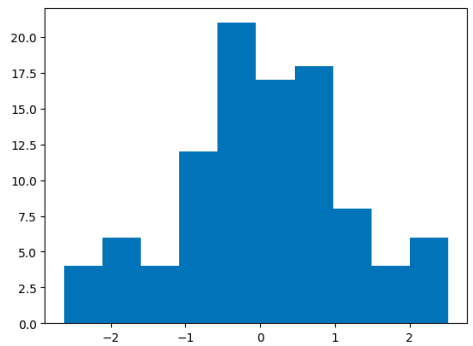
scatter chart

2 変量の連続値データ同士の相関や分布などを視覚する目的で使われる。



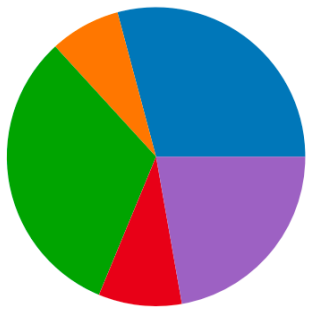
boxplot

カテゴリ毎に分類されるデータの分布などを視覚化する目的で使われる。



histogram

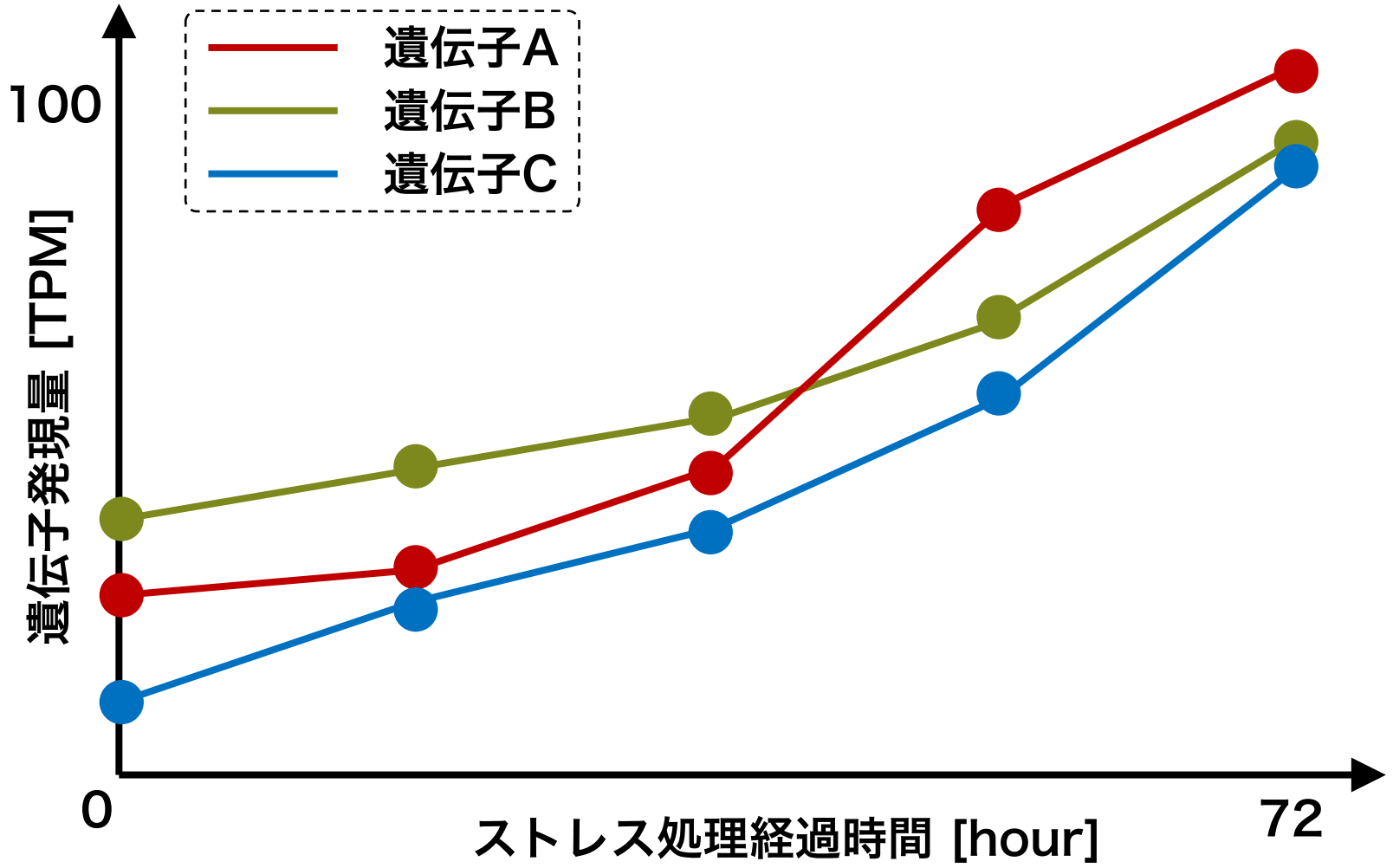
1 変量の連続値データの分布などを視覚化する目的で使われる。



pie chart

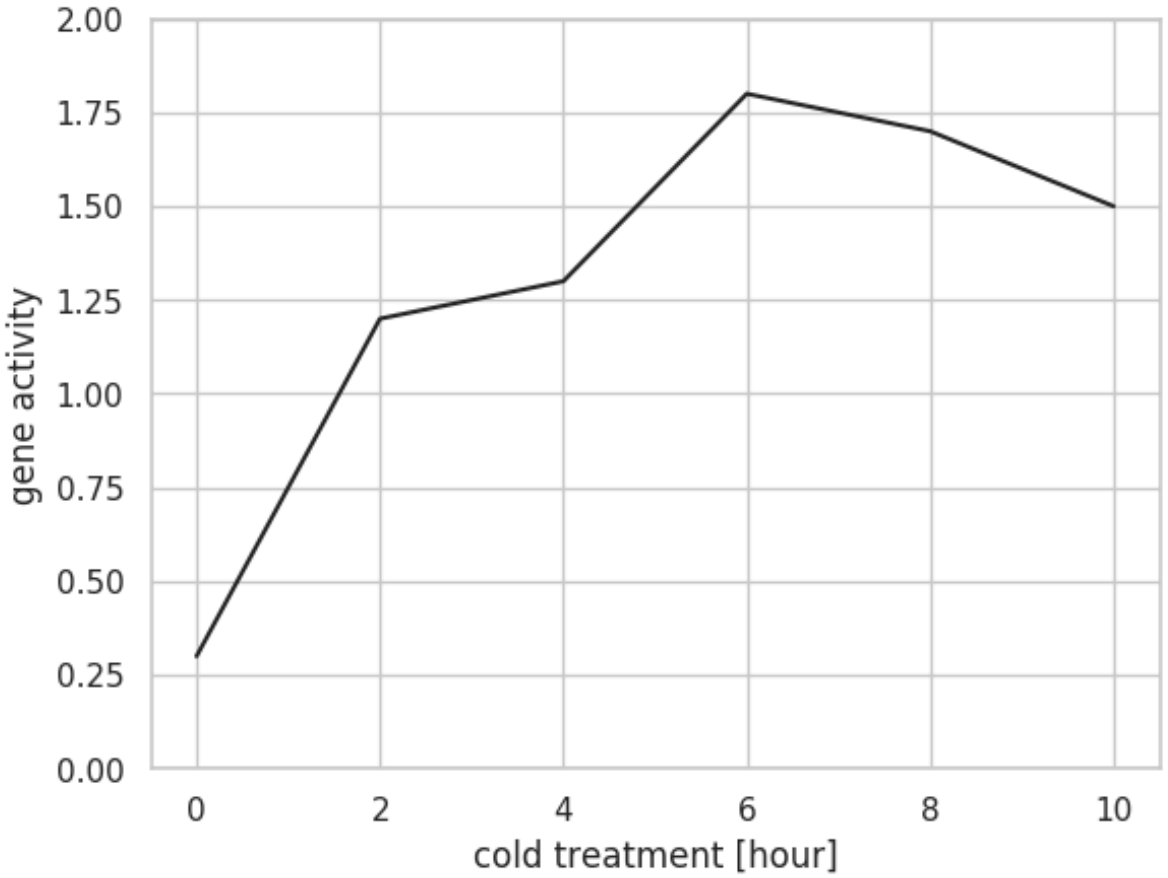
割合データを視覚する目的で使われる。

線グラフ



- 線グラフは基本的にデータの系列的な変化を見るためのグラフ
- 縦軸および横軸は連続量
- 原点 (0, 0) が省略されているグラフを十分に注意すること
- 観測値を強調するために、観測値には点を明示することもある

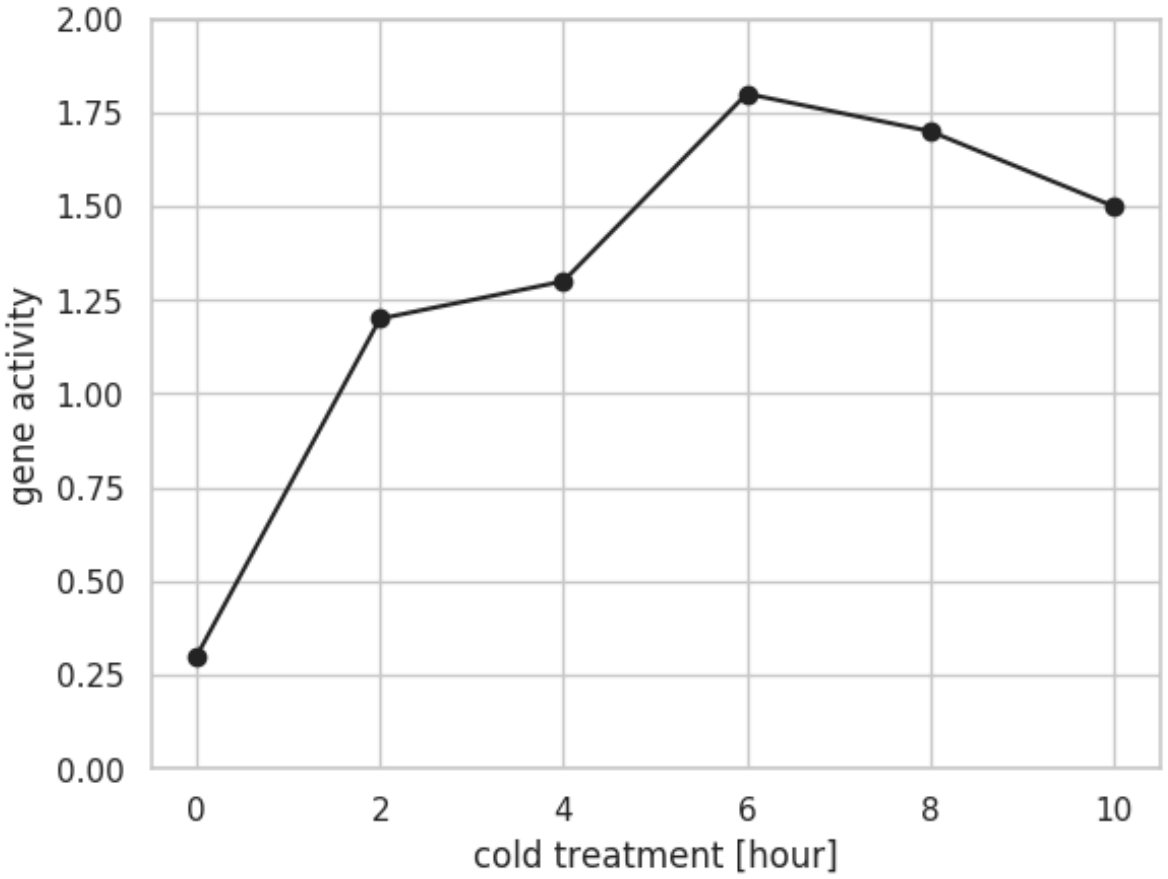
線グラフ




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

x = np.array([0, 2, 4, 6, 8, 10])
y = np.array([0.3, 1.2, 1.3, 1.8, 1.7, 1.5])

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
ax.set_xlabel('cold treatment [hour]')
ax.set_ylabel('gene activity')
ax.set_ylim(0, 2)
fig.show()
```

set_xlabel、set_ylabel、set_ylim などの機能を、関数の名前と出力グラフと見比べながら推定してみよう。

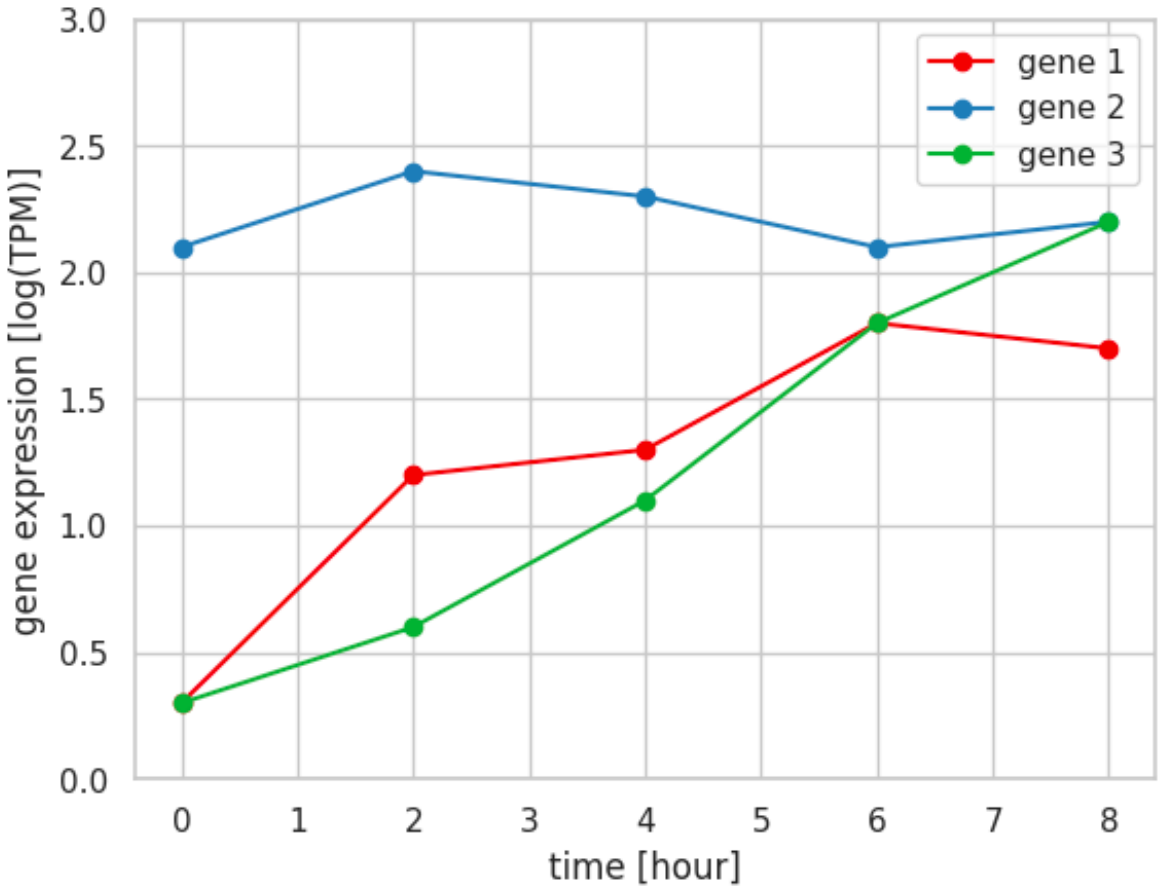
線グラフ




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

x = np.array([0, 2, 4, 6, 8, 10])
y = np.array([0.3, 1.2, 1.3, 1.8, 1.7, 1.5])

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y, marker='o')
ax.set_xlabel('cold treatment [hour]')
ax.set_ylabel('gene activity')
ax.set_ylim(0, 2)
fig.show()
```

marker の値を変えることで、様々なマーカーがプロットされる。matplotlib で利用できるマーカー一覧は次のページで確認できる。 https://matplotlib.org/api/markers_api.html

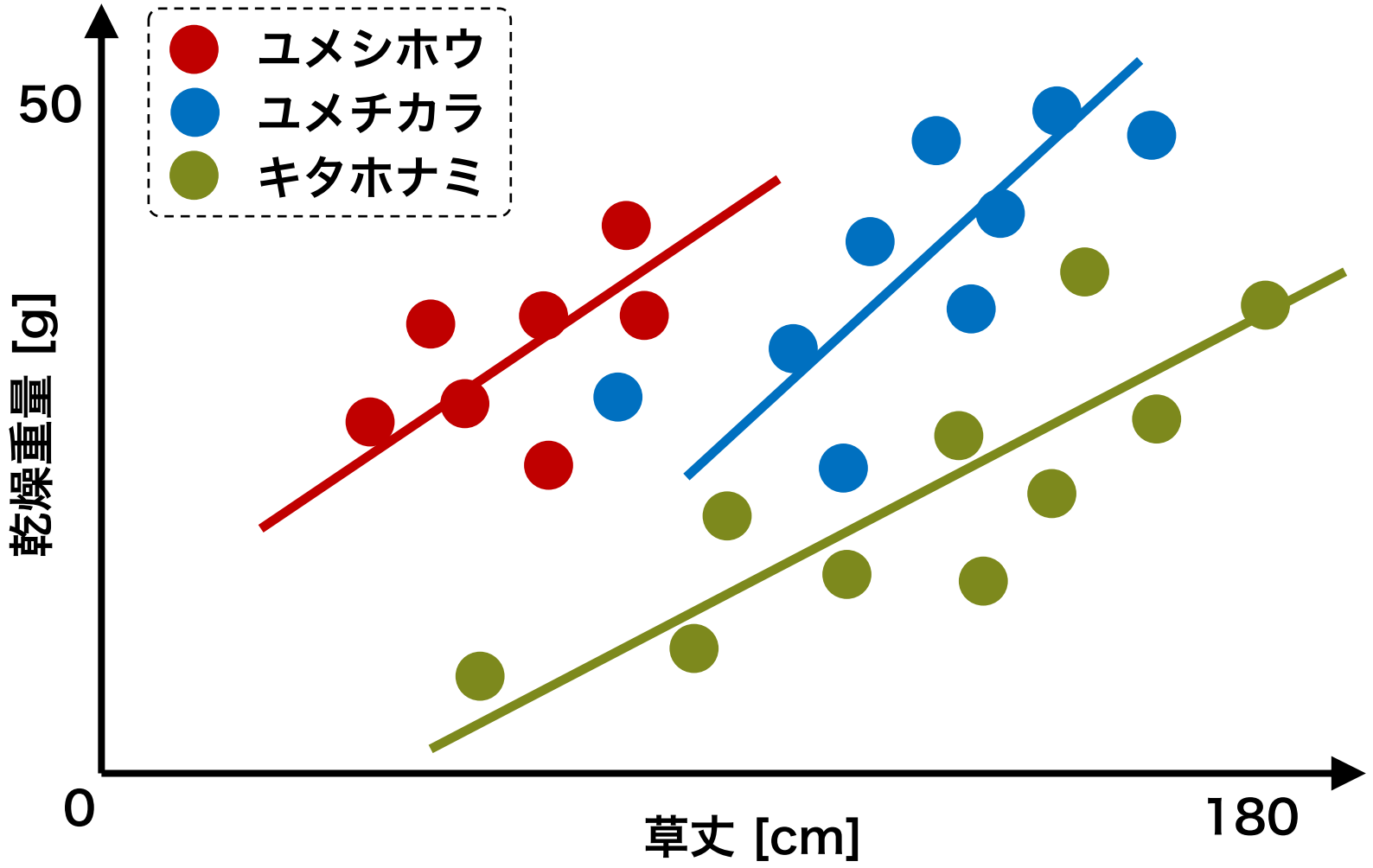
線グラフ




```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([0, 2, 4, 6, 8])
gene_1 = np.array([0.3, 1.2, 1.3, 1.8, 1.7])
gene_2 = np.array([2.1, 2.4, 2.3, 2.1, 2.2])
gene_3 = np.array([0.3, 0.6, 1.1, 1.8, 2.2])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, gene_1, label='gene 1',
        marker='o')
ax.plot(x, gene_2, label='gene 2',
        marker='o')
ax.plot(x, gene_3, label='gene 3',
        marker='o')
ax.legend()
ax.set_title('Gene expression')
ax.set_xlabel('time [h]')
ax.set_ylabel('log10FPKM')
fig.show()
```

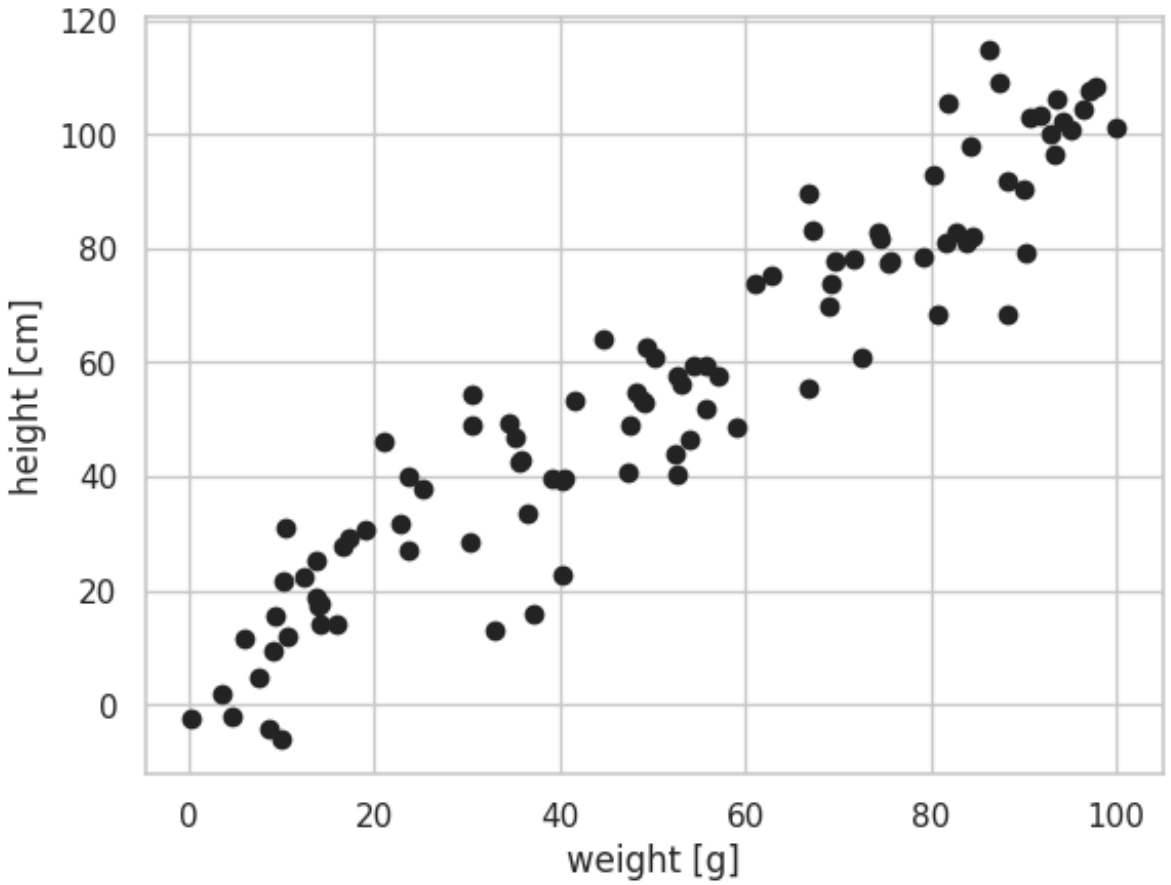
plot メソッドを複数回使うことで、複数の線グラフを描くことができる。グラフを描く際に色が指定されていない場合は、自動的に配色される。

散布図



- 連続量からなる多変量データ同士の相関や分布などを見るためのグラフ
- 縦軸および横軸は連続量
- 原点 (0, 0) が省略されているグラフを十分に注意すること
- 回帰直線との併用もよく見られる

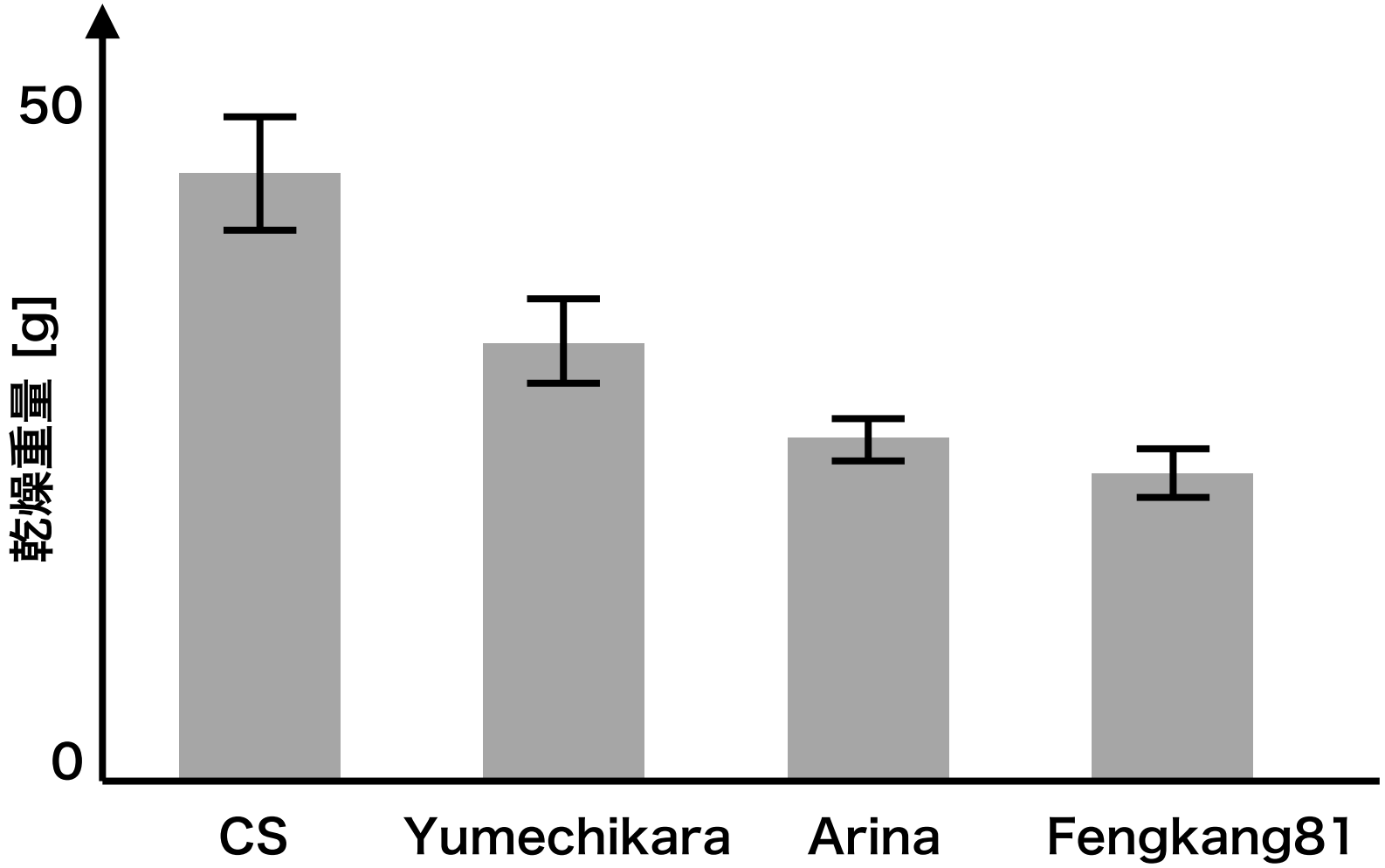
散布図




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

np.random.seed(2018)
x = np.random.uniform(0, 100, 100)
y = x + np.random.normal(5, 10, 100)

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.scatter(x, y)
ax.set_xlabel('weight [g]')
ax.set_ylabel('height [cm]')

fig.show()
```

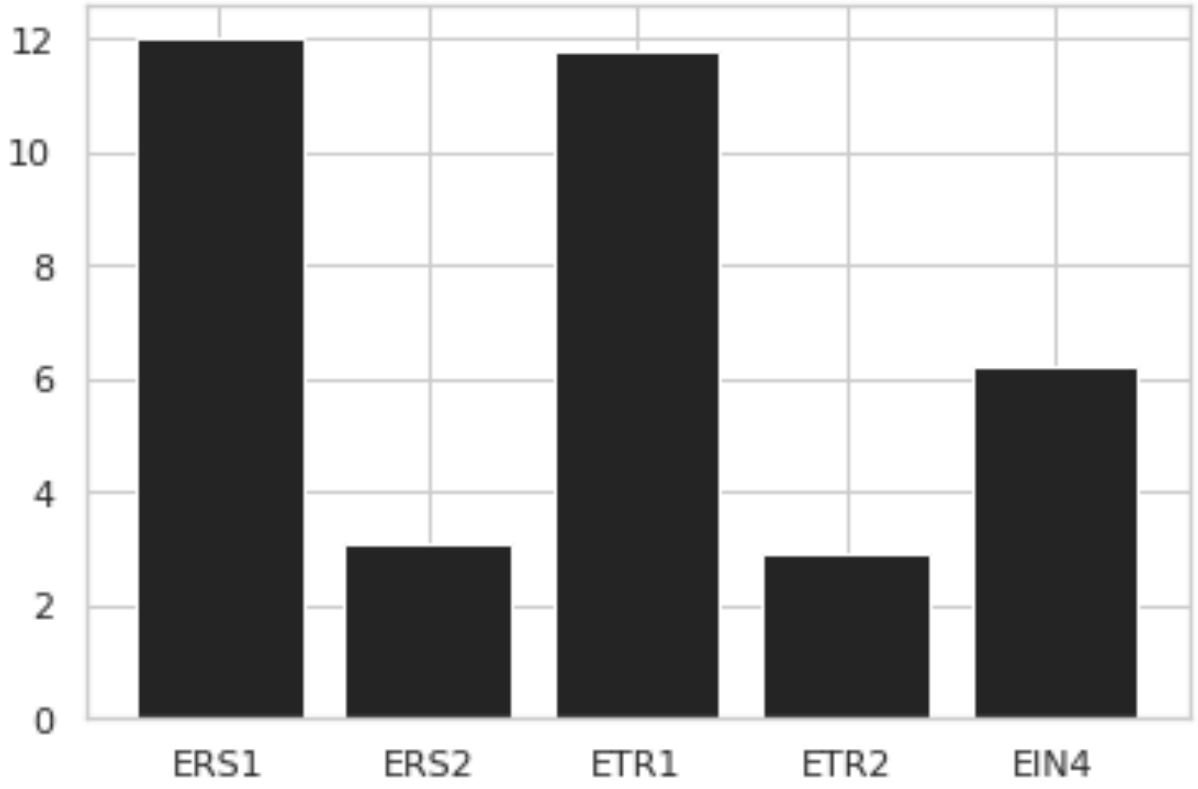
plot 関数は線グラフ、scatter 関数は点グラフなどのように、関数を変えることで様々なグラフを描けるようになる。

棒グラフ



- 離散データを視覚化するためのグラフ
- 横軸が連続量、縦軸が離散量、あるいはその逆
- 原点が省略されているグラフを十分に注意すること
- エラーバーとともに用いられることがある

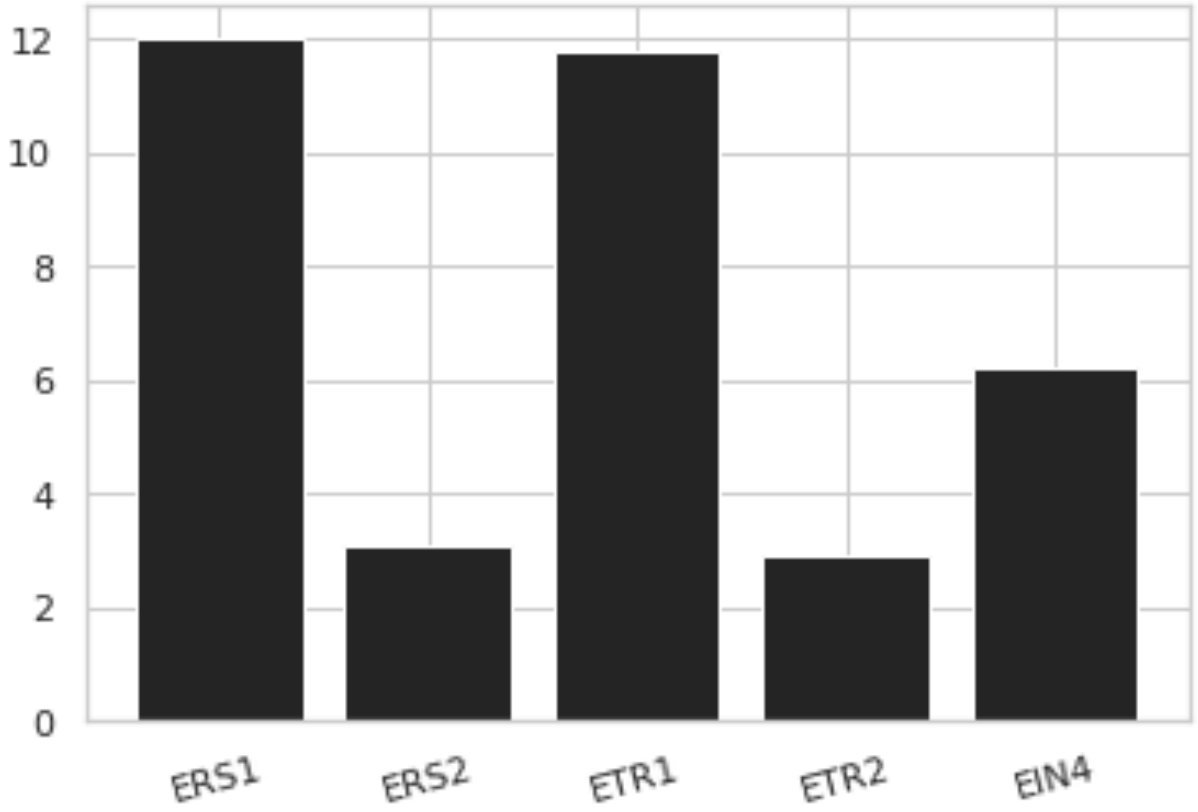
棒グラフ




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

x = np.array(['ERS1', 'ERS2', 'ETR1',
              'ETR2', 'EIN4'])
y = np.array([12.0, 3.1, 11.8, 2.9, 6.2])

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.bar(x, y)

fig.show()
```

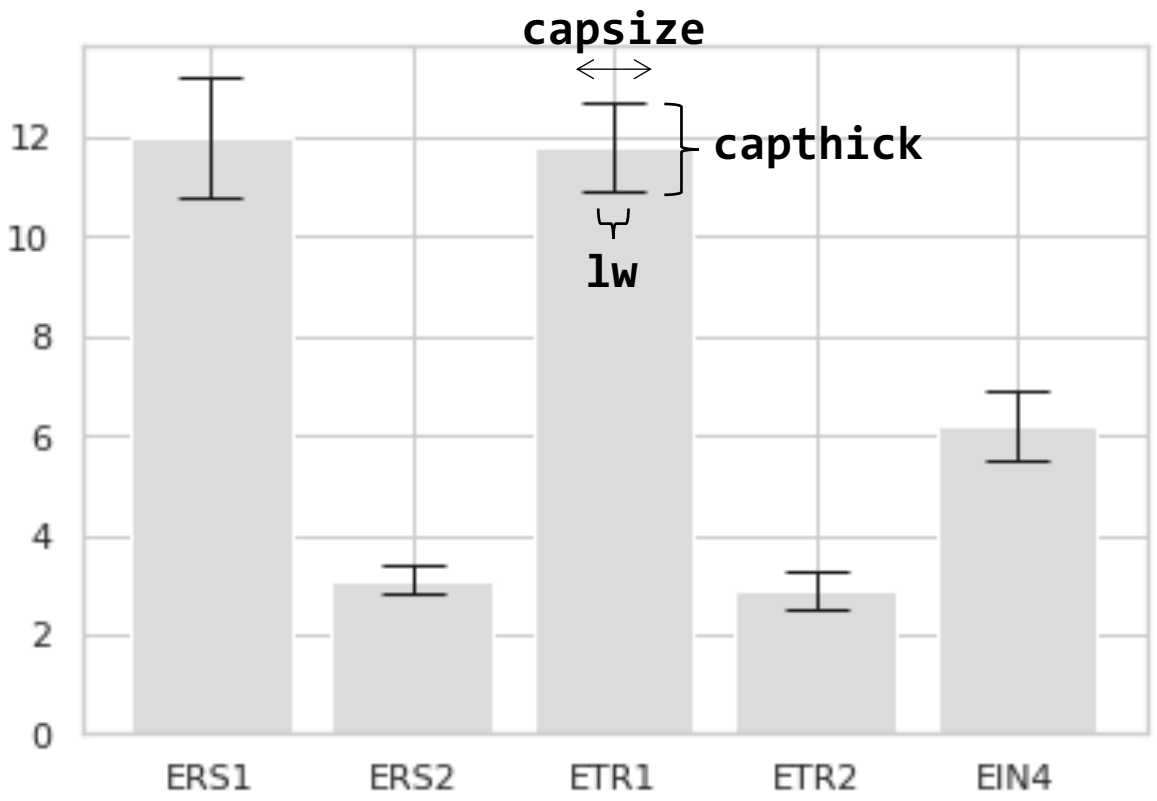
matplotlib のバージョンが古いと、横軸ラベルがアルファベット順に並べ替えられて描かれる場合がある。

棒グラフ



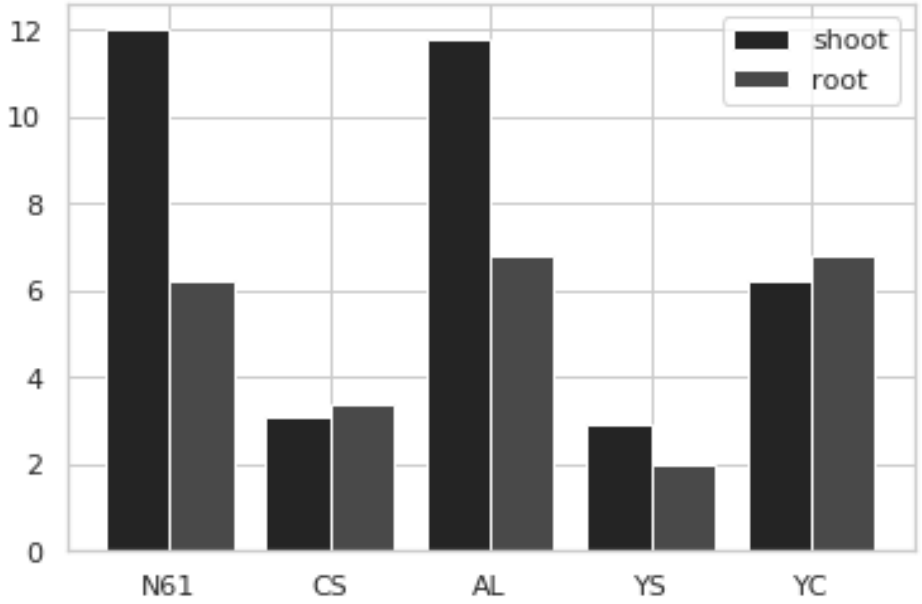

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array(['ERS1', 'ERS2', 'ETR1',
              'ETR2', 'EIN4'])
y = np.array([12.0, 3.1, 11.8, 2.9, 6.2])

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.bar(x, y)
ax.set_xticklabels(x, rotation=15)

fig.show()
```

棒グラフ




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


x = np.array(['ERS1', 'ERS2', 'ETR1',
              'ETR2', 'EIN4'])
y = np.array([12.0, 3.1, 11.8, 2.9, 6.2])
e = np.array([1.2, 0.3, 0.9, 0.4, 0.7])


fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
error_bar_set = dict(lw = 1, capthick = 1,
                     capsize = 10)
ax.bar(x, y, yerr = e,
       error_kw=error_bar_set)
fig.show()
```

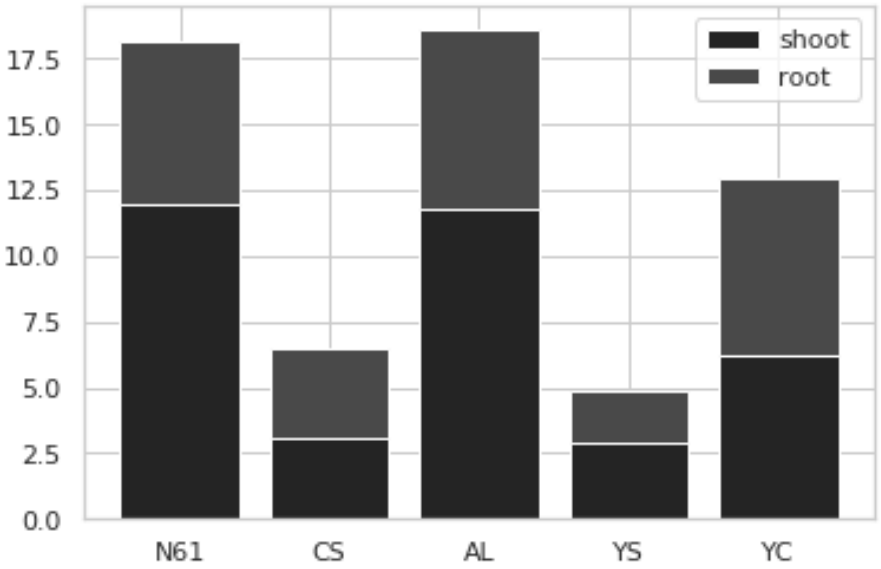
棒グラフ



横並びの棒グラフを描くとき、棒の x 座標および棒の幅をあらかじめ指定する必要がある。

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

xlabel = np.array(['N61', 'CS', 'AL', 'YS', 'YC'])
x = np.arange(len(xlabel))
y_shoot = np.array([12.0, 3.1, 11.8, 2.9, 6.2])
y_root = np.array([6.2, 3.4, 6.8, 2.0, 6.8])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.bar(x - 0.2, y_shoot, width=0.4, label='shoot')
ax.bar(x + 0.2, y_root, width=0.4, label='root')
ax.legend()
ax.set_xticks(x)
ax.set_xticklabels(xlabel)
fig.show()
```

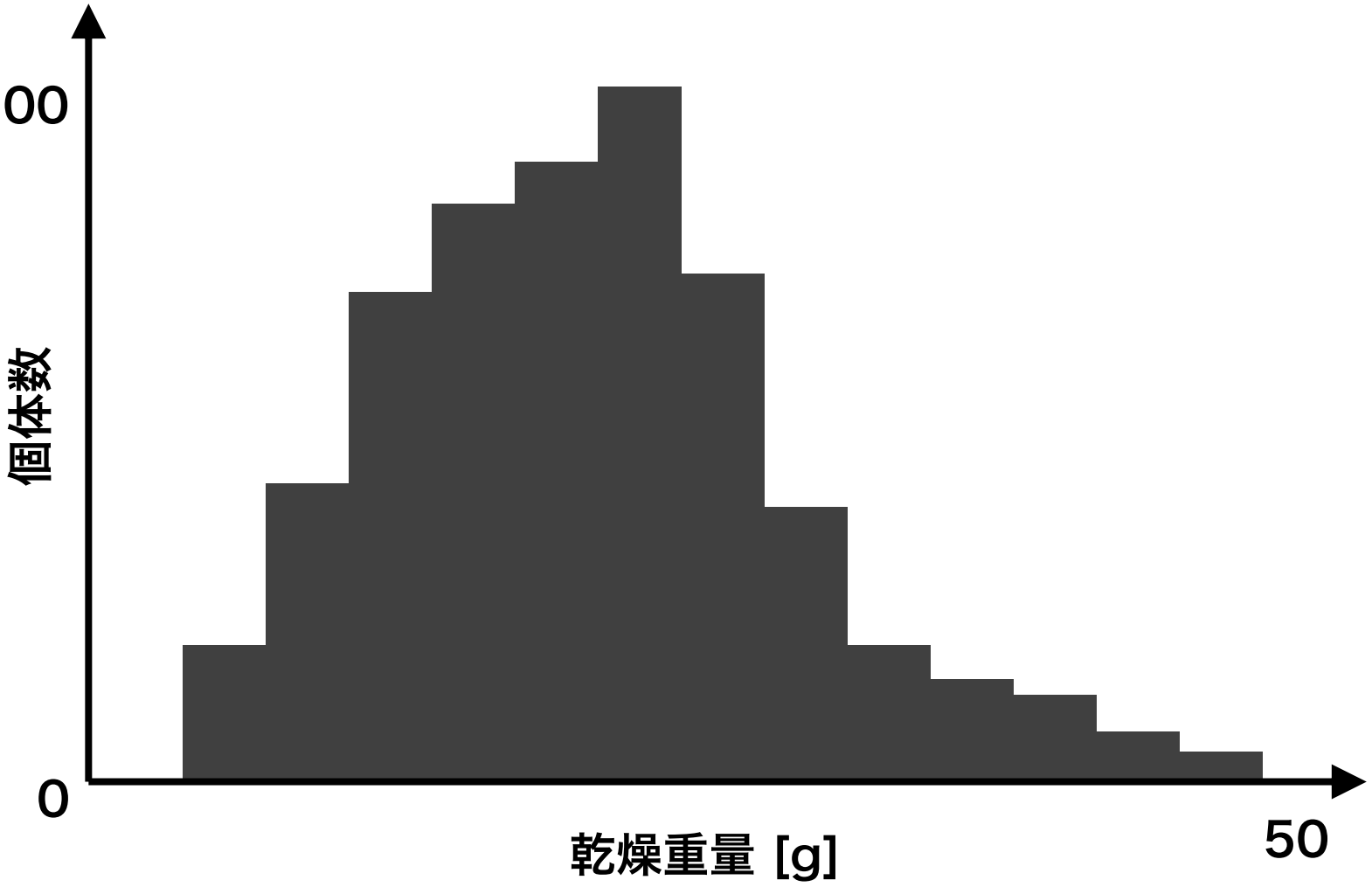
棒グラフ



積み重ね棒グラフの場合は、上の方の棒グラフを描くとき、そのスタート地点を調整する必要がある。

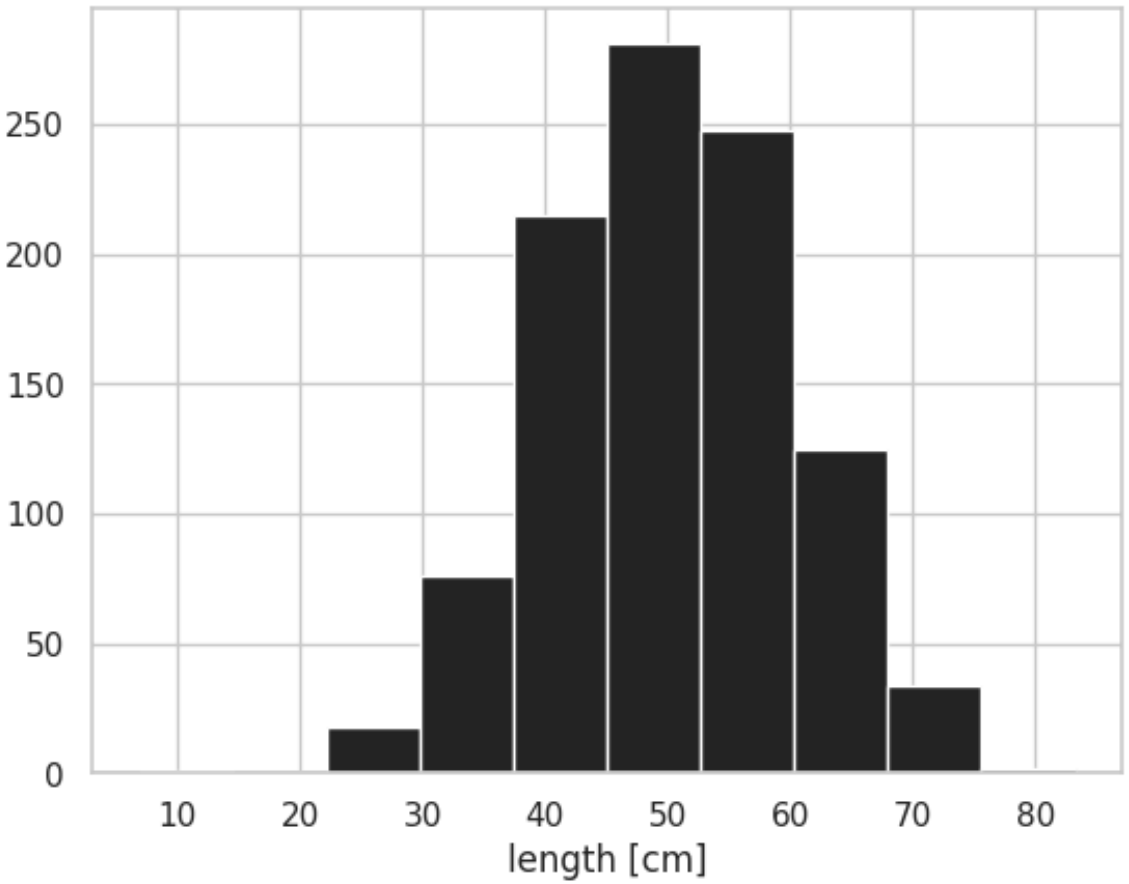
```
import matplotlib.pyplot as plt
import numpy as np

xlabel = np.array(['N61', 'CS', 'AL', 'YS', 'YC'])
x = np.arange(len(xlabel))
y_shoot = np.array([12.0, 3.1, 11.8, 2.9, 6.2])
y_root = np.array([6.2, 3.4, 6.8, 2.0, 6.8])
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.bar(x, y_shoot, label='shoot')
ax.bar(x, y_root, label='root', bottom=y_shoot)
ax.legend()
ax.set_xticks(x)
ax.set_xticklabels(xlabel)
fig.show()
```

ヒストグラム



- 1変量の連続値データを視覚化するためのグラフ
- 横軸が連続量であり、縦軸は頻度・個数または確率である
- 横幅は恣意的（経験的）に決められることもあれば、スタージェスの公式などで決めることもある

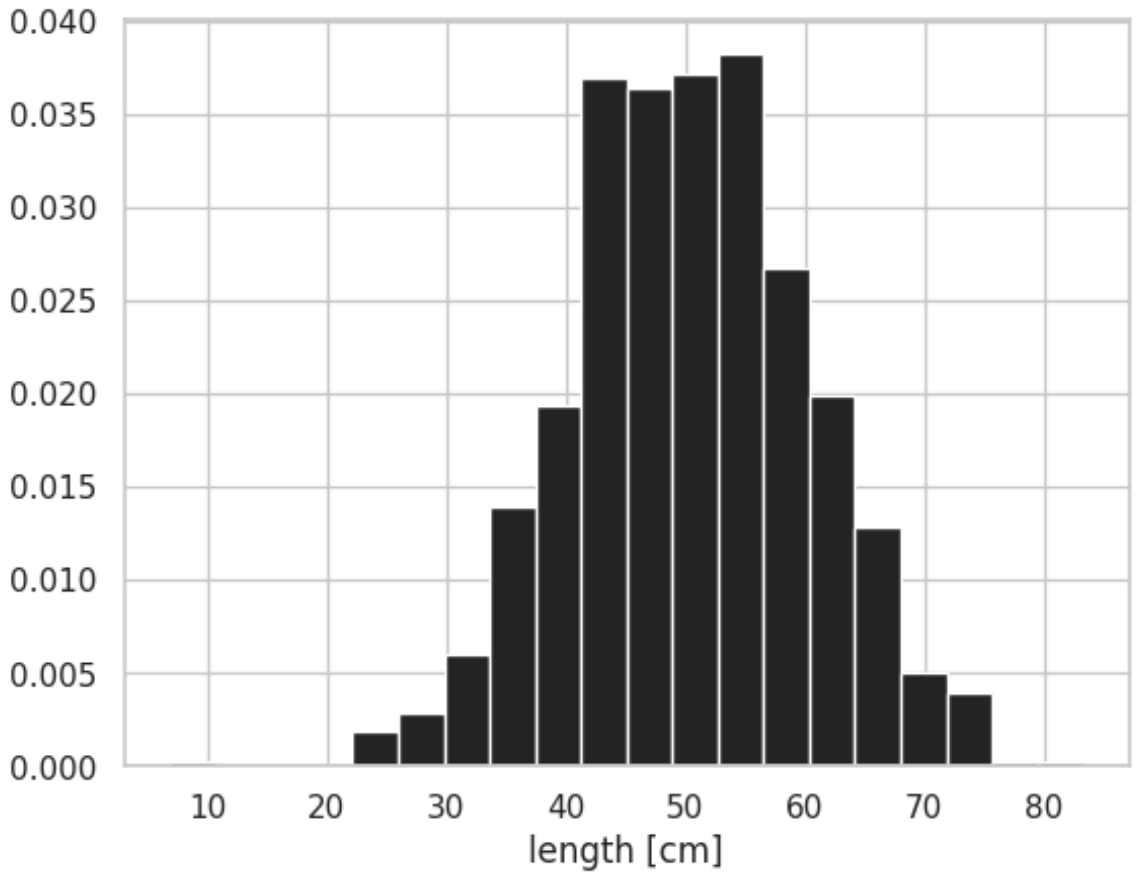
ヒストグラム



matplotlib デフォルトでは、ヒストグラムの横幅はスタージェスまたはフリードマン・ダイアコニスの公式により算出される。

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

np.random.seed(2018)

x = np.random.normal(50, 10, 1000)

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.hist(x)

ax.set_xlabel('length [cm]')

fig.show()
```

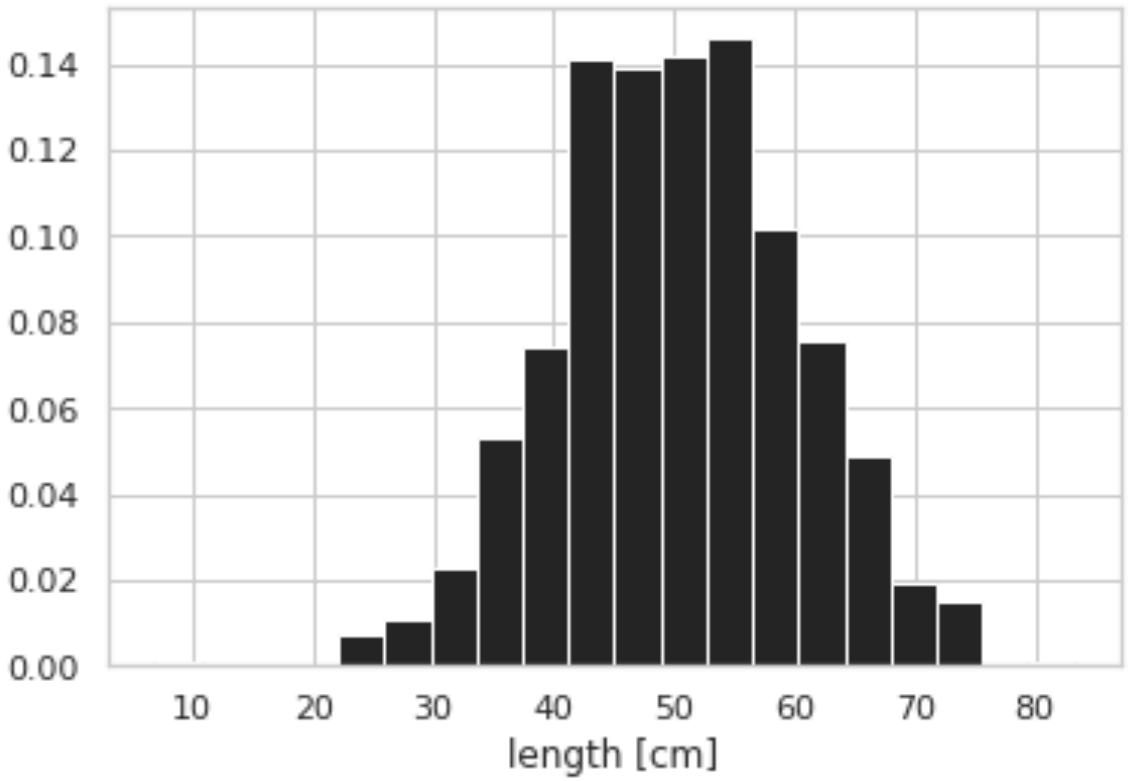
ヒストグラム



density=True を指定したとき、すべての棒を足したとき、その長さが 1 になる。面積が 1 になるわけではないことに注意。

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

np.random.seed(2018)

x = np.random.normal(50, 10, 1000)

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.hist(x, bins=20, density=True)

ax.set_xlabel('length [cm]')

fig.show()
```

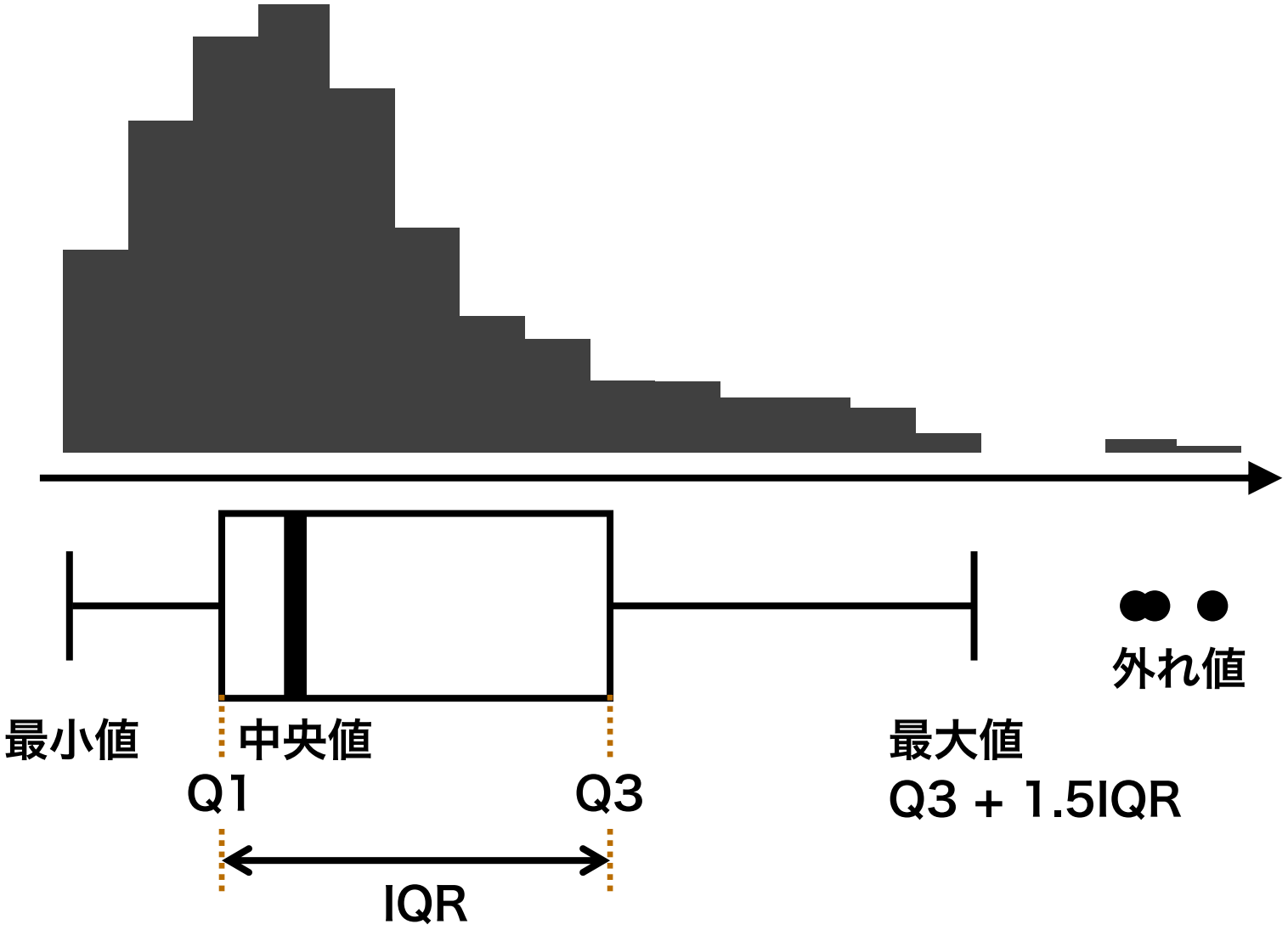
ヒストグラム



weights を指定して面積が 1 になるように調整できる。

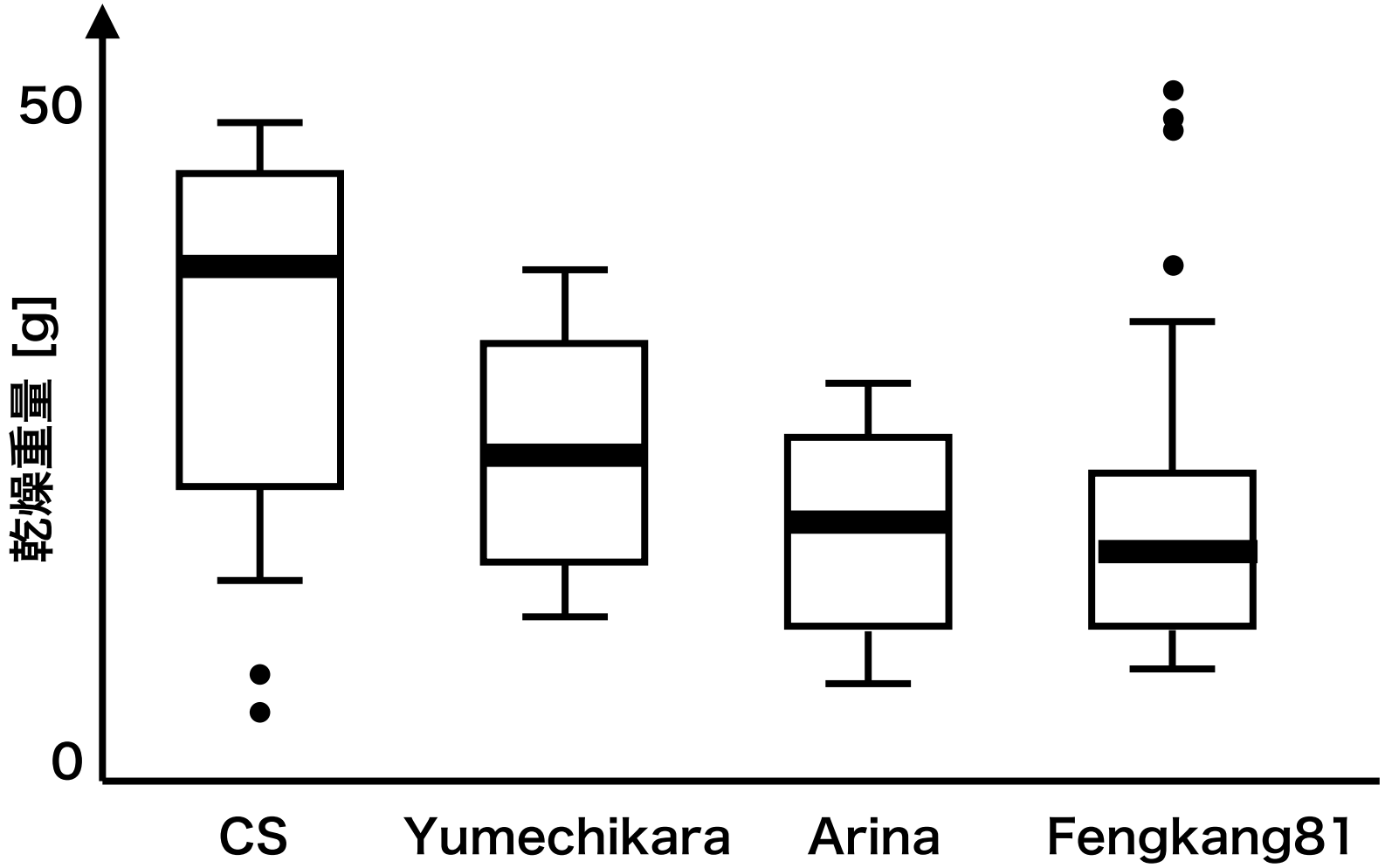
```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(2018)

x = np.random.normal(50, 10, 1000)

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.hist(x, bins=20,
        weights=np.ones(len(x))/len(x))
ax.set_xlabel('length [cm]')

fig.show()
```

ボックスプロット



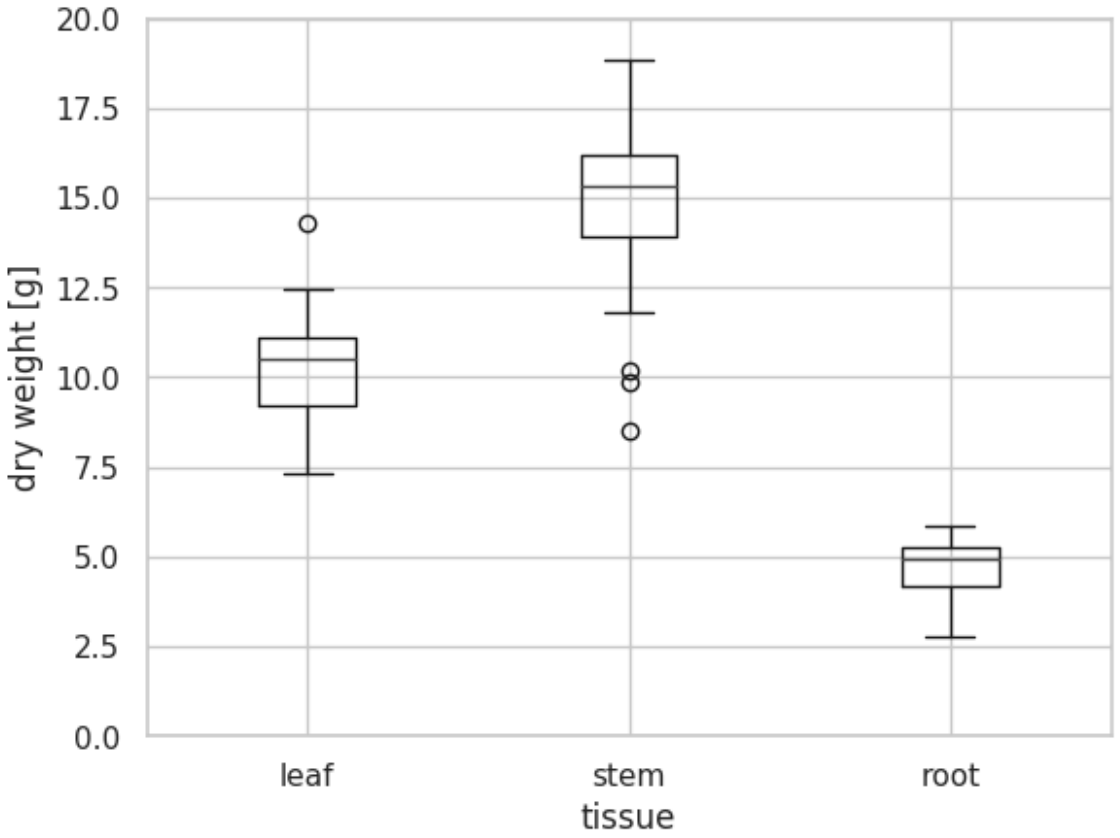
- 1変量の連続値データを視覚化するためのグラフ
- 最大値、最小値、第1四分位数（Q1）、中央値、第3四分位数（Q3）などを簡単に確認できる
- Q3±1.5(Q3-Q1) 範囲外にあるデータは外れ値と定義される

ボックスプロット



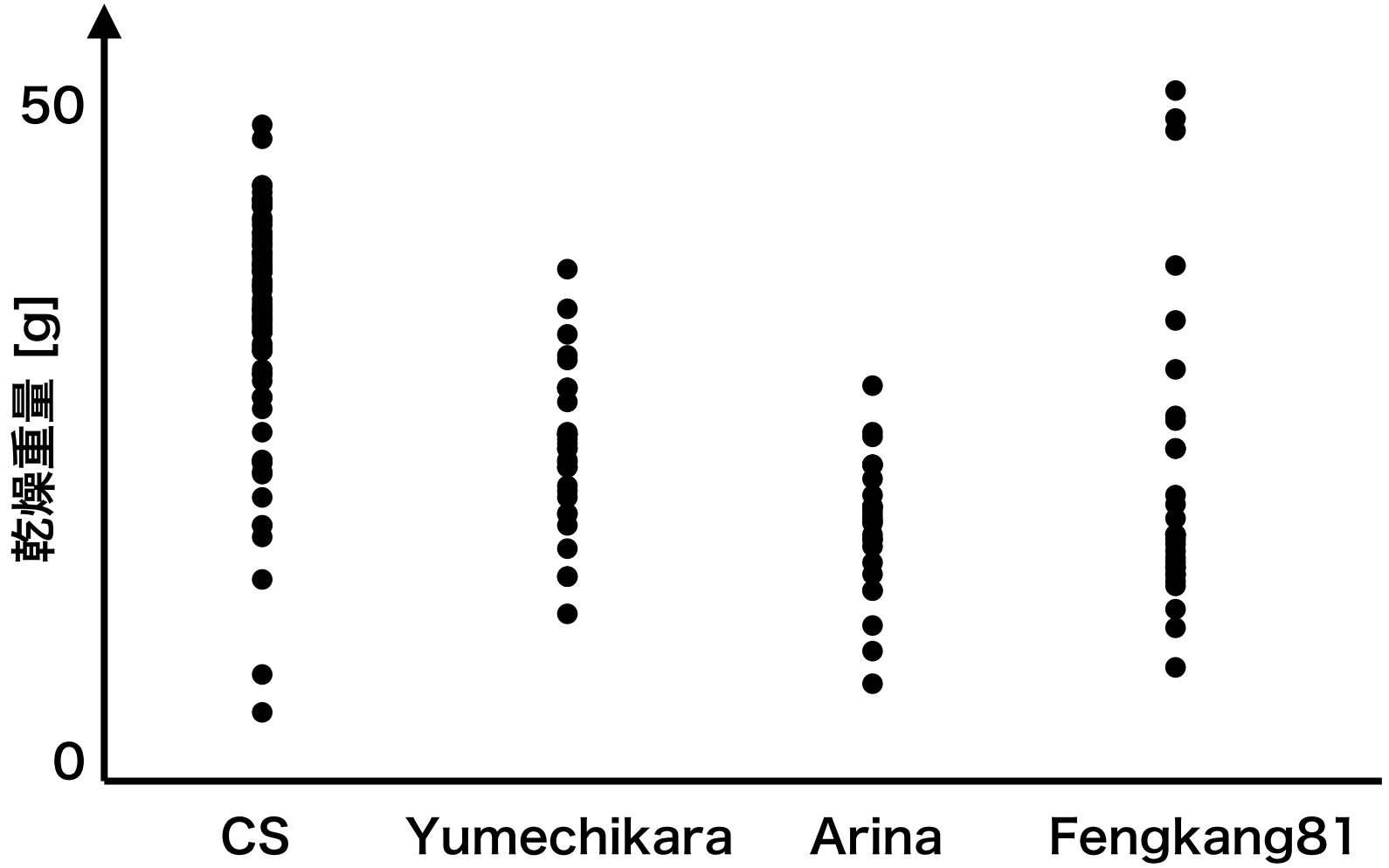
- 1変量の連続値データを視覚化するためのグラフ
- 最大値、最小値、第1四分位数（Q1）、中央値、第3四分位数（Q3）などを簡単に確認できる
- Q3±1.5(Q3-Q1) 範囲外にあるデータは外れ値と定義される
- 複数カテゴリのデータの分布を確認するときに便利

ヒストグラム



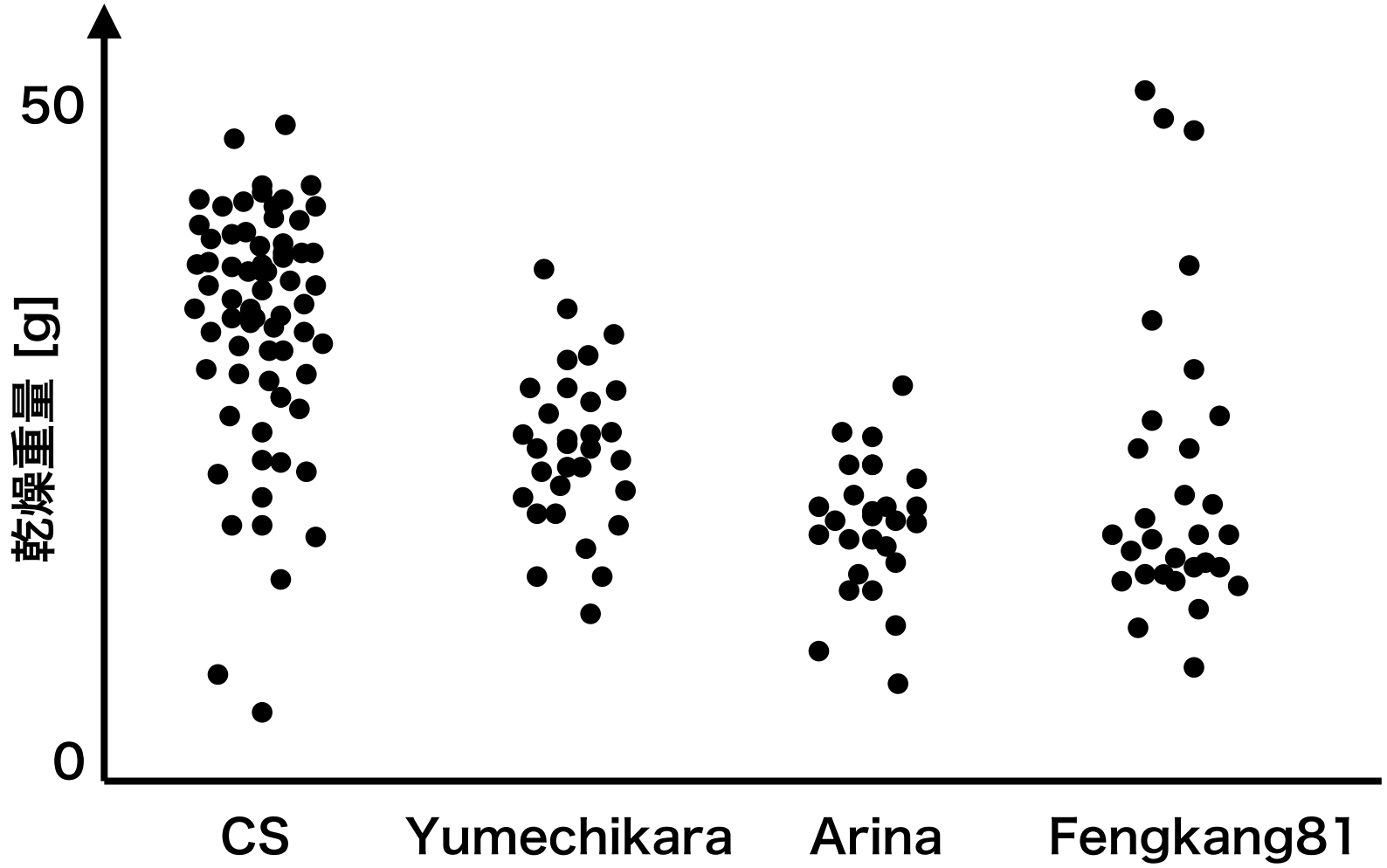

```
import numpy as  np
import matplotlib.pyplot as plt
np.random.seed(2018)
x1 = np.random.normal(10, 2, 20)
x2 = np.random.normal(15, 3, 20)
x3 = np.random.normal(5, 1, 20)
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.boxplot([x1, x2, x3],
           labels=['leaf', 'stem', 'root'])
ax.set_xlabel('tissue')
ax.set_ylabel('dry weight [g]')
ax.set_ylim(0, 20)
fig.show()
```

jitter



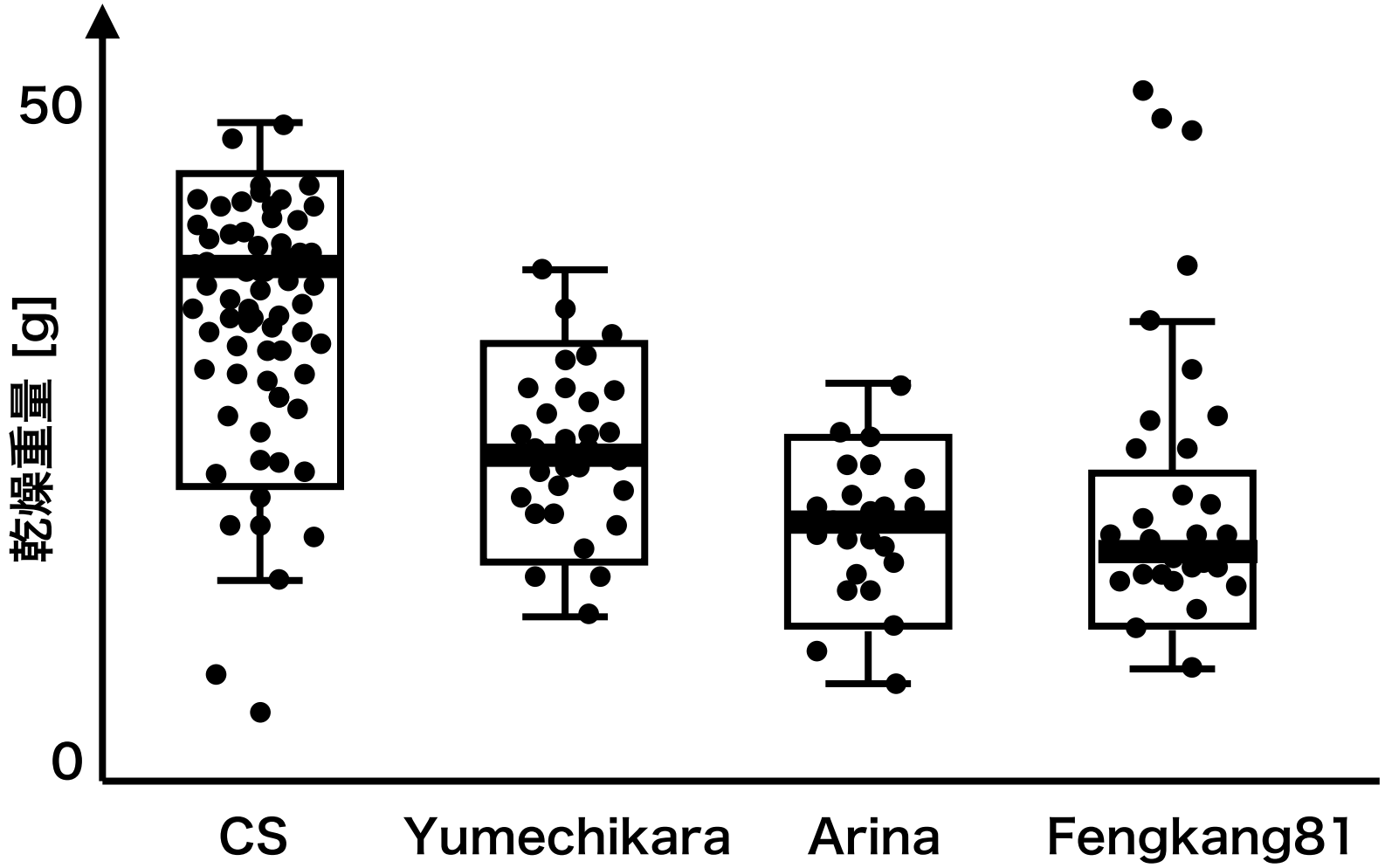
- 複数カテゴリの1変量連続値データの分布を確認ためのグラフ
- 実際の値をプロットすると、データが多いところが重なるため、点を左右にランダムにずらしてプロットする

jitter



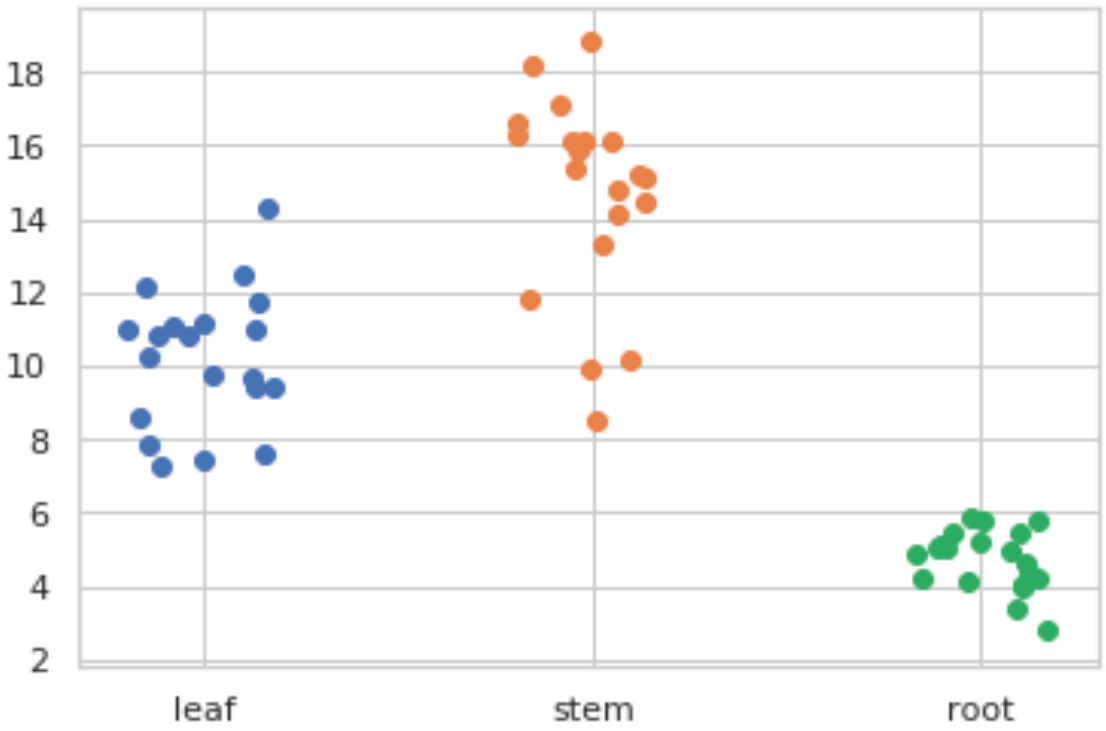
- 複数カテゴリの1変量連続値データの分布を確認ためのグラフ
- 実際の値をプロットすると、データが多いところが重なるため、点を左右にランダムにずらしてプロットする

jitter



- 複数カテゴリの1変量連続値データの分布を確認ためのグラフ
- 実際の値をプロットすると、データが多いところが重なるため、点を左右にランダムにずらしてプロットする
- ボックスプロットと合わせて使われる場合もある

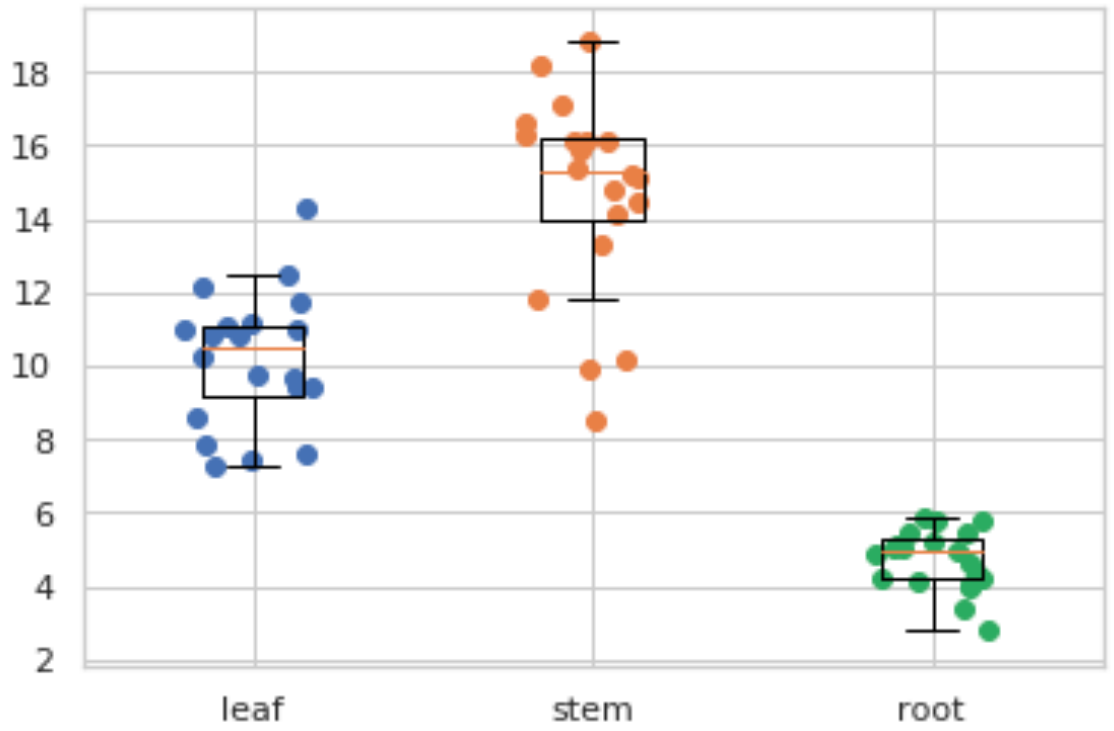
jitter



y1、y2、y3 のデータの y 座標をデータの値のそのままにして、x 座標をそれぞれ 1、2、3 から少し左右にずらした値にする。

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(2018)
y1 = np.random.normal(10, 2, 20)
y2 = np.random.normal(15, 3, 20)
y3 = np.random.normal(5, 1, 20)

x1 = 1 + np.random.uniform(-0.2, 0.2, len(y1))
x2 = 2 + np.random.uniform(-0.2, 0.2, len(y2))
x3 = 3 + np.random.uniform(-0.2, 0.2, len(y3))

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

ax.scatter(x1, y1)
ax.scatter(x2, y2)
ax.scatter(x3, y3)

ax.set_xticks([1, 2, 3])
ax.set_xticklabels(['leaf', 'stem', 'root'])
fig.show()
```

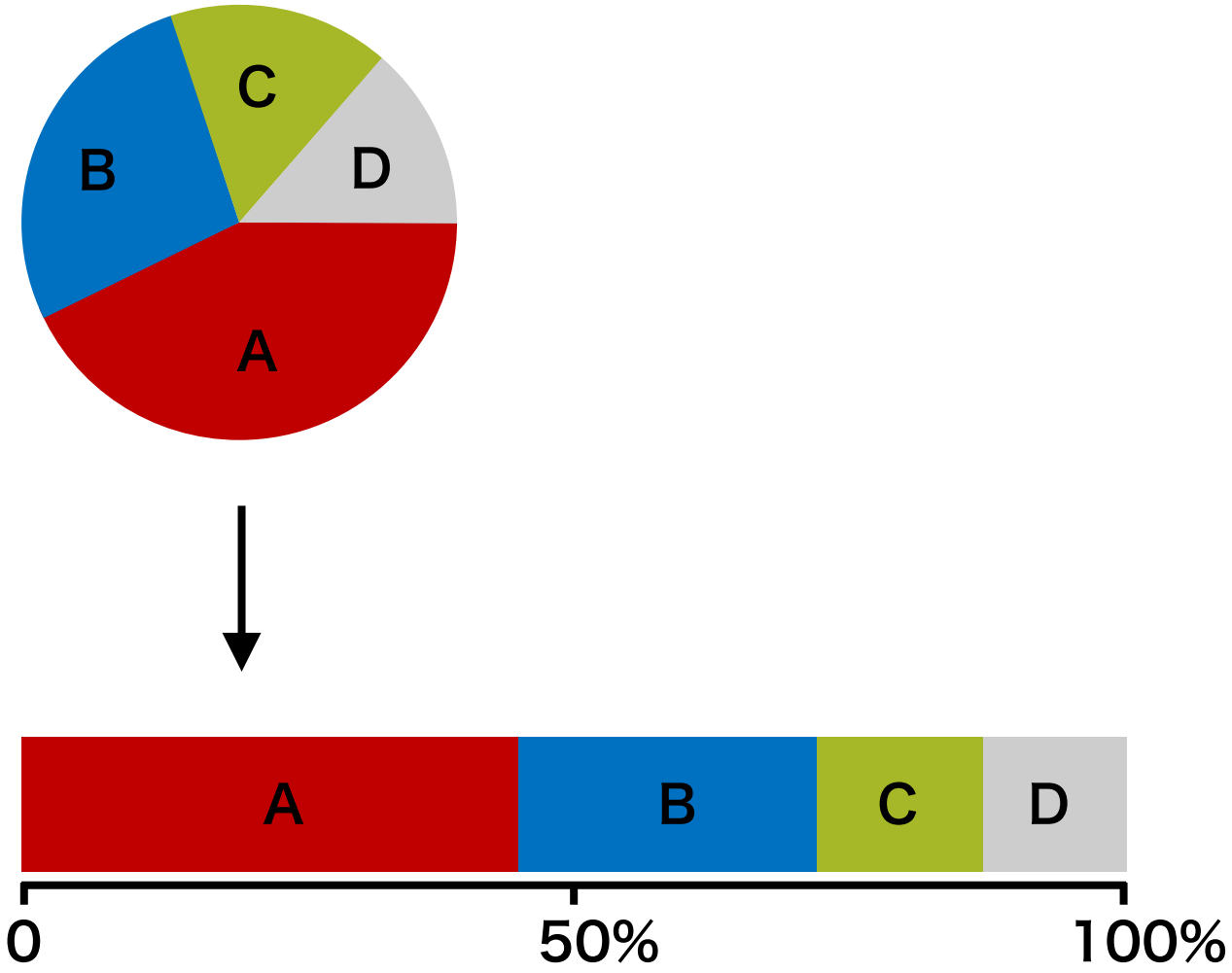
jitter + ボックスプロット




```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(2018)
y1 = np.random.normal(10, 2, 20)
y2 = np.random.normal(15, 3, 20)
y3 = np.random.normal(5, 1, 20)

x1 = 1 + np.random.uniform(-0.2, 0.2, len(y1))
x2 = 2 + np.random.uniform(-0.2, 0.2, len(y2))
x3 = 3 + np.random.uniform(-0.2, 0.2, len(y3))

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

ax.scatter(x1, y1)
ax.scatter(x2, y2)
ax.scatter(x3, y3)

ax.boxplot([y1, y2, y3],
           labels=['leaf', 'stem', 'root'],
           showfliers=False)
fig.show()
```

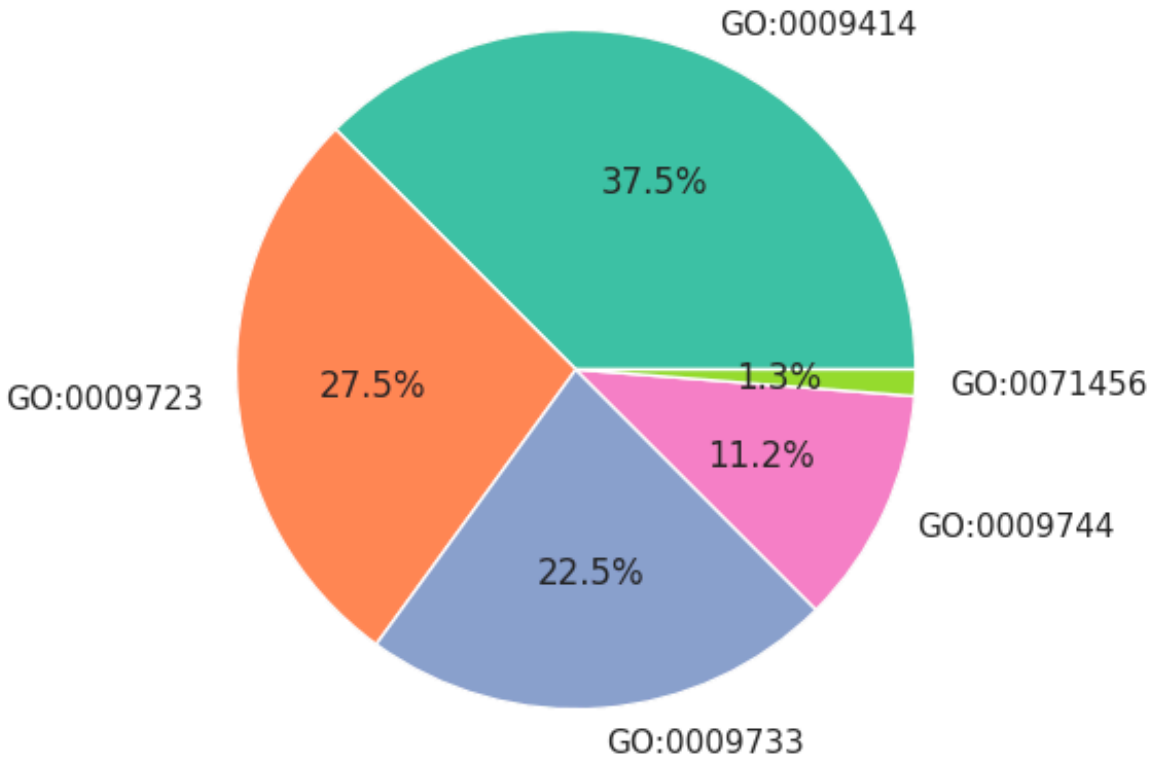
showfliers=False を指定することで、boxplot メソッドは外れ値を描かなくなる。

円グラフ



- 円グラフは誤解されやすい、円グラフを用いる前に帯状グラフで代用できないかを確認すること
- 3D円グラフは基本的に人を騙すようのグラフであることに注意

円グラフ



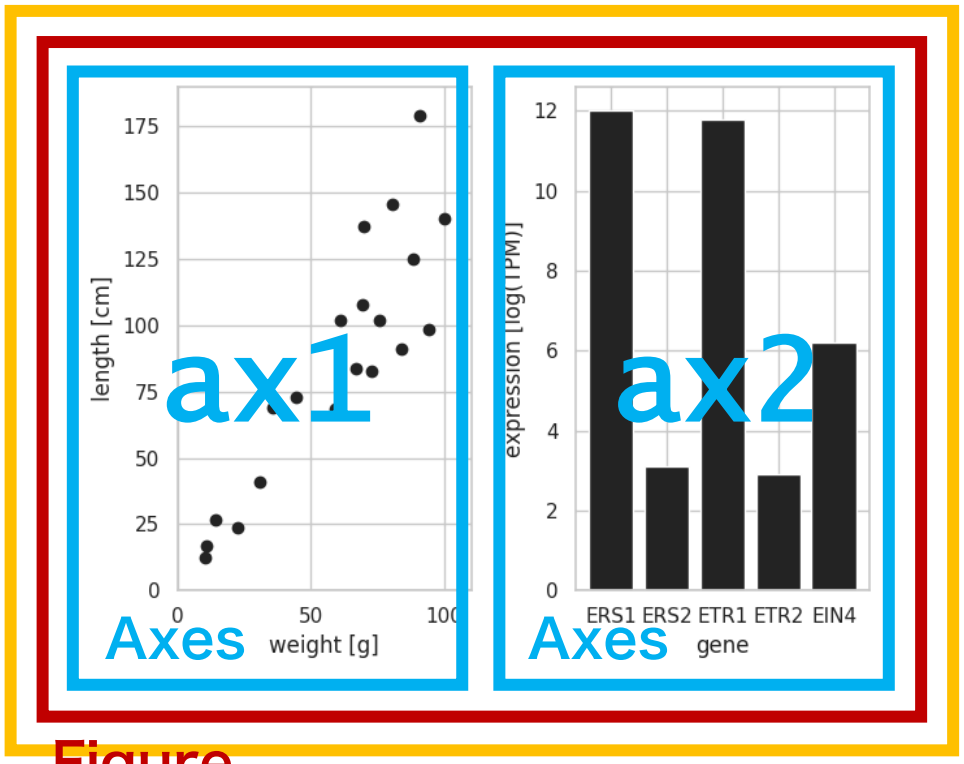

```
import matplotlib.pyplot as plt
import numpy as np

x = ['GO:0009414', 'GO:0009723',
     'GO:0009733', 'GO:0009744',
     'GO:0071456']
y = np.array([300, 220, 180, 90, 10])

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.pie(y, labels=x, autopct="%1.1f%%")
ax.axis('equal')

fig.show()
```

useful references

- matplotlib gallery
  https://matplotlib.org/gallery.html
- seaborn example gallery
  https://seaborn.pydata.org/examples/index.html
- Python Graph Gallery
  https://python-graph-gallery.com/
  https://stats.biopapyrus.jp/python/python-graph-gallery.html

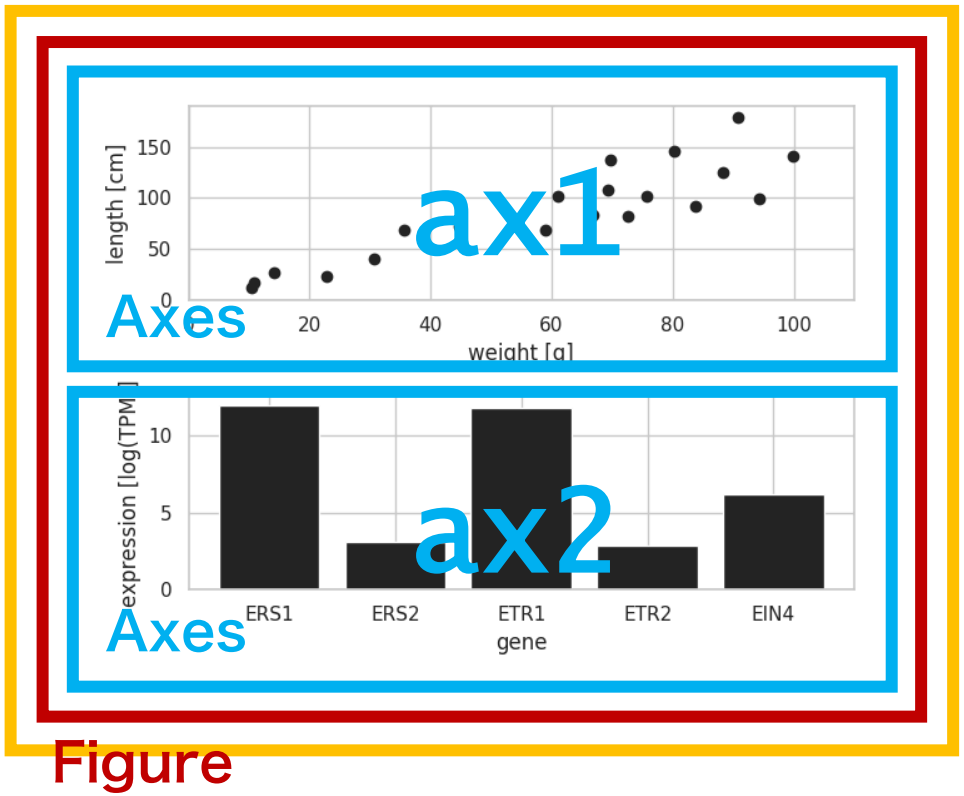
複数グラフ

画面分割



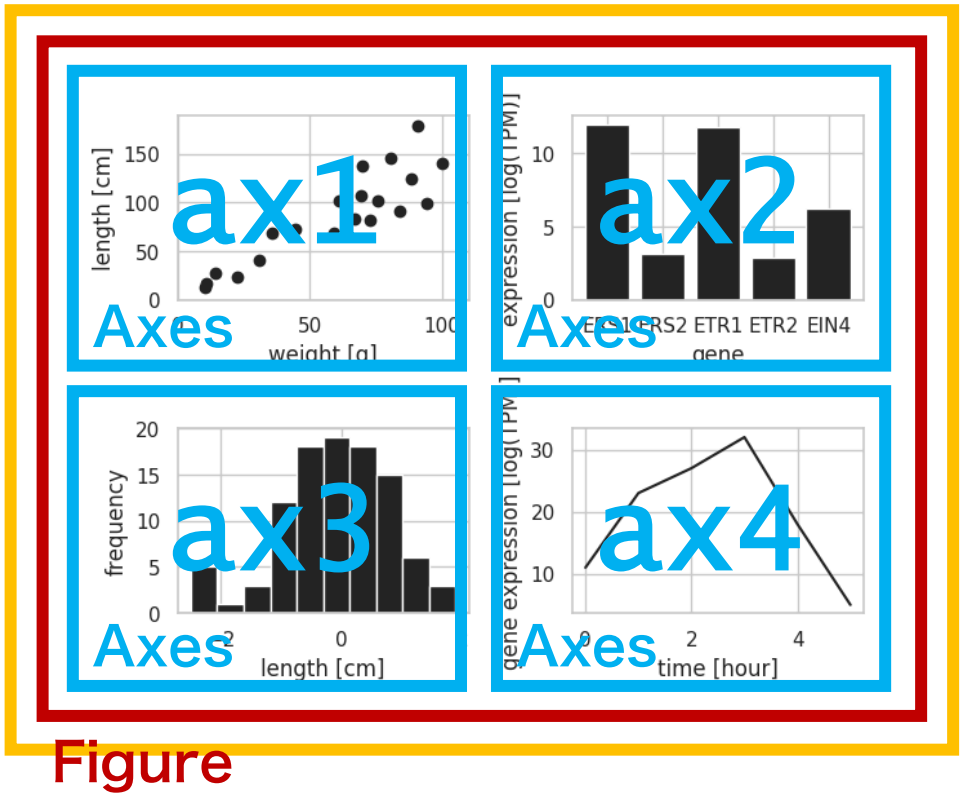

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
fig = plt.figure()
x1 = np.random.uniform(0, 100, 20)
y1 = x1 * np.random.uniform(1, 2, 20)
ax1 = fig.add_subplot(1, 2, 1)
ax1.scatter(x1, y1)
x2 = np.array(['ERS1', 'ERS2', 'ETR1', 'ETR2', 'EIN4'])
y2 = np.array([12.0, 3.1, 11.8, 2.9, 6.2])
x2_position = np.arange(len(x2))
ax2 = fig.add_subplot(1, 2, 2)
ax2.bar(x2_position, y2, tick_label=x2)
fig.show()
```

画面分割




```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
fig = plt.figure()
x1 = np.random.uniform(0, 100, 20)
y1 = x1 * np.random.uniform(1, 2, 20)
ax1 = fig.add_subplot(2, 1, 1)
ax1.scatter(x1, y1)
x2 = np.array(['ERS1', 'ERS2', 'ETR1', 'ETR2', 'EIN4'])
y2 = np.array([12.0, 3.1, 11.8, 2.9, 6.2])
x2_position = np.arange(len(x2))
ax2 = fig.add_subplot(2, 1, 2)
ax2.bar(x2_position, y2, tick_label=x2)
fig.show()
```

画面分割




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


fig = plt.figure()
ax1 = fig.add_subplot(2, 2, 1)
ax1.scatter(x1, y1)
ax2 = fig.add_subplot(2, 2, 2)
ax2.bar(x2, y2)
ax3 = fig.add_subplot(2, 2, 3)
ax3.hist(x3)
ax4 = fig.add_subplot(2, 2, 4)
ax4.plot(x4, y4)
fig.show()
```

視覚化 × seaborn

seaborn

seaborn を利用したグラフ作成は、matplotlib と同様な手順で作成できる。

seaborn のインポート

seaborn クラスをインポートして、利用可能な状態にする。

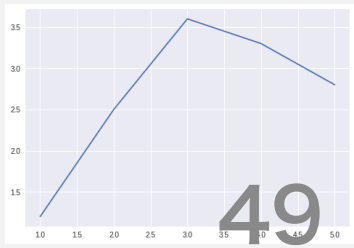
seaborn.set の実行

seaborn.set メソッドを実行して、グラフ描画パラメータなどを seaborn 風に書き換える。

グラフの作成と表示

matplotlib.pyplot のメソッドをそのまま使用してグラフの作成と表示を行う。

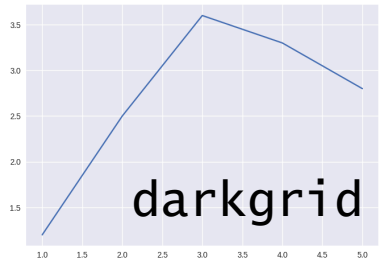
```
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
sns.set()



x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
fig.show()
```

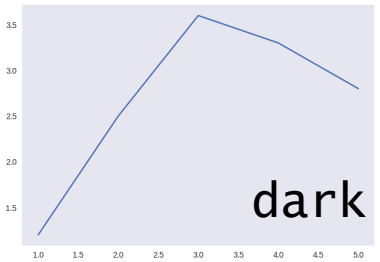


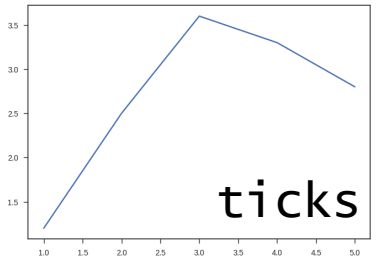
seaborn スタイルシート

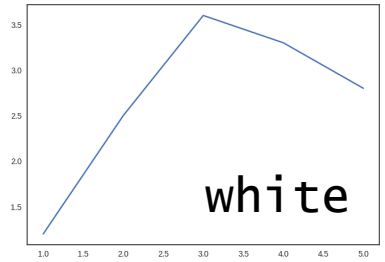
グラフのスタイルを設定する。

seaborn ではスタイルシートが用意されている。スタイルシートを書き換えることで、グラフの描画エリアのスタイルを変更することができる。









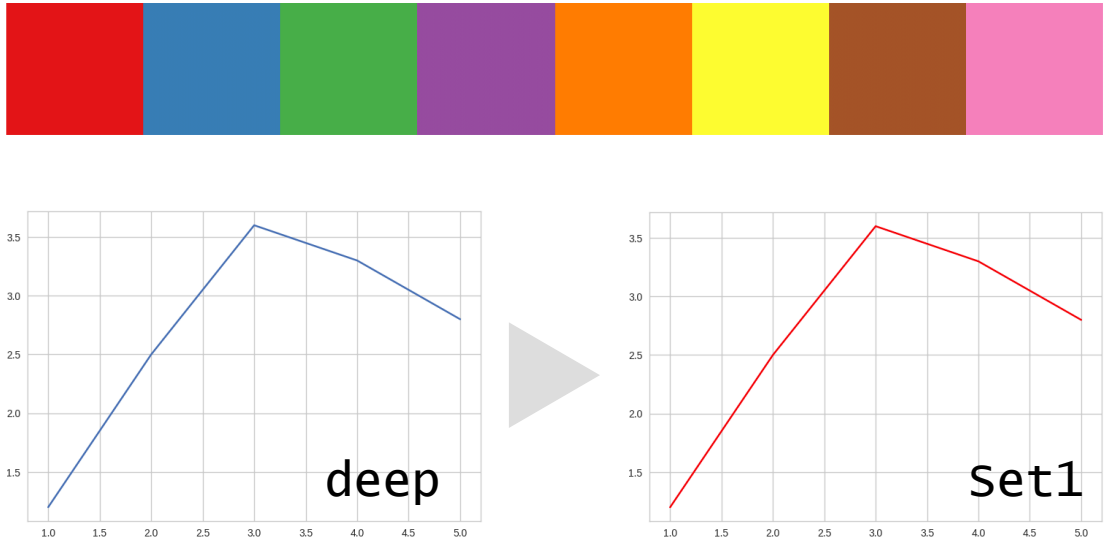

```
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
sns.set()
sns.set_style('whitegrid')
sns.set_palette('Set1')

x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
fig.show()
```

seaborn カラーパレット

カラーパレットを設定する。

あらかじめ決められた色の組み合わせ。カラーパレットを書き換えることで、グラフ全体の配色パターンを簡単に変更できる。

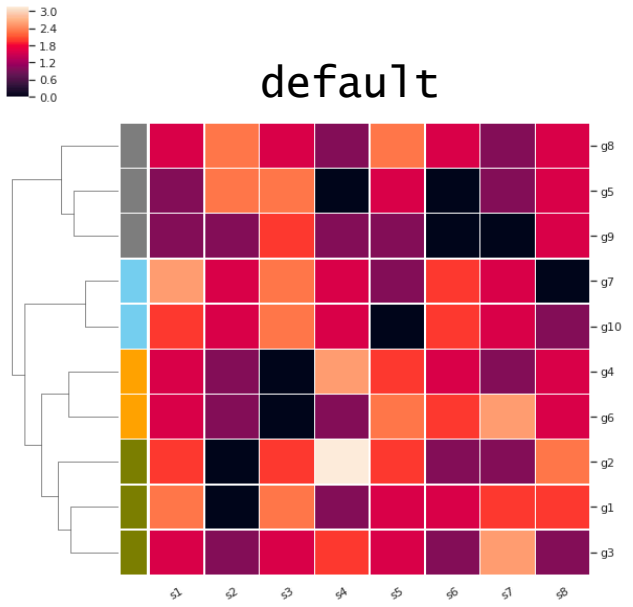
Set1

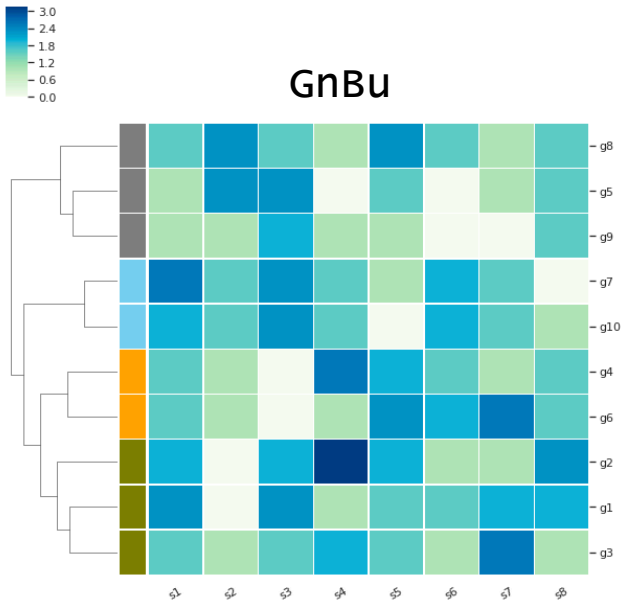


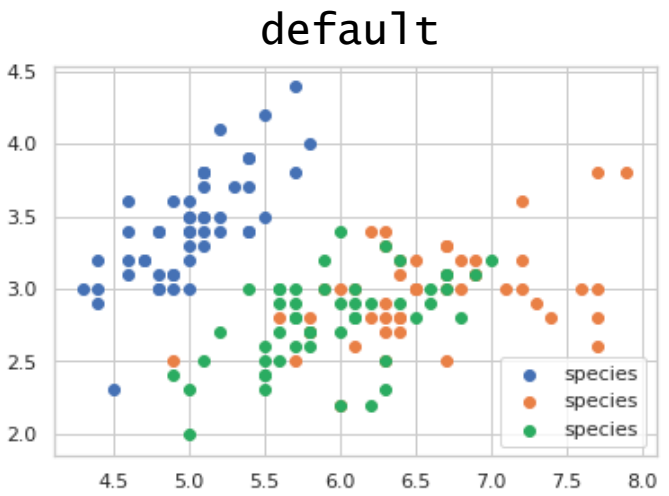

```
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
sns.set()
sns.set_style('whitegrid')
sns.set_palette('Set1')

x = np.array([1.0, 2.0, 3.0, 4.0, 5.0])
y = np.array([1.2, 2.5, 3.4, 3.3, 2.8])

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.plot(x, y)
fig.show()
```

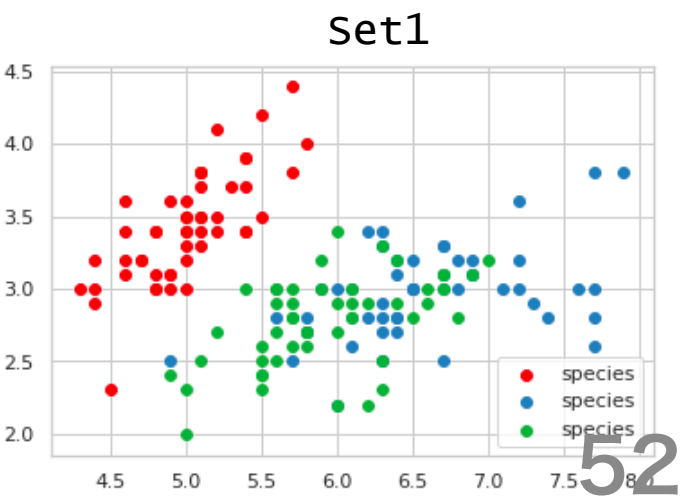
seaborn カラーパレット

あらかじめ決められた色の組み合わせ。カラーパレットを書き換えることで、グラフ全体の配色パターンを簡単に変更できる。









seaborn カラーパレット

あらかじめ決められた色の組み合わせ。カラーパレットを書き換えることで、グラフ全体の配色パターンを簡単に変更できる。seaborn のデフォルトのカラーパレットは deep となっている。set_palette メソッドを使うことで、カラーパレットの変更を行うことができる。