農学生命情報科学特論 I

ICT や IoT 等の先端技術を活用し、効率よく高品質生産を可能にするスマート農業への取り組みは世界的に進められています。その基礎を支えている技術の一つがプログラミング言語。なかでも、習得しやすくかつ応用範囲の広い Python がとくに注目されています。本科目では、農学や分子生物学などの分野で利用される Python の最新事例を紹介しながら、Python の基礎文法の講義を行います。

孫　建強　https://aabbdd.jp/
農研機構・農業情報研究センター

June
08 17:15–20:30

基本構文

第 1 回目の授業では、プログラミング言語の基本であるデータ構造とアルゴリズムを簡単に紹介してから、Python の基本構文を紹介する。Python のスカラー、リスト、ディクショナリ、条件構文と繰り返し構文を取り上げる。

June
15 17:15–20:30

文字列処理

バイオインフォマティックスの分野において、塩基配列やアミノ酸配列などの文字列からなるデータを扱うことが多い。第 2 回目の授業では、Python を利用した文字列処理を紹介し、FASTA や GFF などのファイルから情報を抽出する方法を取り上げる。

June
22 17:15–20:30

データ解析

Python で CSV や TSV ファイルに保存された数値データを扱うときに、NumPy や Pandas ライブラリーを使用する。第 3 回目の授業では、NumPy や Pandas の機能を紹介し、ベクトルや行列のデータ処理を中心に取り上げる。

June
29 17:15–20:30

データ可視化

Python で数値データを可視化するときに matplotlib ライブラリーを使用する。第 4 回目の授業では、matplotlib の基本的な使い方を紹介し、全回の授業を復習しながらデータの可視化を行う。

成績評価

- 授業の始めに出席確認を 1 回だけを行う。出席 1 回につき 1 点を与える。
- 出席確認後、レポート課題を示す。毎回 3 問出題し、授業全体を通して合計 12 問出題する。1 問 8 点とする。
- 授業を聞かなくても、レポート課題を解けそうであれば、遠慮なく退室していただいて構いません。

レポート課題の提出方法

- レポート課題を Jupyter Notebook / Jupyter Lab 上で解き、そのファイル（.ipynb）を メールで提出する。

  ✉ report＠aabbdd.jp

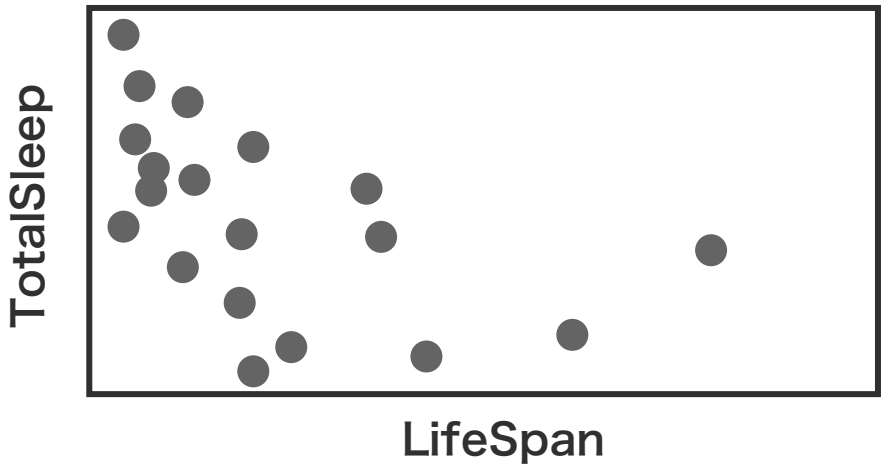
- 注意事項
  - メールの件名を「特論課題4」としてください。
  - メールの本文に名前を書いてください。
  - ファイルの名前を学生証番号（例：310000A.ipynb）としてください。
  - ファイル中に、名前・所属・研究内容などの個人情報・機密情報を書かないでください。個人情報・機密情報を含むレポートを零点とする。

レポート課題（第 4 回）

問題 1

sleep_in_mammals.txt には哺乳類の睡眠時間や寿命のデータが保存されている。このデータを分析し、睡眠時間（TotalSleep）と寿命（LifeSpan）の関係をグラフで示せ。

解答例



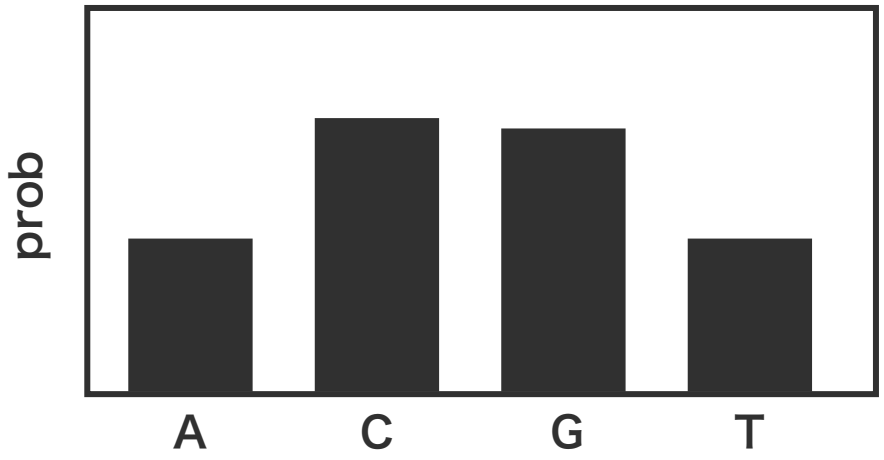
https://aabbdd.jp/notes/data/sleep_in_mammals.txt

レポート課題（第 4 回）

問題 2

ft.fa には FT 遺伝子の塩基配列が保存されている。この配列を読み込み、塩基 A、C、G、T の出現確率を棒グラフで示せ。

解答例



https://aabbdd.jp/notes/data/ft.fa

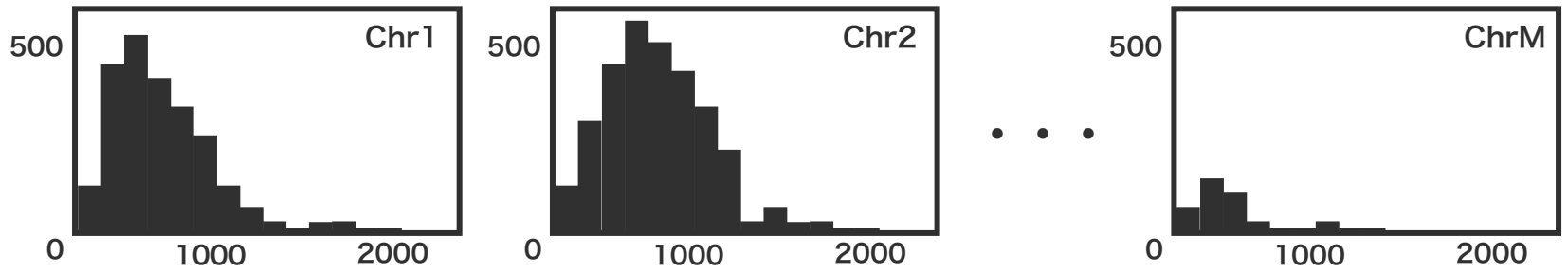
レポート課題（第 4 回）

問題 3

GFF3 フォーマットのシロイヌナズナの遺伝子アノテーション TAIR10_GFF3_genes.gff を読み込み、遺伝子の長さの分布を染色体ごとにグラフで示せ。

```
Chr1   TAIR10   gene      3631   5899   .   +   .   ID=AT1G01010;Note=protein_coding_gene;Name=AT1G01010
Chr1   TAIR10   mRNA      3631   5899   .   +   .   ID=AT1G01010.1;Parent=AT1G01010;Name=AT1G01010.1;Index=1
Chr1   TAIR10   protein   3760   5630   .   +   .   ID=AT1G01010.1-Protein;Name=AT1G01010.1;Derives_from=AT1G01010.1
Chr1   TAIR10   gene      5928   8737   .   -   .   ID=AT1G01020;Note=protein_coding_gene;Name=AT1G01020
Chr1   TAIR10   mRNA      5928   8737   .   -   .   ID=AT1G01020.1;Parent=AT1G01020;Name=AT1G01020.1;Index=1
Chr1   TAIR10   protein   6915   8666   .   -   .   ID=AT1G01020.1-Protein;Name=AT1G01020.1;Derives_from=AT1G01020.1
```

解答例

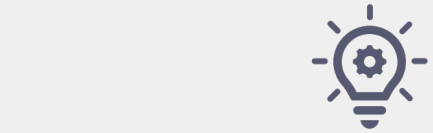


https://www.arabidopsis.org/download_files/Genes/TAIR10_genome_release/TAIR10_gff3/TAIR10_GFF3_genes.gff

農学生命情報科学特論Ⅰ

4

- 可視化



- 深層学習研究事例

pandas

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

Pandas はデータ分析用のパッケージであり、可視化機能も実装されている。シリーズやデータフレーム等を手軽に可視化できる。

ggplot

R / 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。

matplotlib

matplotlib は初期から存在する可視化パッケージである。ユーザーが多いため、情報量も多い。細かな調整が効き、複雑なグラフも描ける。

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。解析結果をリアルタイムに表示させたいときに利用する。

seaborn

seaborn は matplotlib を補完する位置付けである。ペアプロットやヒートマップなどの応用グラフも関数一つで描ける。

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。

可視化

- matplotlib
- seaborn

matplotlib / Application Programming Interface

matplotlib には 2 種類の可視化 API が用意されている。1 つはオブジェクト指向型プログラミング言語を意識した object-oriented interface である。もう 1 つは、MATLAB の使い方を踏襲した state-based interface である。

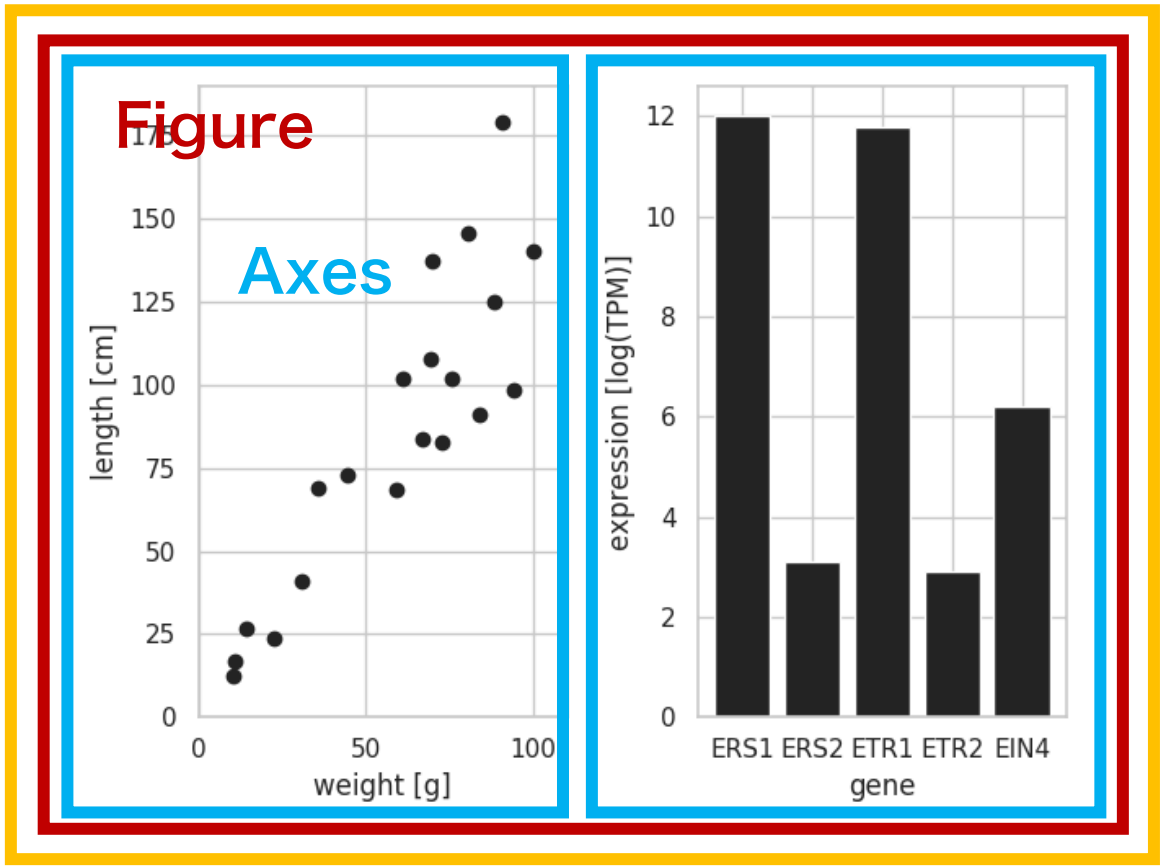
object-oriented interface

プロット領域をいくつかのクラス（サブ領域）に分割し、そのクラスで定義されたメソッド（関数）を使用して、グラフを作成していくインタフェースである。

state-based interface

クラスを意識せずに、あらかじめ用意された関数を使用してグラフを描いていくインタフェースである。

Pyplot



matplotlib API

object-oriented interface

```
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()


ax.plot(x, y)

fig.show()
```

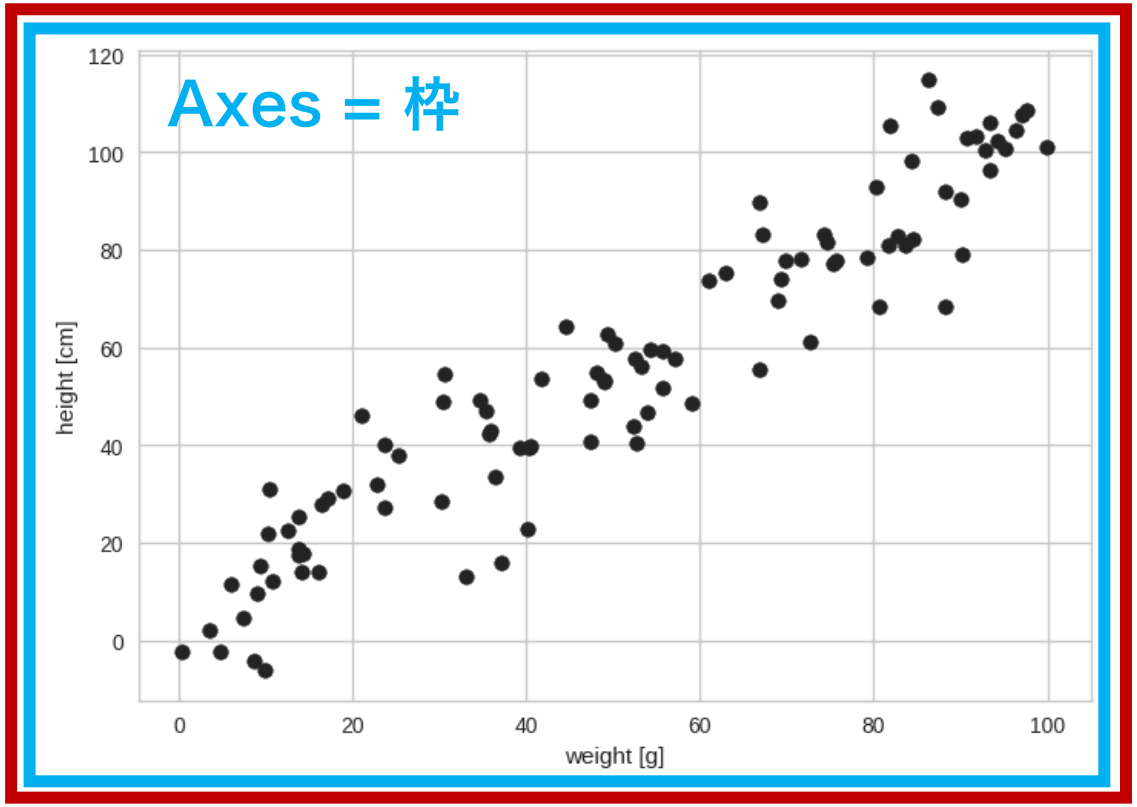
state-based interface

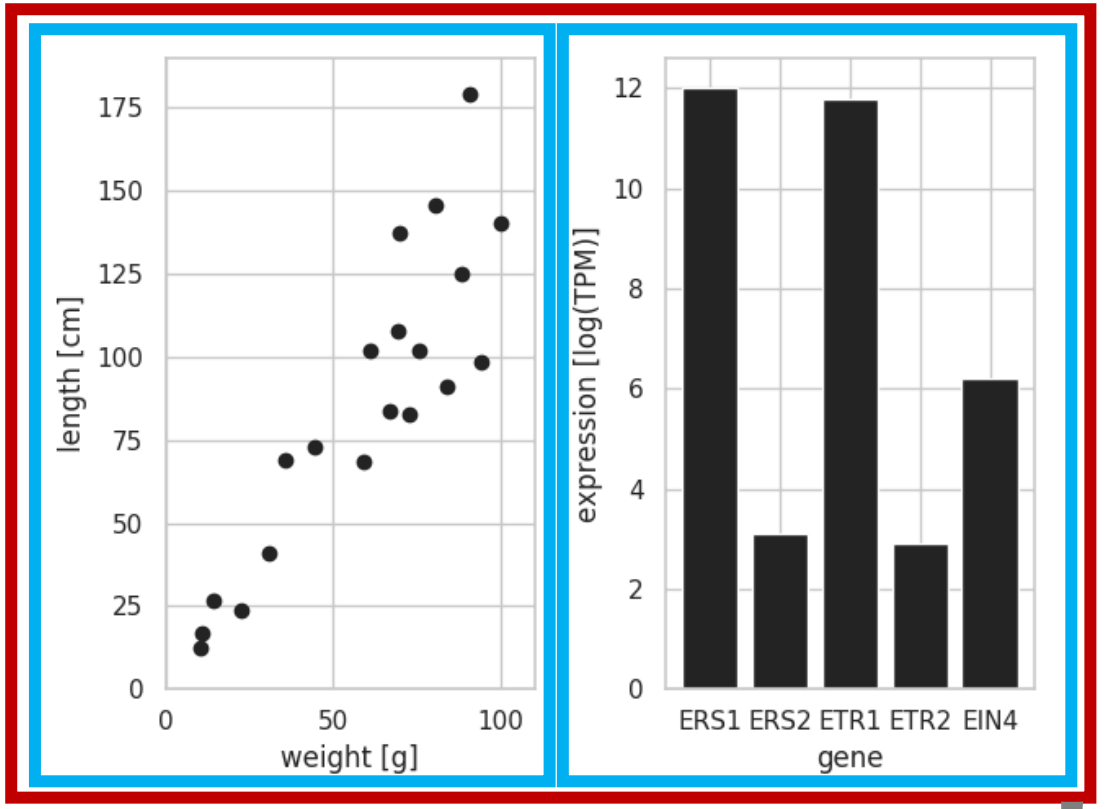
```
import numpy as np
import matplotlib.pyplot as plt

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])




plt.plot(x, y)

plt.show()
```

object-oriented interface

Pyplot = 落書き帳

Figure = ページ

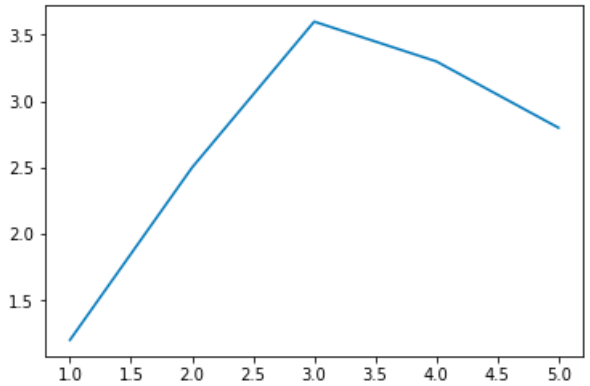




object-oriented interface

matplotlib を利用してグラフを作成するには 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()
ax.plot(x, y)
fig.show()
```



object-oriented interface

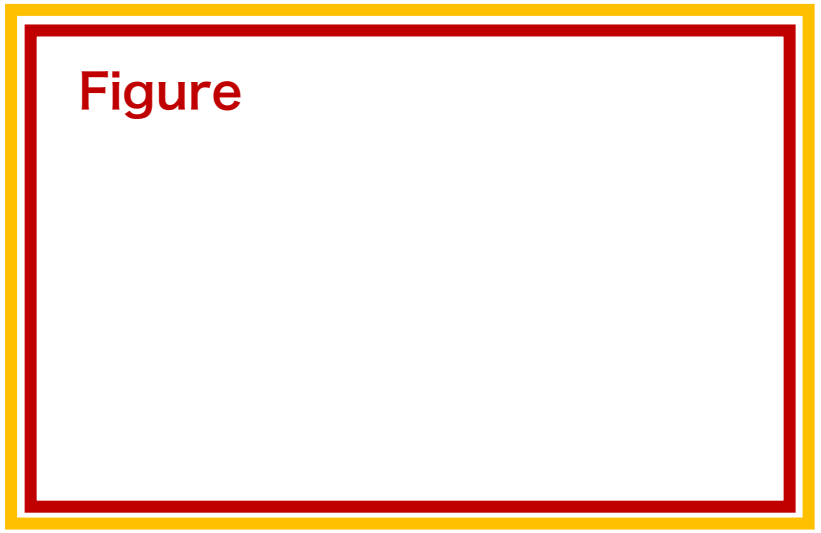
1. matplotlib.pyplot モジュールの機能を呼び出す。pyplot 描画デバイスが用意される。

```
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()
ax.plot(x, y)
fig.show()
```

object-oriented interface

1. matplotlib.pyplot モジュールの機能を呼び出す。pyplot 描画デバイスが用意される。
2. Figure クラスのオブジェクト（領域）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()
ax.plot(x, y)
fig.show()
```



object-oriented interface

1. matplotlib.pyplot モジュールの機能を呼び出す。pyplot 描画デバイスが用意される。
2. Figure クラスのオブジェクト（領域）を用意する。
3. Figure 領域の中に、さらに Axes クラスのオブジェクト（領域）を作成する。

▶

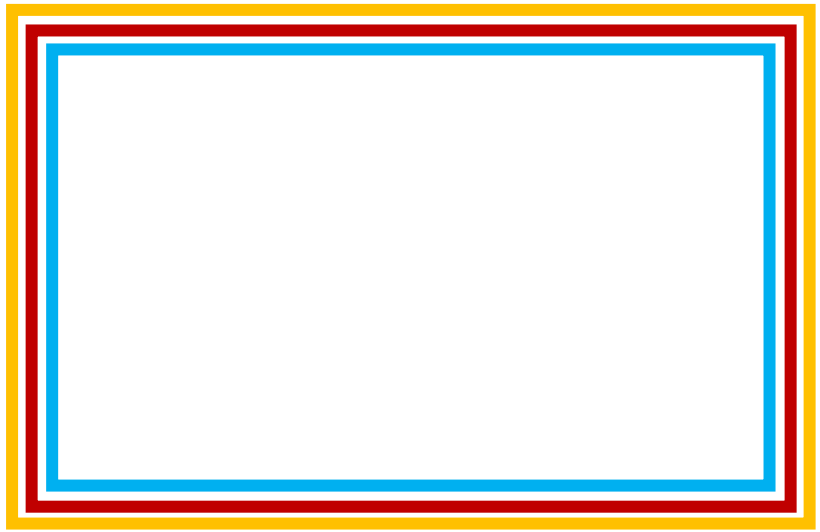
```
import numpy as np
import matplotlib.pyplot as plt

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()
ax.plot(x, y)
fig.show()
```

object-oriented interface

1. matplotlib.pyplot モジュールの機能を呼び出す。pyplot 描画デバイスが用意される。
2. Figure クラスのオブジェクト（領域）を用意する。
3. Figure 領域の中に、さらに Axes クラスのオブジェクト（領域）を作成する。
4. Axes 領域に線グラフを描く。

▶

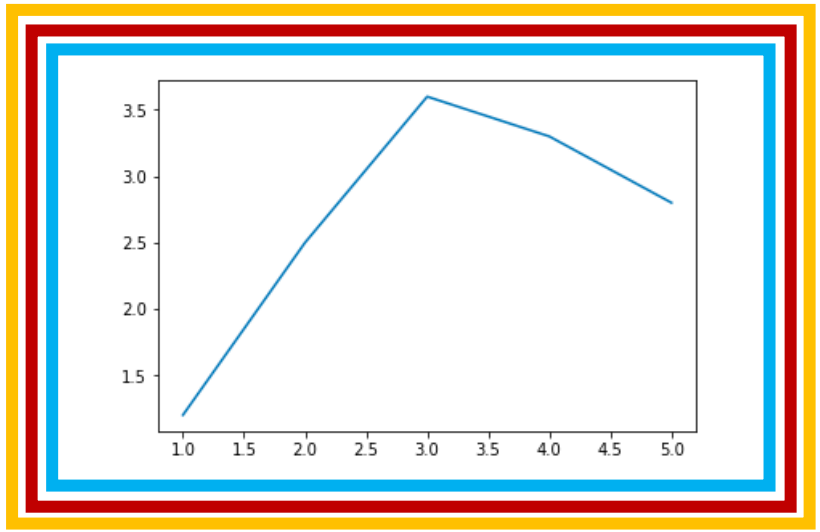
```
import numpy as np
import matplotlib.pyplot as plt

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()
ax.plot(x, y)
fig.show()
```



object-oriented interface

1. matplotlib.pyplot モジュールの機能を呼び出す。pyplot 描画デバイスが用意される。
2. Figure クラスのオブジェクト（領域）を用意する。
3. Figure 領域の中に、さらに Axes クラスのオブジェクト（領域）を作成する。
4. Axes 領域に線グラフを描く。
5. グラフを表示する。

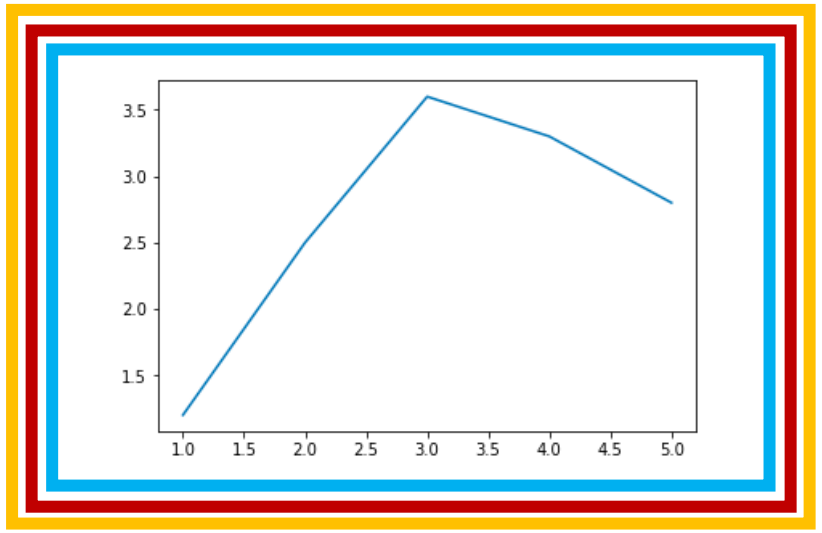
```
import numpy as np
import matplotlib.pyplot as plt

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()
ax.plot(x, y)
fig.show()
```



object-oriented interface / グラフ保存

グラフをファイルに保存するとき、show メソッドの代わりに savefig メソッドを使用する。ファイルのフォーマットは format 引数で指定する。png の他に pdf、ps、eps、svg を指定できる。

```
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()
ax.plot(x, y)
fig.savefig('fig1.png', format='png')
```



object-oriented interface / グラフ保存

グラフの軸目盛りなどに使う文字の大きさを調整したい場合は、set_xlabel などのメソッドを使う。また、グラフ画像のサイズや解像度などを調整したい場合は、Figure 領域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()
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')
```

state-based interface

State-based interface は、すべての操作を pyplot のメソッドとして行うインタフェースである。pyplot が現在操作中の Figure 領域や Axes 領域を自動的に識別して操作し、グラフを作成する。State-based interface を用いて散布図を描くとき、右のようなコードを書く。

```
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])

plt.plot(x, y)
plt.show()
```

matplotlib API 可視化関数

object-oriented interface と state-based interface で利用できる可視化関数は次のように対応している。

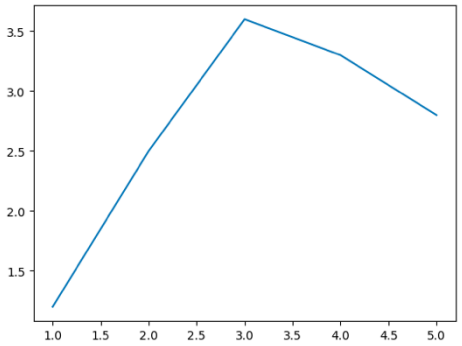
グラフ	object-oriented	state-based
線グラフ	`ax.plot`	`plt.plot`
散布図	`ax.scatter`	`plt.scatter`
棒グラフ	`ax.bar`	`plt.bar`
ヒストグラム	`ax.hist`	`plt.hist`
ボックスプロット	`ax.boxplot`	`plt.boxplot`

matplotlib API 可視化関数

object-oriented interface と state-based interface で利用できる可視化関数は次のように対応している。

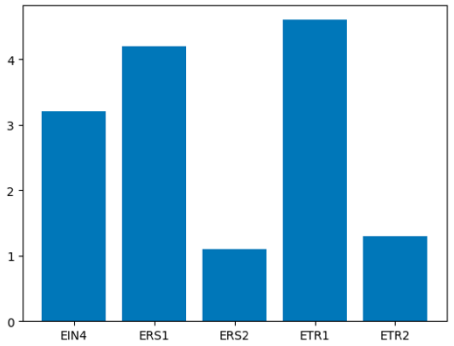
グラフ	object-oriented	state-based
グラフのタイトル	`ax.set_title`	`plt.title`
目盛りの比率	`ax.set_aspect`	`plt.axes().set_aspect`
グラフの凡例	`ax.legend`	`plt.legend`
x 軸の表示範囲	`ax.set_xlim`	`plt.xlim`
x 軸のラベル	`ax.set_xlabel`	`plt.xlabel`
x 軸の目盛りの表示位置	`ax.set_xticks`	`plt.xticks`
x 軸の目盛りの値	`ax.set_xticklabels`	
x 軸の目盛りのスケール	`ax.set_xscale`	`plt.xscale`

基本グラフ



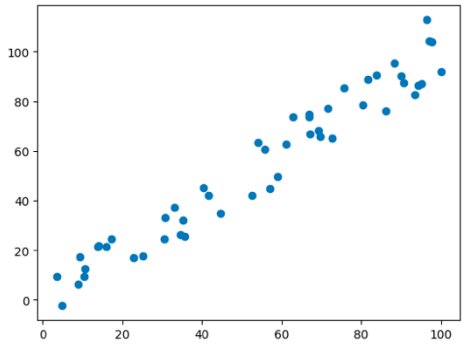
line chart

折れ線グラフは、系列データの変化を捉えるために使われる。



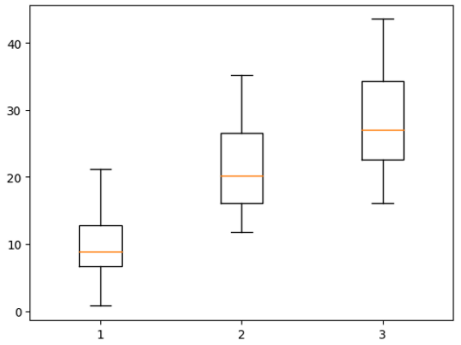
bar chart

棒グラフは、カテゴリカルデータを可視化する目的で使われる。なお、人を騙す目的で使用する場合は縦軸の起点を 0 以外にすると効果的である。



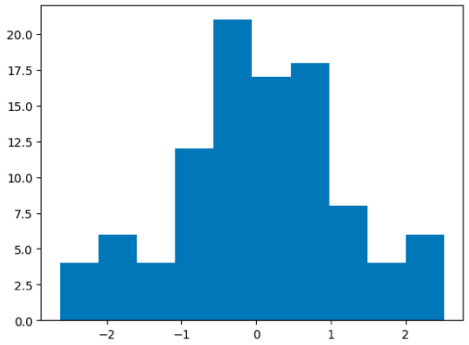
scatter chart

散布図は、2 変量の連続値データ同士の相関や分布などを可視化する目的で使われる。



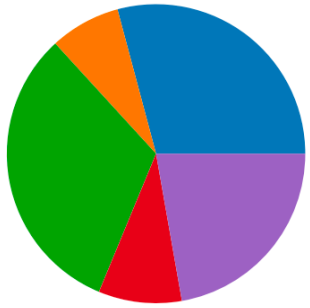
boxplot

ボックスプロットは、複数の連続量データの分布の特徴を可視化する目的で使われる。



histogram

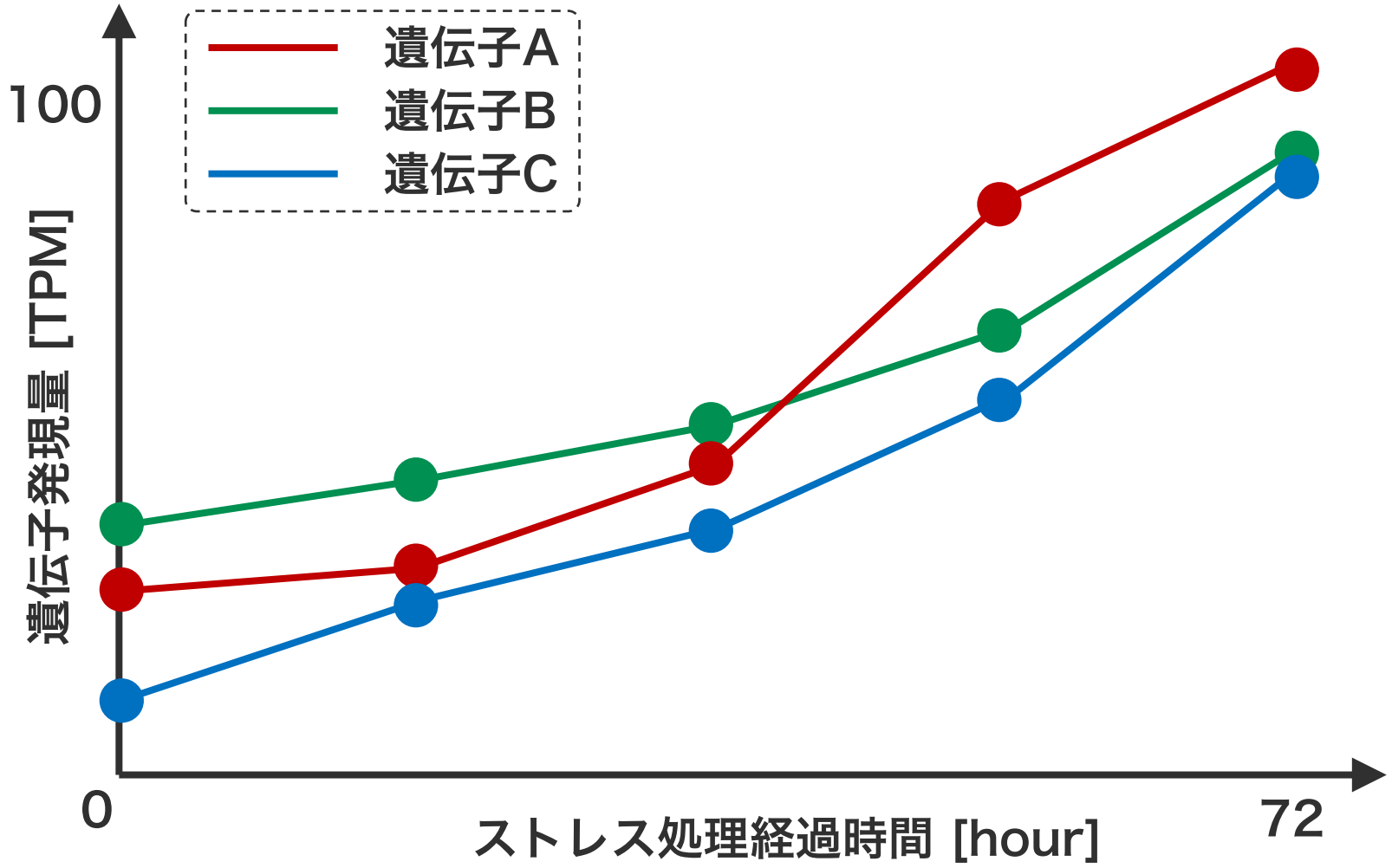
ヒストグラムは、1 変量の連続値データの分布を可視化する目的で使われる。



pie chart

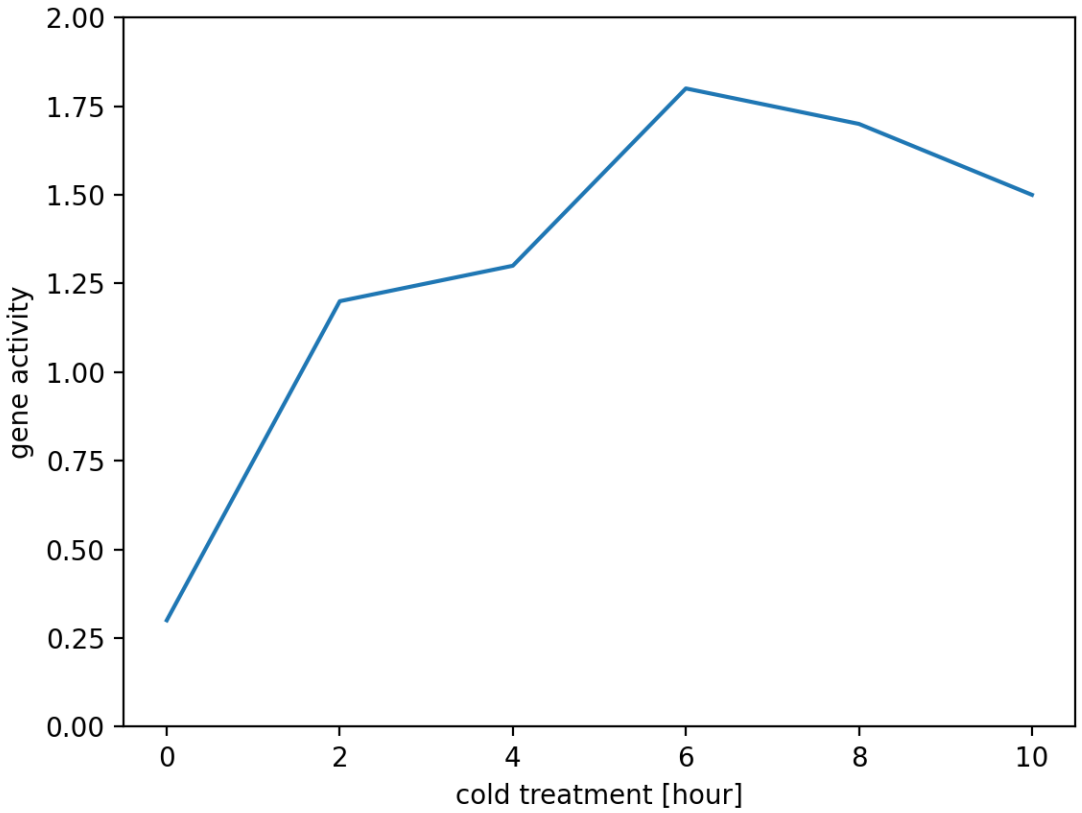
円グラフは、人を騙す目的で多用される。なお、騙し効果を上げるために、3D 円グラフや歪んだ円グラフを用いると良い。

線グラフ



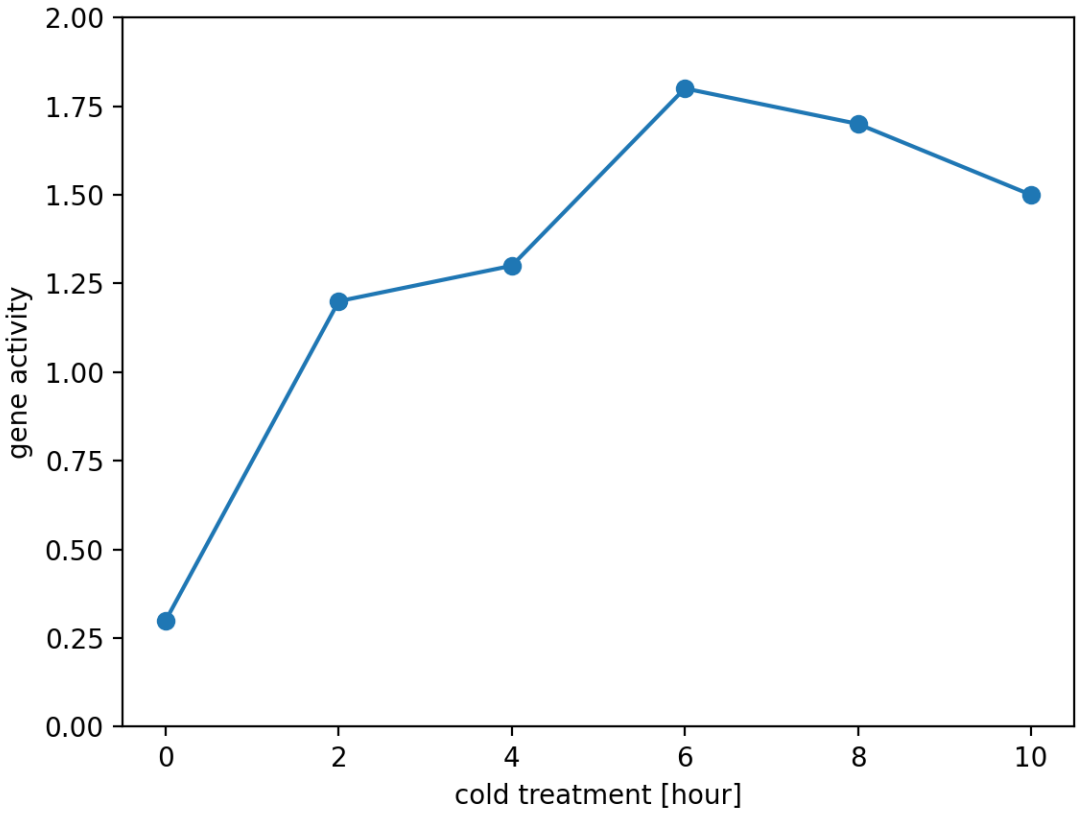
- 線グラフはデータの系列的な変化を見るためのグラフ
- 縦軸および横軸は連続量
- 原点 (0, 0) が省略されることがある
- 観測値を強調するために、観測値に点を明示することもある

線グラフ



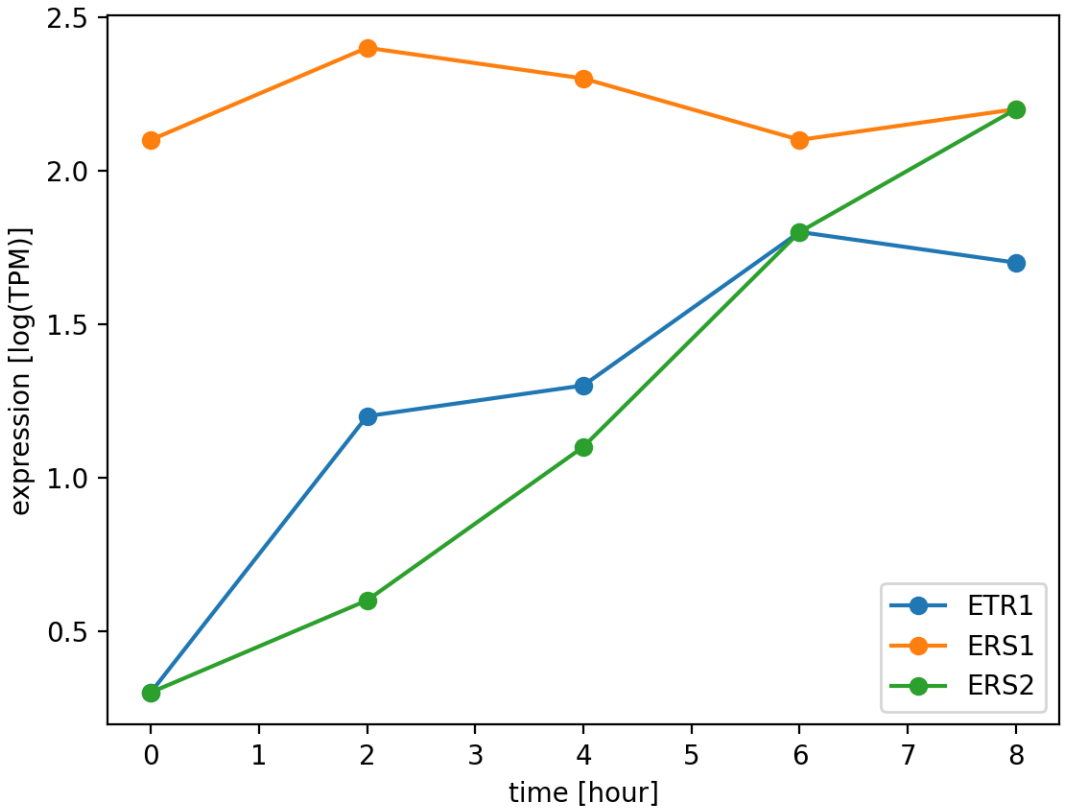

```
import matplotlib.pyplot as plt
import numpy as np

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()

ax.plot(x, y)

ax.set_xlabel('cold treatment [hour]')
ax.set_ylabel('gene activity')
ax.set_ylim(0, 2)

fig.show()
```

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()

ax.plot(x, y, marker='o')

ax.set_xlabel('cold treatment [hour]')
ax.set_ylabel('gene activity')
ax.set_ylim(0, 2)

fig.show()
```

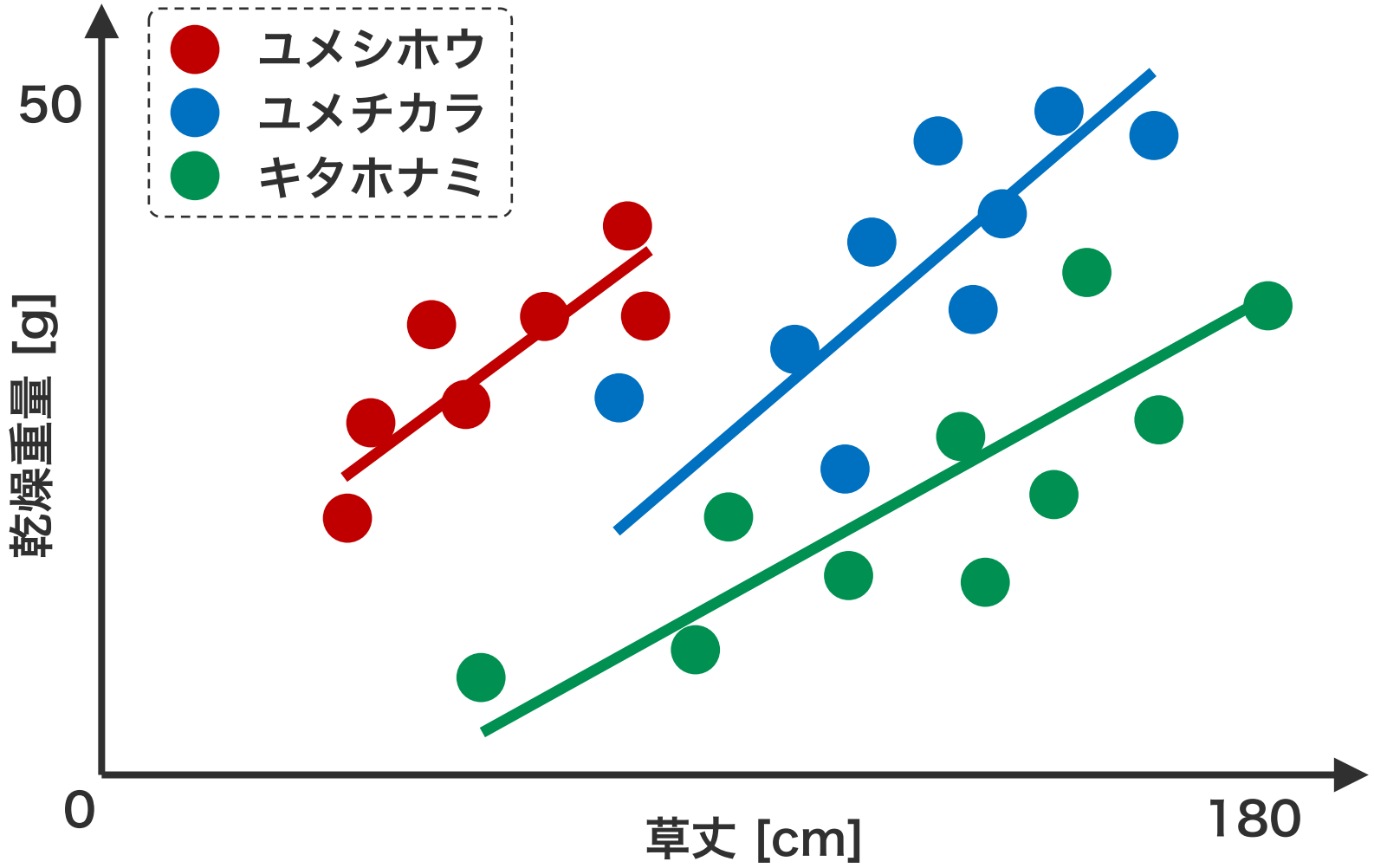
線グラフ




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

x = np.array([0, 2, 4, 6, 8])
g1 = np.array([0.3, 1.2, 1.3, 1.8, 1.7])
g2 = np.array([2.1, 2.4, 2.3, 2.1, 2.2])
g3 = np.array([0.3, 0.6, 1.1, 1.8, 2.2])

fig = plt.figure()

ax = fig.add_subplot()

ax.plot(x, g1, label='ETR1', marker='o')
ax.plot(x, g2, label='ERS1', marker='o')
ax.plot(x, g3, label='ERS2', marker='o')

ax.legend()
ax.set_xlabel('time [hour]')
ax.set_ylabel('expression [log(TPM)]')
fig.show()
```

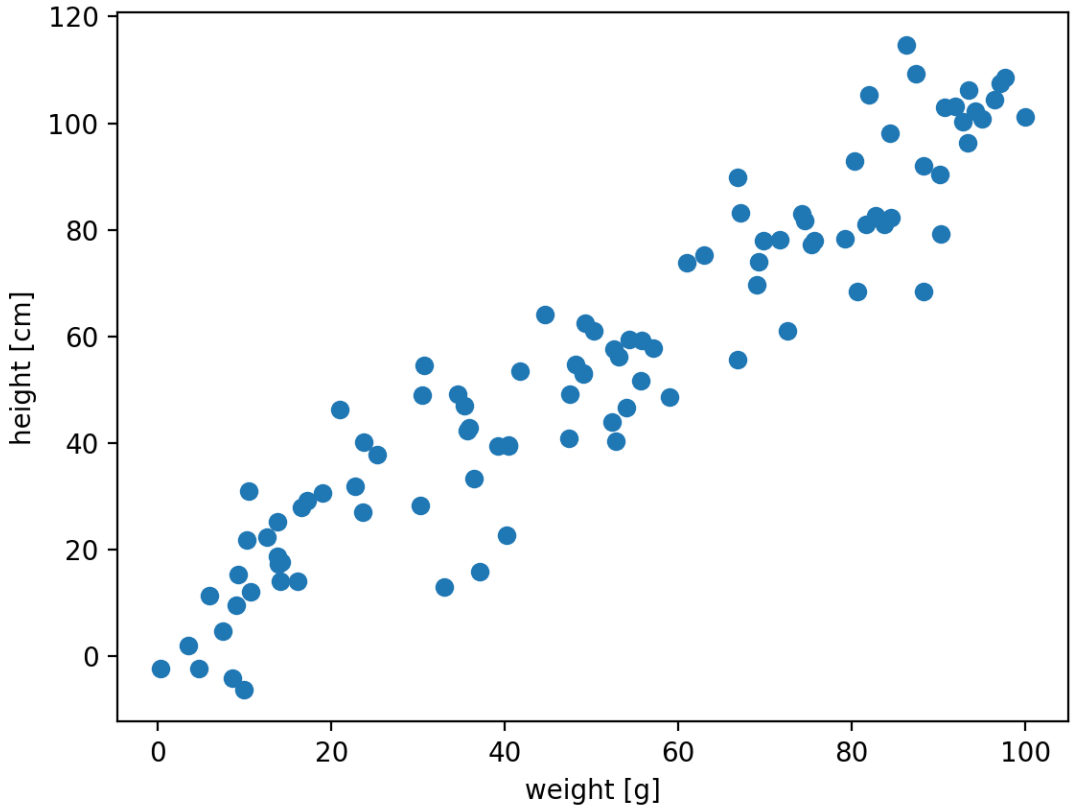
plot メソッドを複数回使うことで、複数の線グラフを描くことができる。グラフを描く際に色を指定しない場合は、自動的に配色される。

散布図



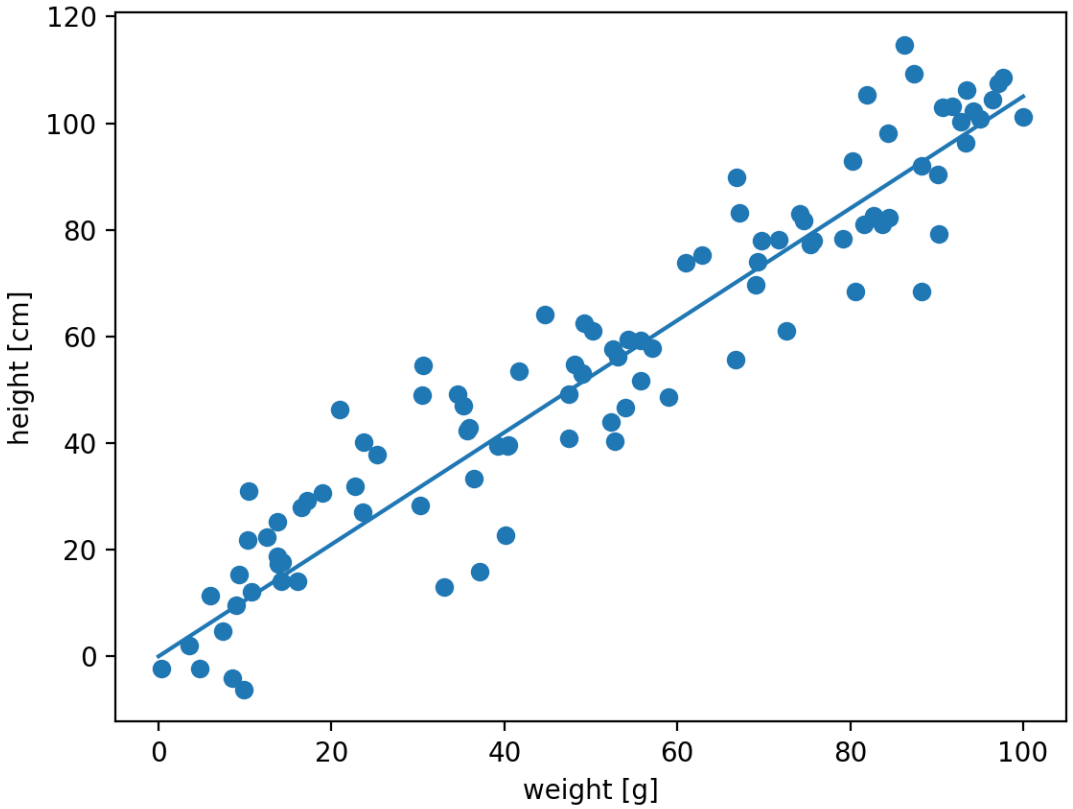
- 多変量データ同士の相関や分布などを見るためのグラフ
- 縦軸および横軸は連続量
- 原点（0, 0）が省略されることがある
- 回帰直線との併用もよく見られる

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()

ax.scatter(x, y)


ax.set_xlabel('weight [g]')
ax.set_ylabel('height [cm]')

fig.show()
```

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()

ax.scatter(x, y)
ax.plot([0, 100], [0, 100 + 5])

ax.set_xlabel('weight [g]')
ax.set_ylabel('height [cm]')

fig.show()
```

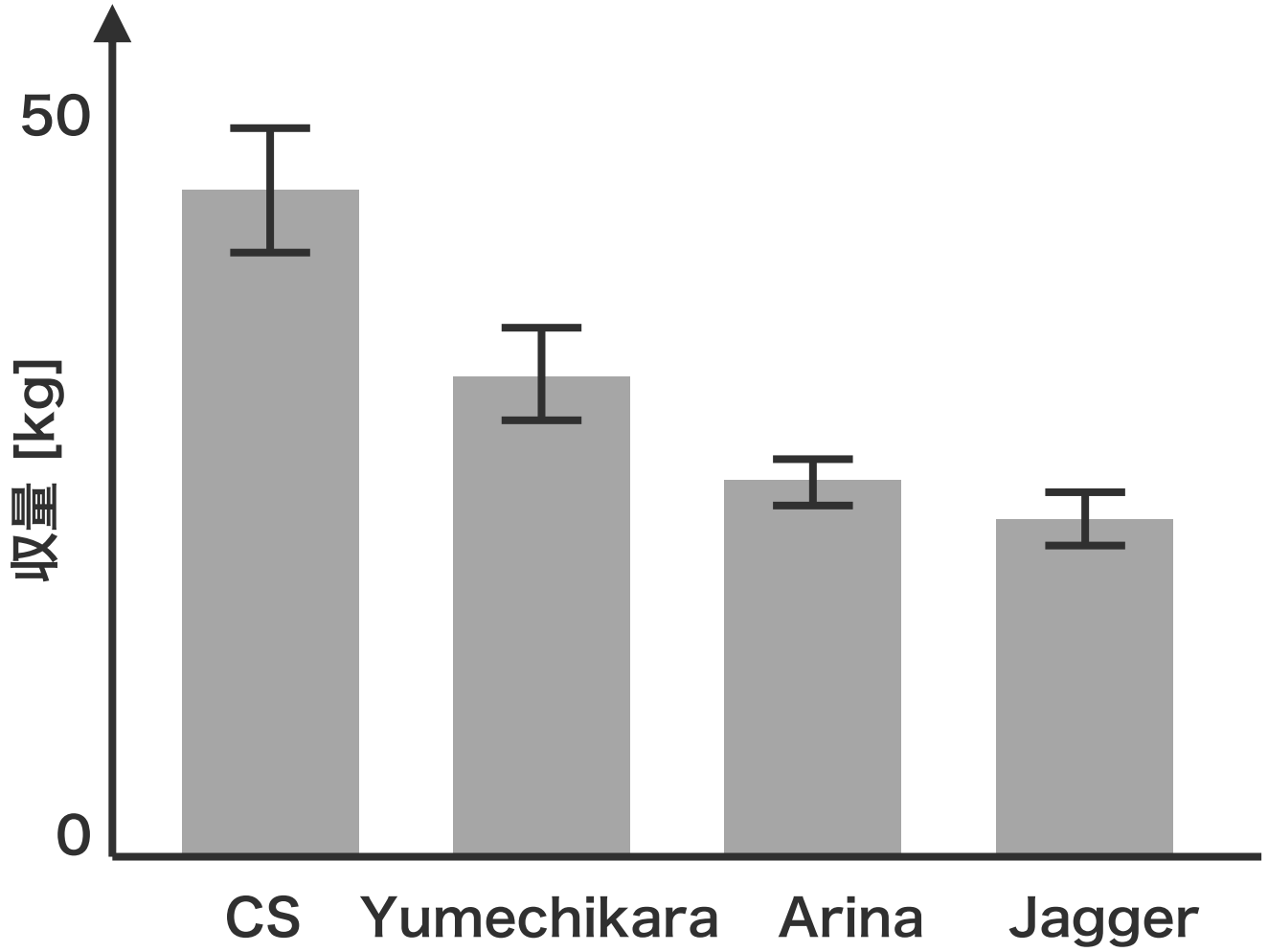
問題 M1-1

diversity_galapagos.txt には、ガラパゴス島における種の多様性データが記載されている。このデータを読み込み、島の面積（Area）と種数（Species）の関係を散布図で描け。

```
import pandas as pd

f = 'diversity_galapagos.txt'
d = pd.read_csv(f, comment='#', header=0, sep='\t', index_col=0)
```

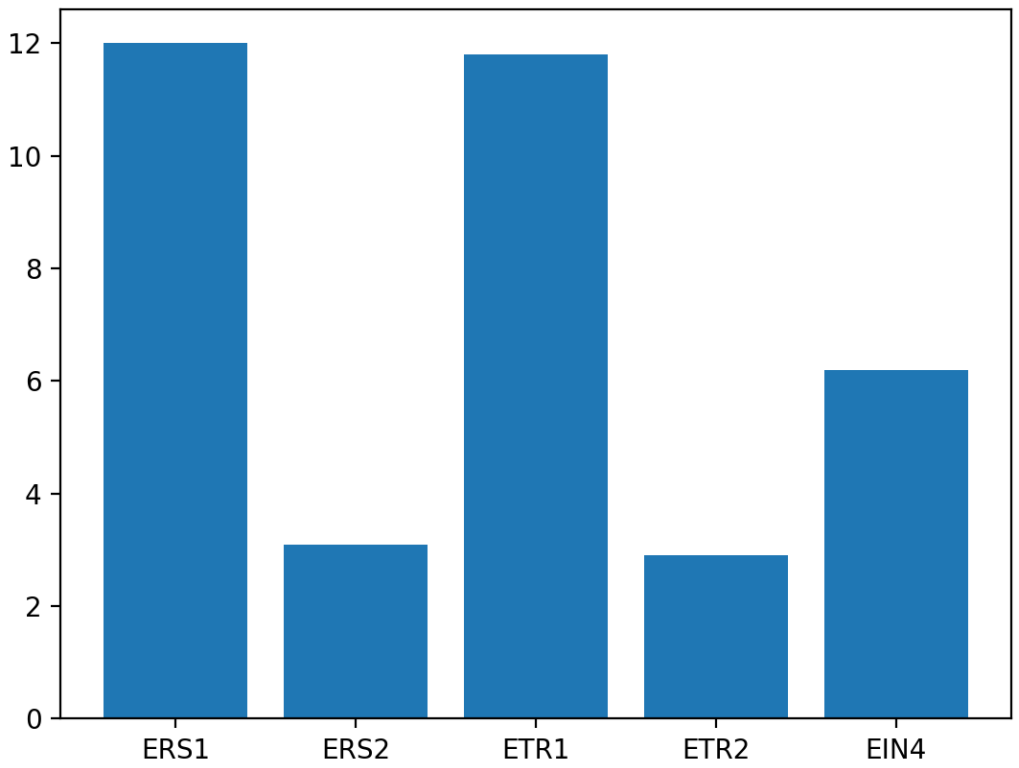
https://aabbdd.jp/notes/data/diversity_galapagos.txt

棒グラフ



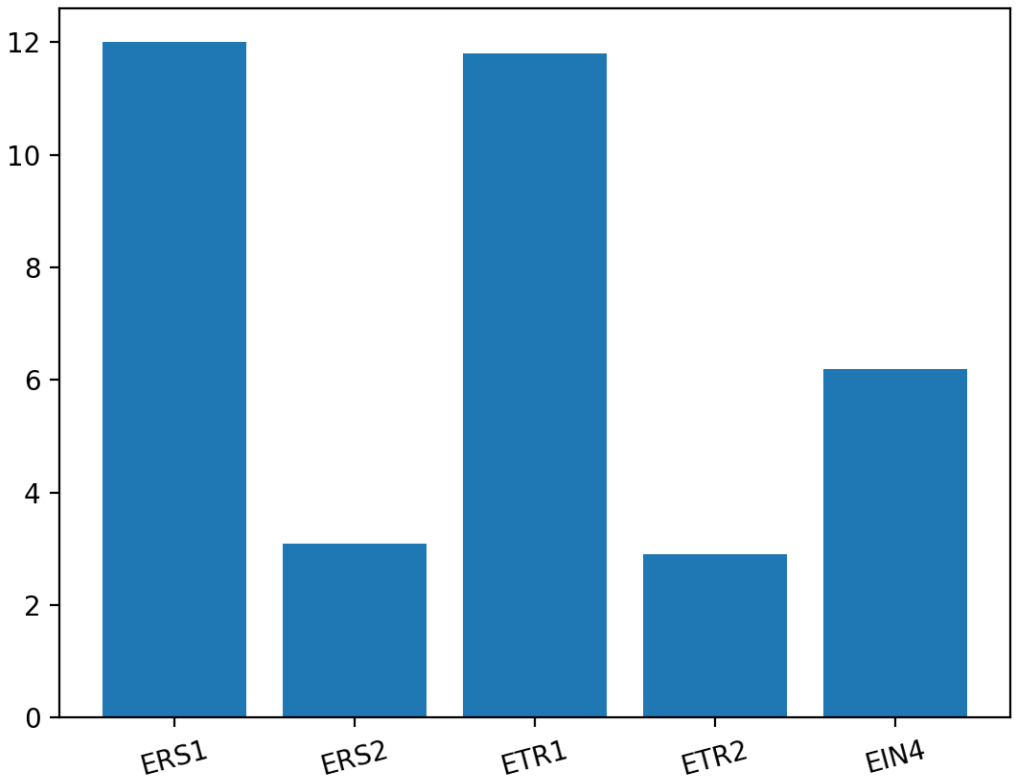
- 複数のカテゴリに属している値同士の大小を可視化するためのグラフ
- 縦軸が連続量、横軸がカテゴリ、あるいはその逆
- 人を騙す目的で使用するとき、原点をゼロ以外の値にしたり、縦軸の目盛り間隔を操作すると効果的である
- エラーバーとともに用いられることがある

棒グラフ



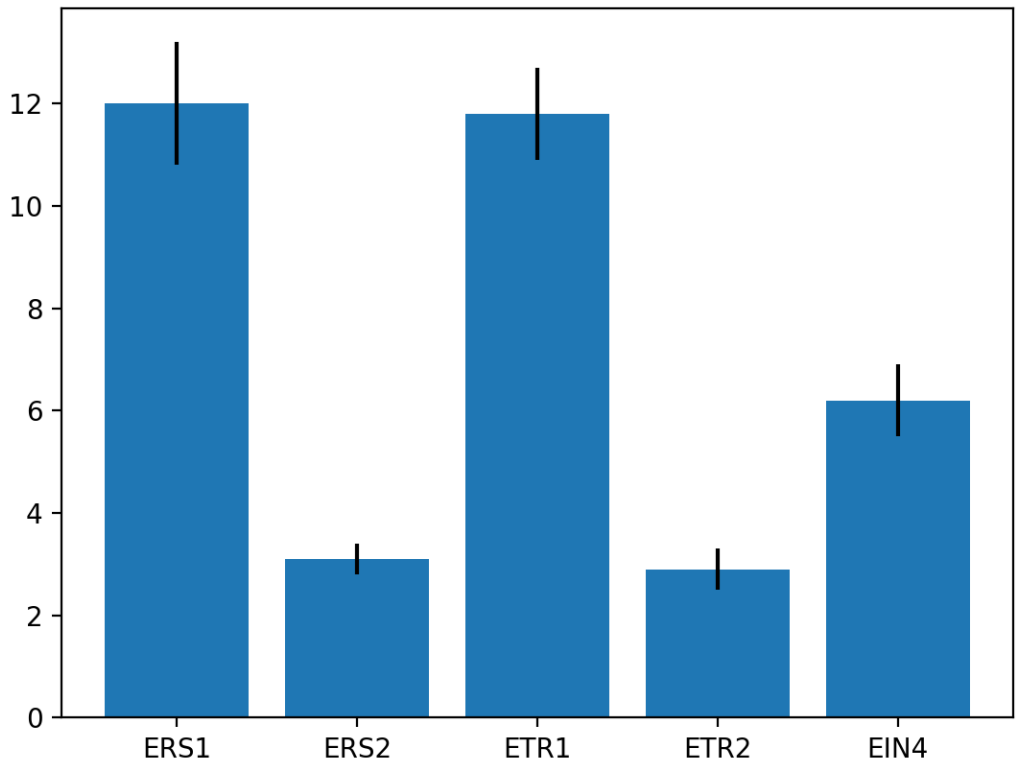

```
import matplotlib.pyplot as plt
import numpy as np

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()
ax.bar(x, y)

fig.show()
```

棒グラフ



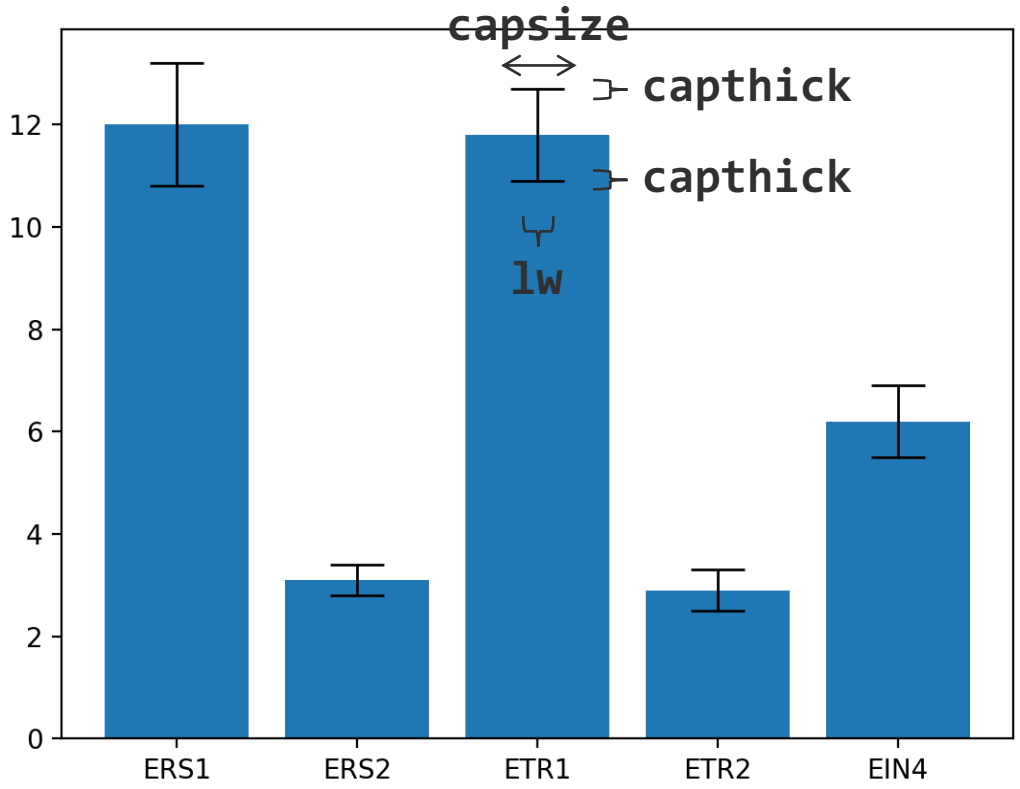

```
import matplotlib.pyplot as plt
import numpy as np

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

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

fig.show()
```

棒グラフ



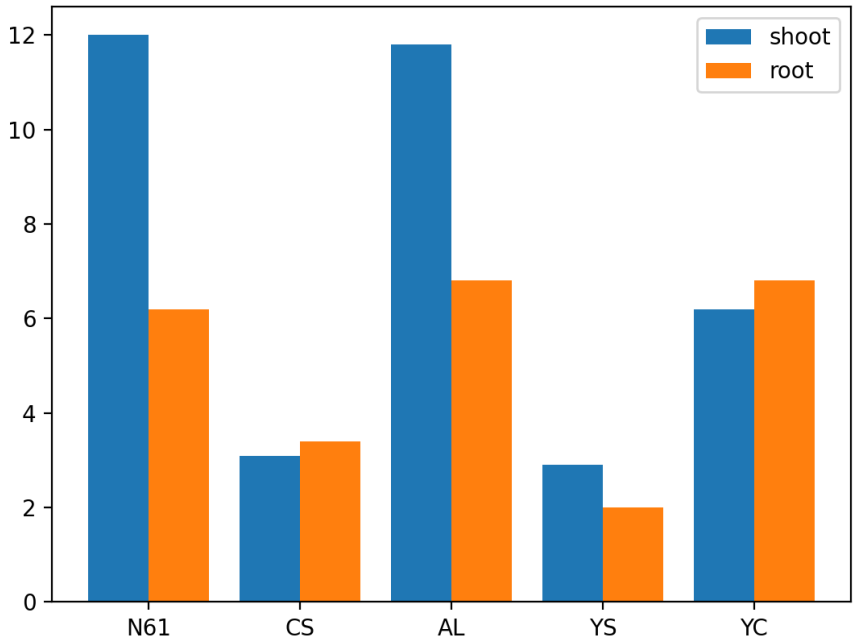

```
import matplotlib.pyplot as plt
import numpy as np

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()


ax.bar(x, y, yerr = e)

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

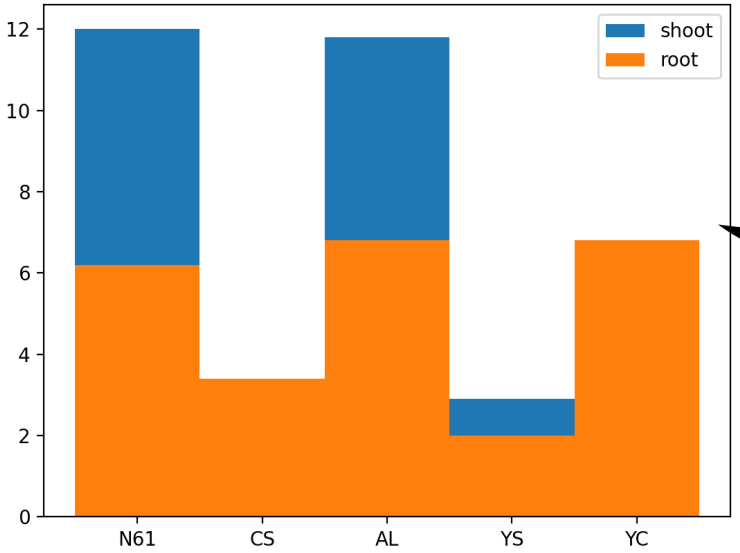
棒グラフ



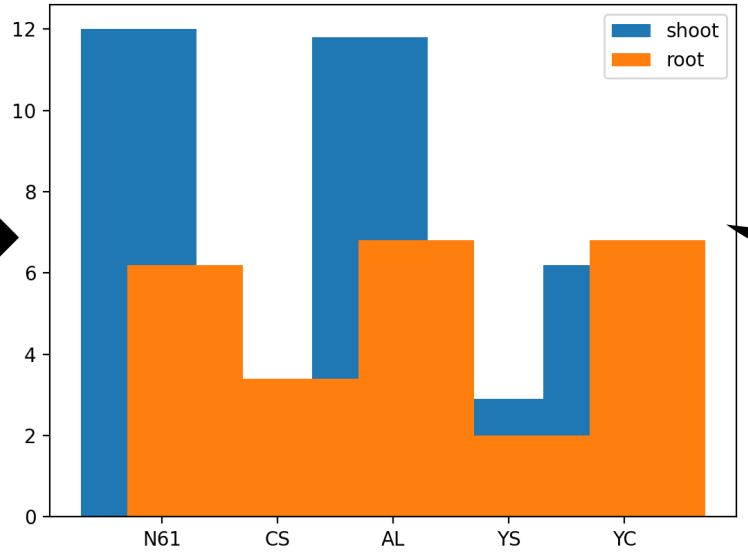

```
import matplotlib.pyplot as plt
import numpy as np

xlabel = np.array(['N61', 'CS', 'AL', 'YS',  'YC'])

x = np.array([0, 1, 2, 3, 4])
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()

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

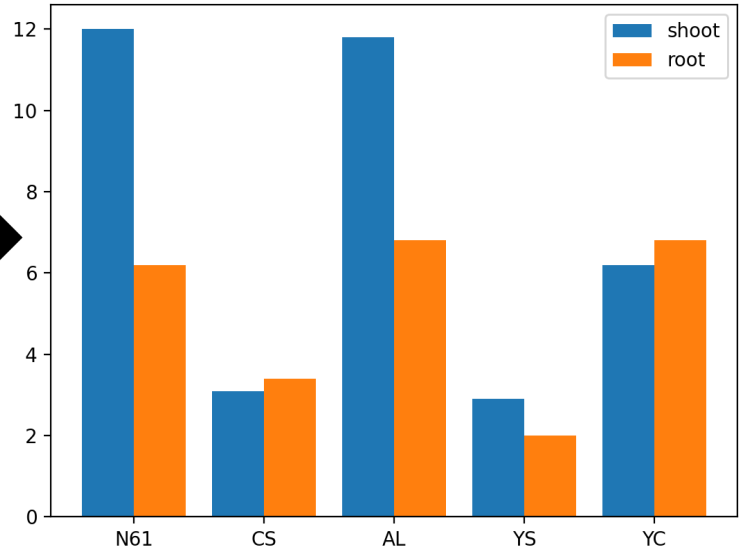
```
ax.bar(x, shoot)
ax.bar(x, root)
```



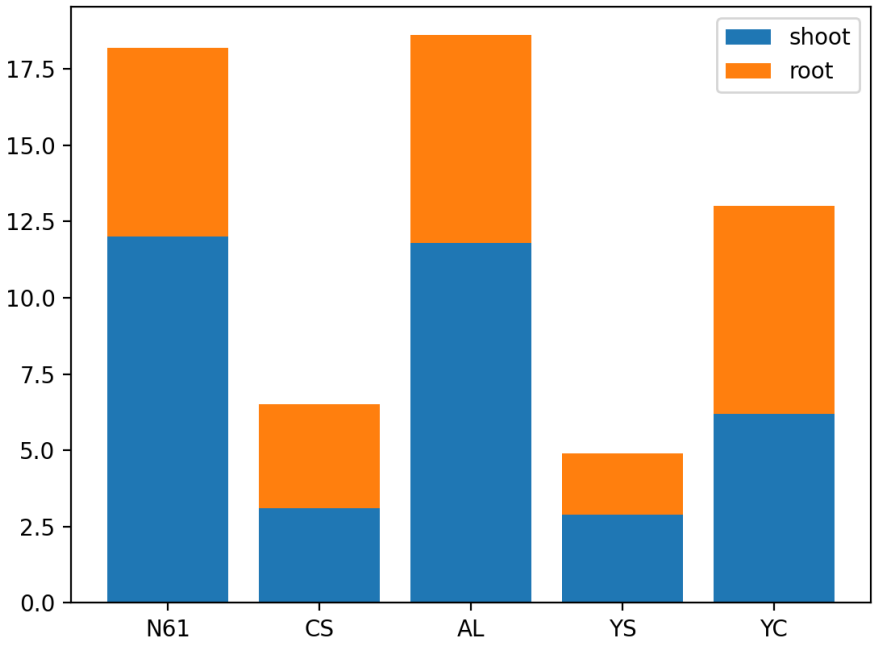

```
ax.bar(x - 0.2, shoot)
ax.bar(x + 0.2, root)
```




```
ax.bar(x - 0.2, shoot, width=0.4)
ax.bar(x + 0.2, root, width=0.4)
```



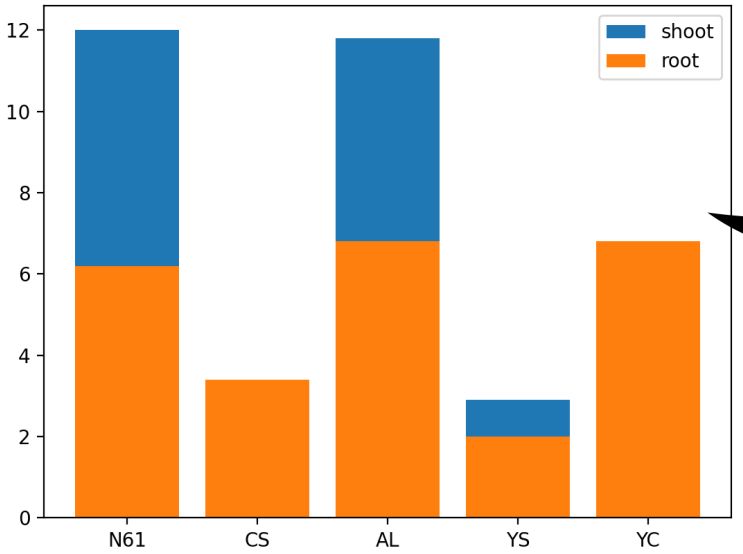
棒グラフ



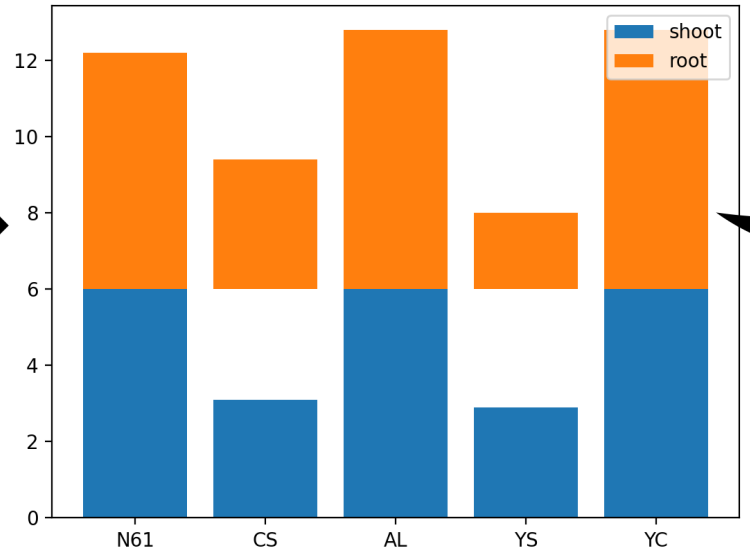

```
import matplotlib.pyplot as plt
import numpy as np

xlabel = np.array(['N61', 'CS', 'AL', 'YS', 'YC'])

x = np.array([0, 1, 2, 3, 4])
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()

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()
```

棒グラフ

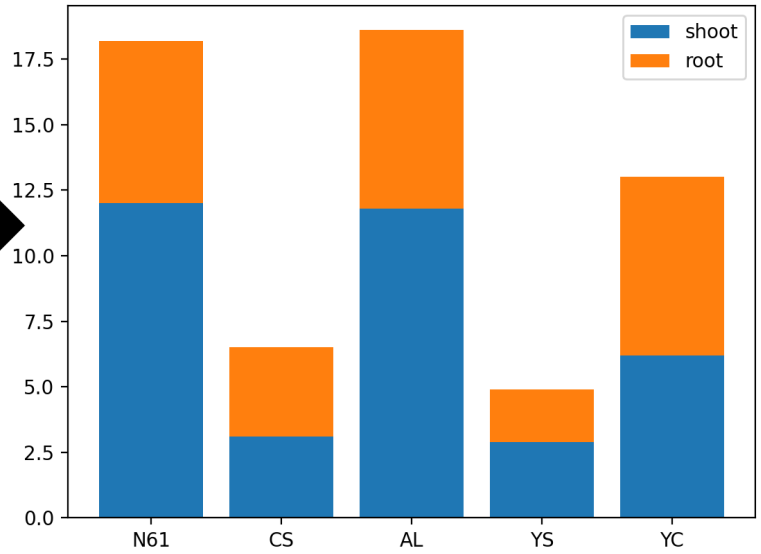
```
ax.bar(x, shoot)
ax.bar(x, root)
```



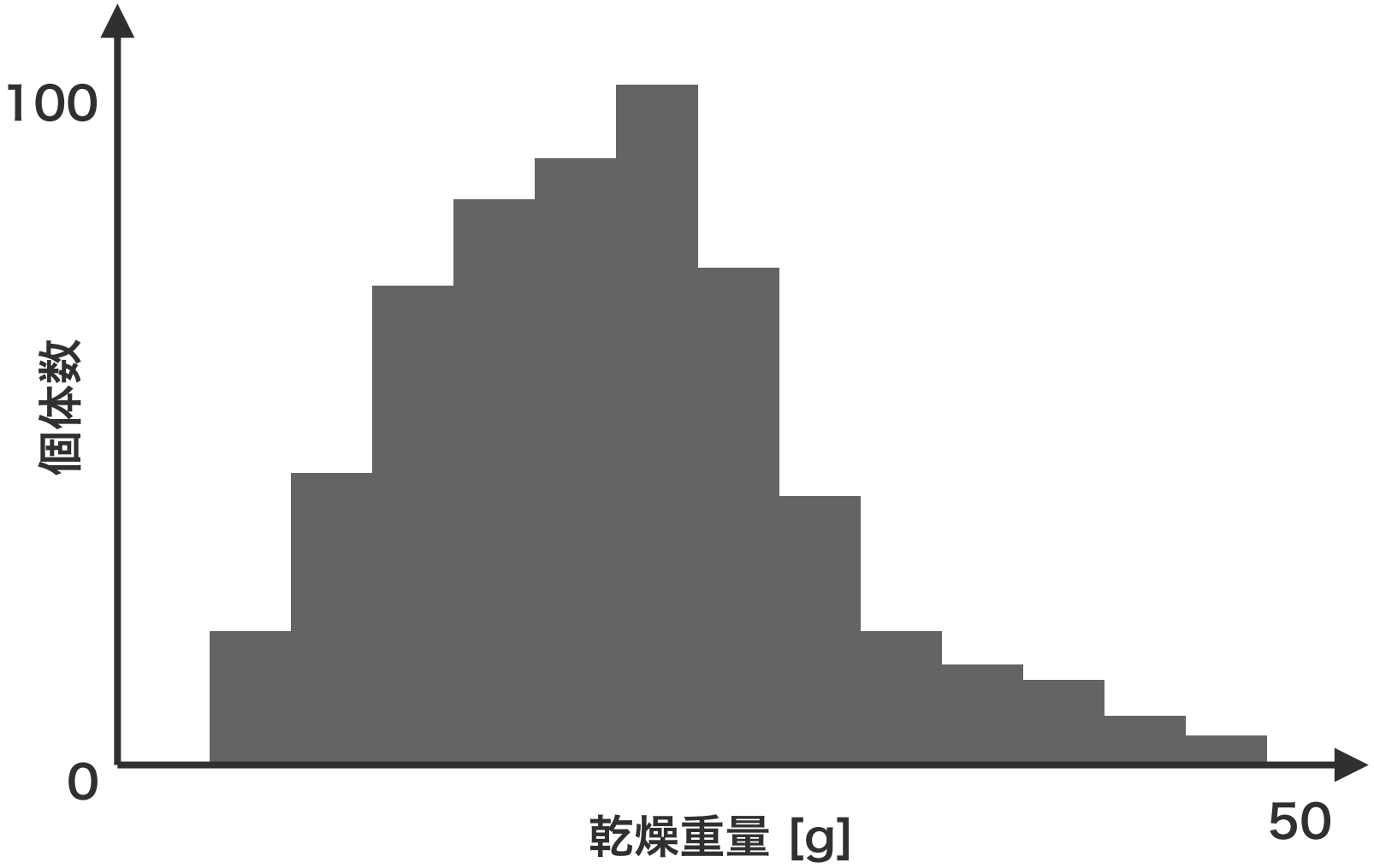

```
ax.bar(x, shoot)
ax.bar(x, root, bottom=6)
```




```
ax.bar(x, shoot)
ax.bar(x, root, bottom=shoot)
```

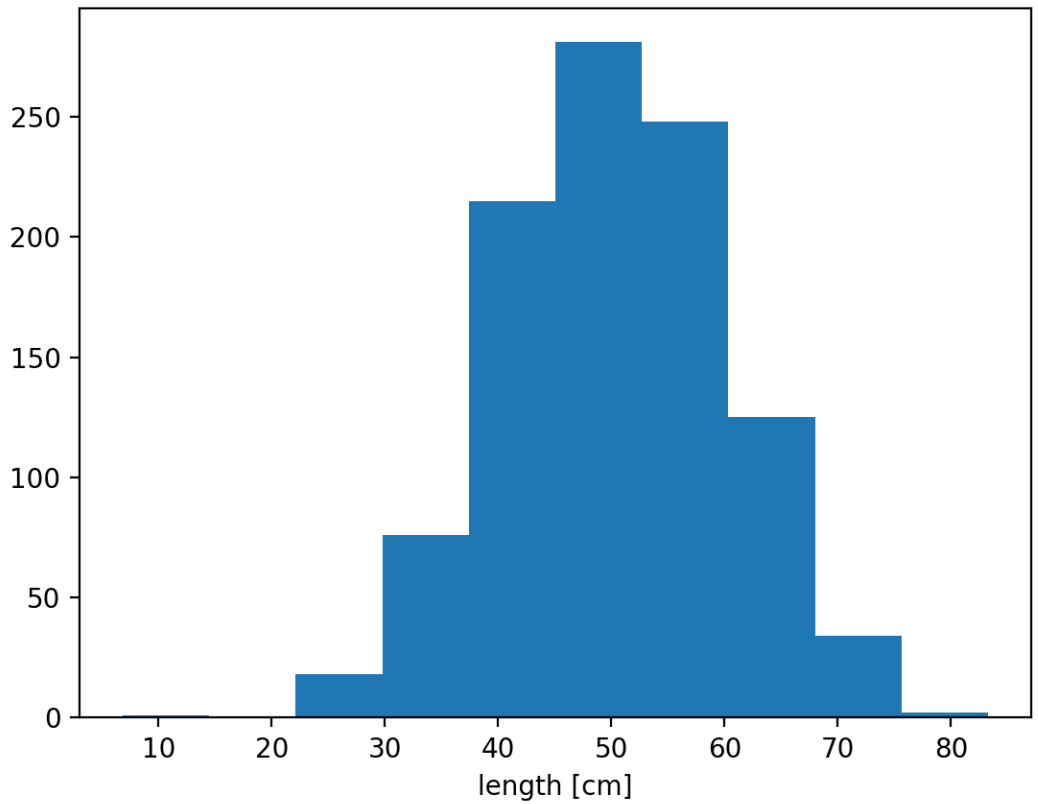


ヒストグラム



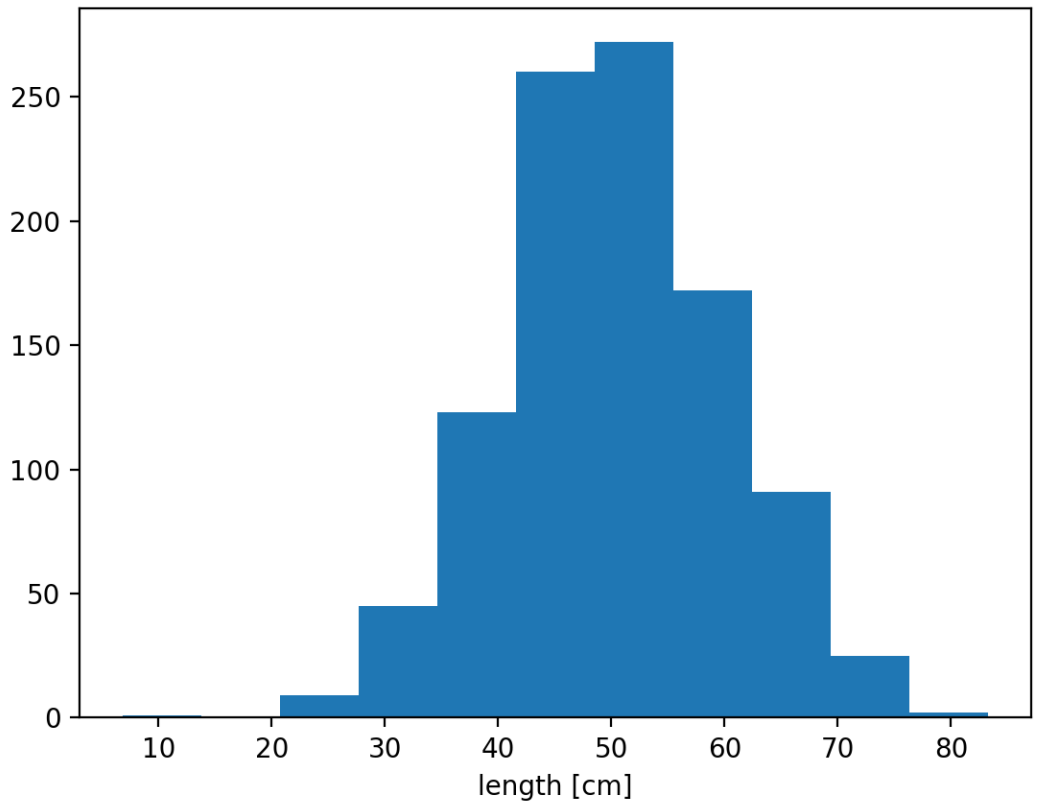
- １変量の連続値データを可視化するためのグラフ
- 横軸が連続量であり、縦軸は頻度・個数または確率である
- 横幅は恣意的（経験的）に決めることもあれば、スタージェスの公式などで決めることもある

ヒストグラム



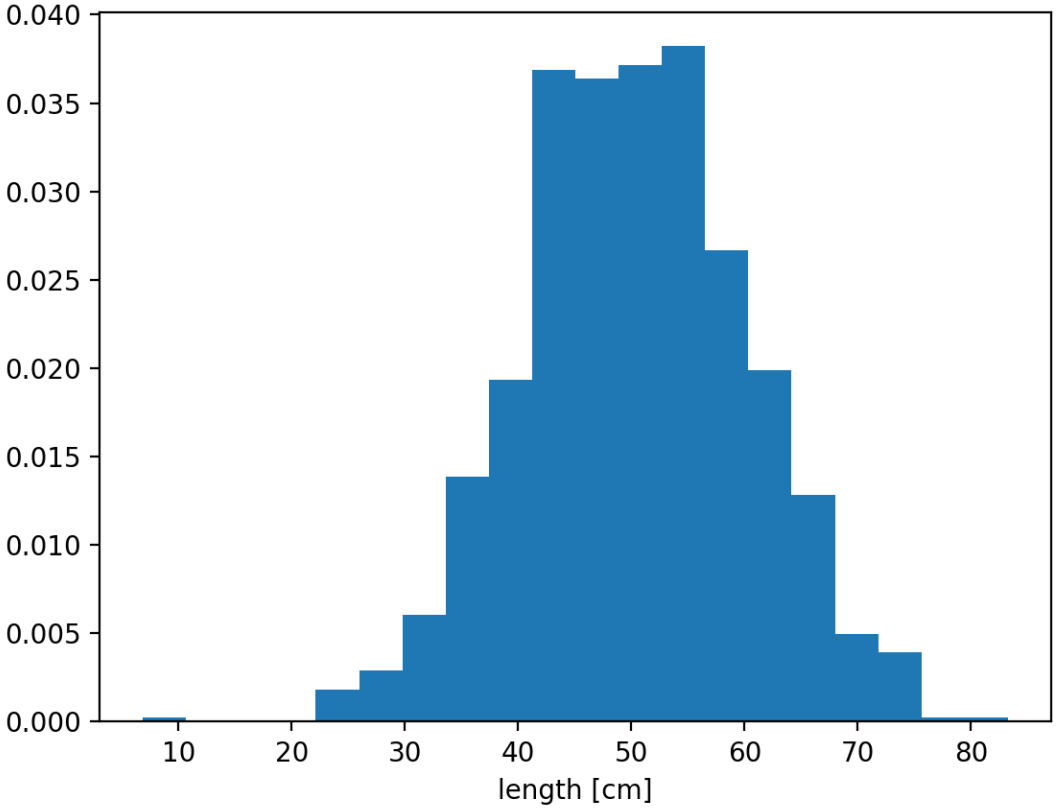

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(2018)

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

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

ax.set_xlabel('length [cm]')

fig.show()
```

ヒストグラム




```
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()
ax.hist(x, bins='sturges')

ax.set_xlabel('length [cm]')

fig.show()
```

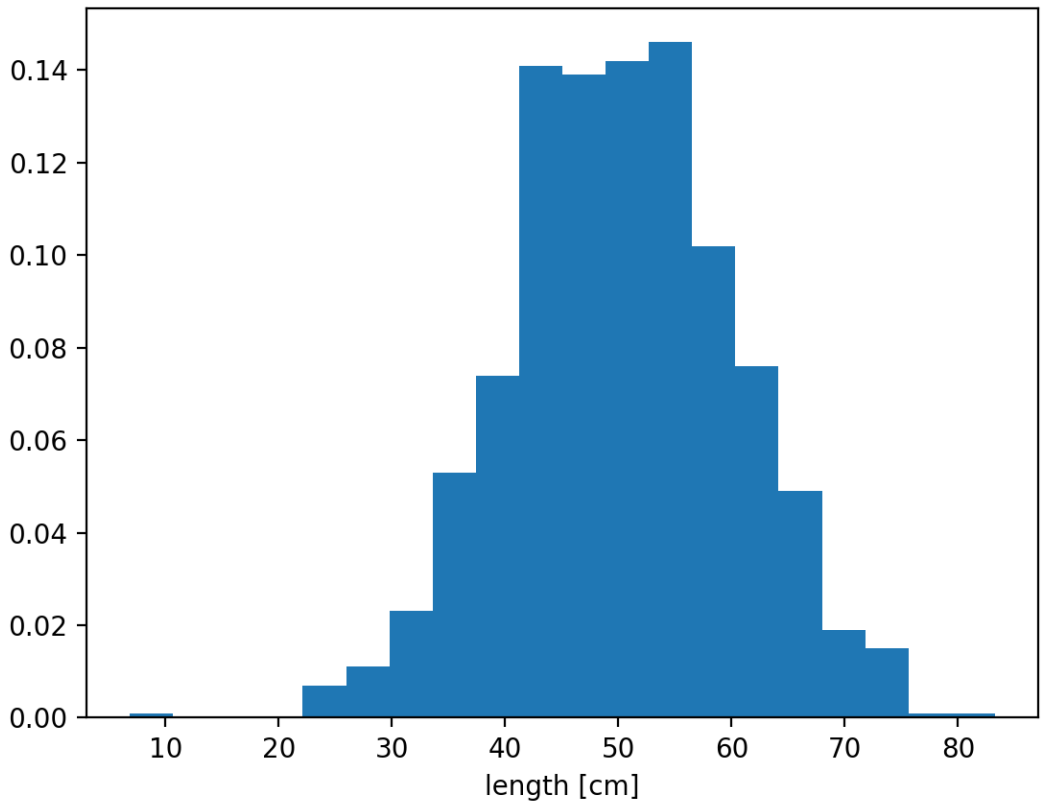
ヒストグラム




```
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()
ax.hist(x, bins=20, density=True)

ax.set_xlabel('length [cm]')

fig.show()
```

density=True を指定したとき、すべてのビンの面積を足すと 1 になる。棒の高さを足して 1 になるわけではないことに注意。

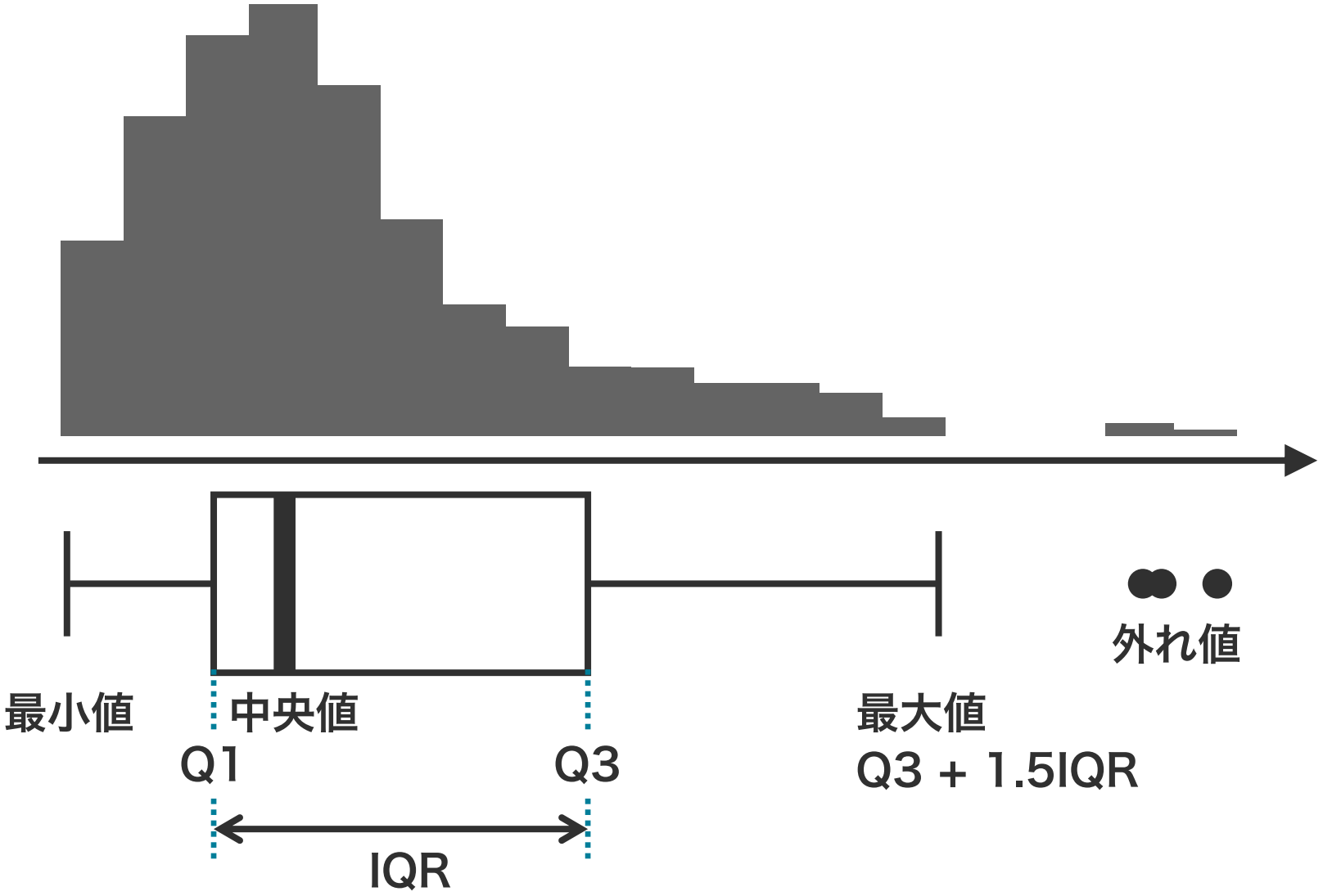
ヒストグラム




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

np.random.seed(2018)

x = np.random.normal(50, 10, 1000)
w = np.ones_like(x)/float(len(x))

fig = plt.figure()
ax = fig.add_subplot()
ax.hist(x, bins=20, weights=w)
ax.set_xlabel('length [cm]')

fig.show()
```

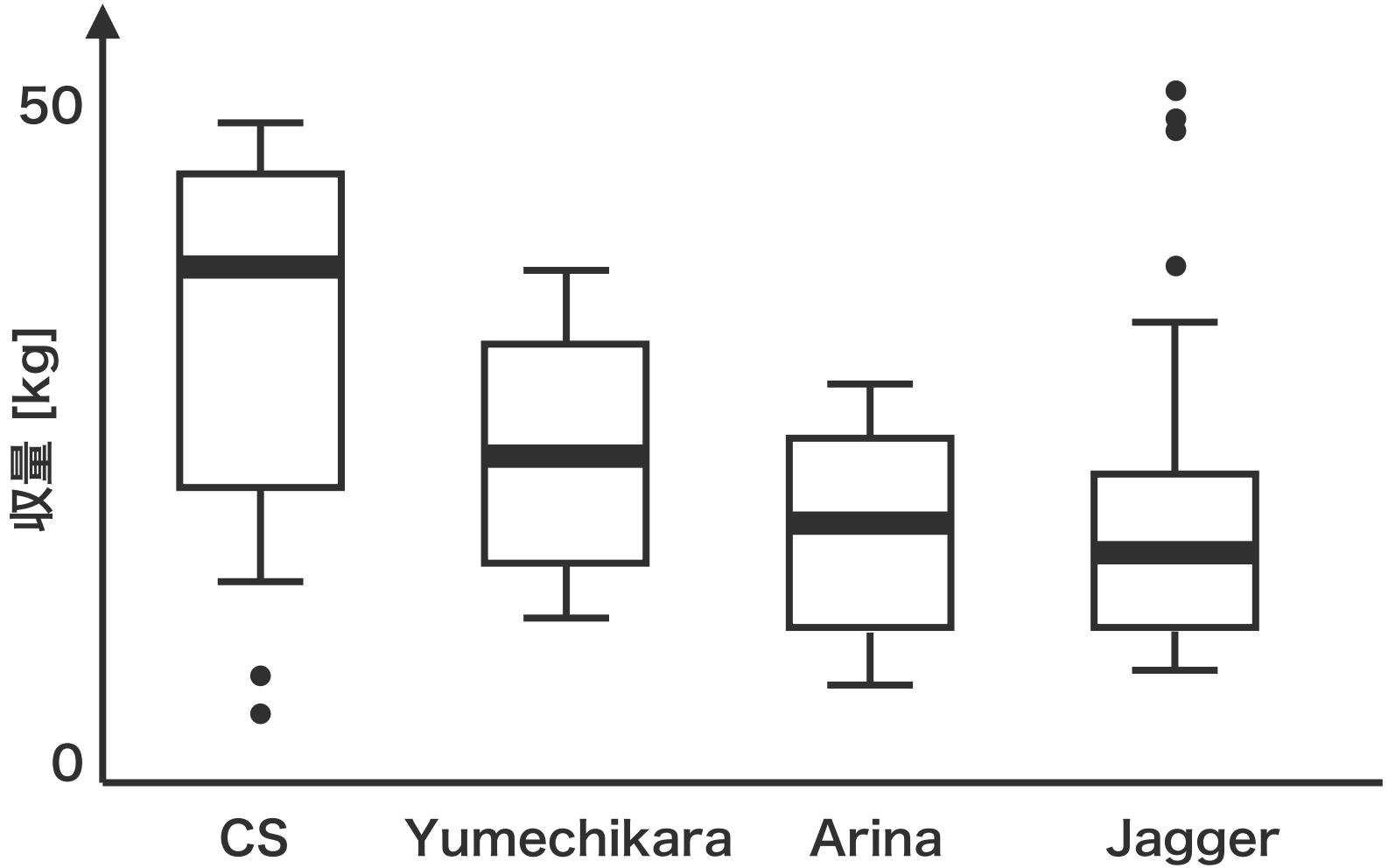
ビンの高さの合計値が 1 となるように、データに重みをかけてヒストグラムを描く。

ボックスプロット



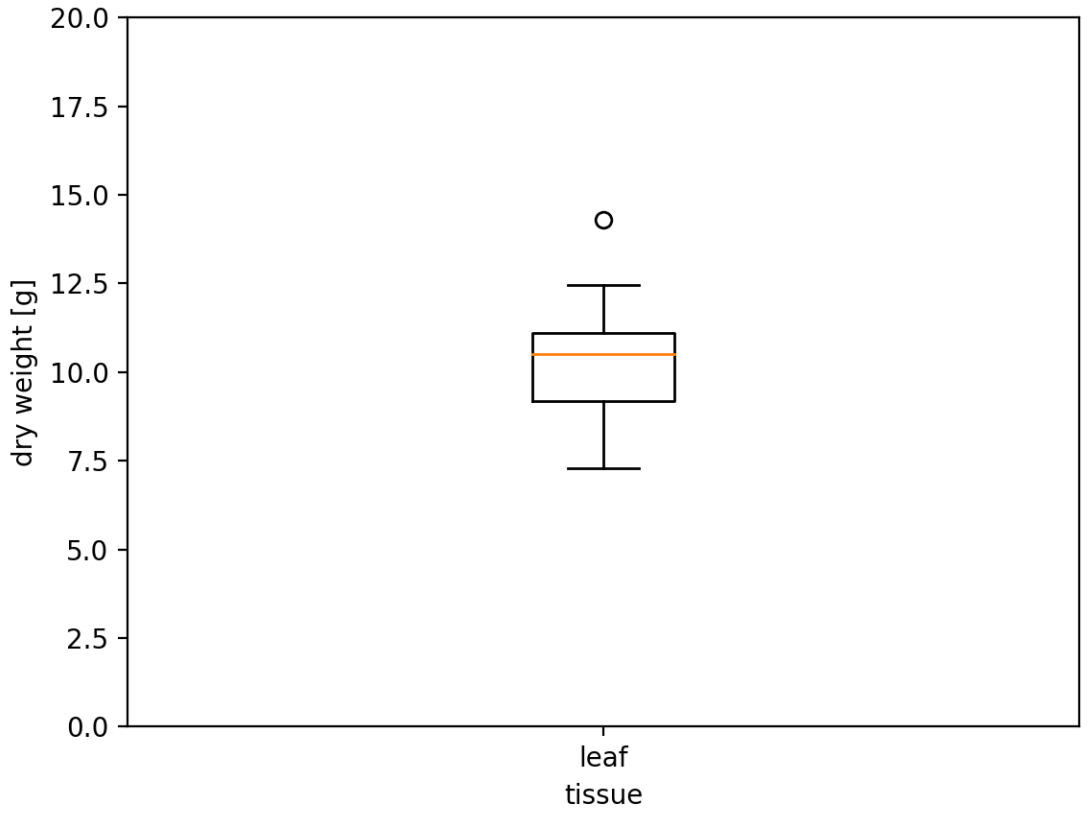
- 複数の 1 変量の連続値データを可視化するためのグラフ
- 最大値、最小値、第 1 四分位数 Q1、中央値、第 3 四分位数 Q3 などを簡単に確認できる
- [Q1 – 1.5IQR, Q3 + 1.5IQR] の外側にあるデータは外れ値

ボックスプロット



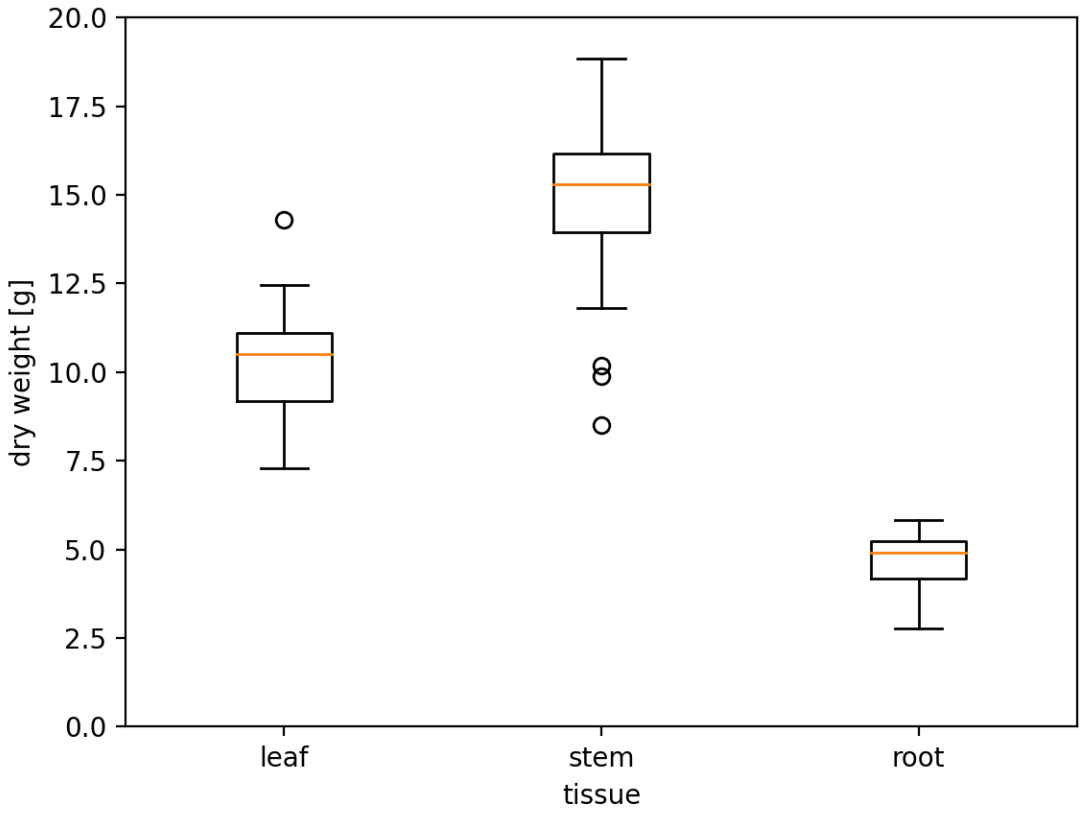
- 複数の 1 変量の連続値データを可視化するためのグラフ
- 最大値、最小値、第 1 四分位数 Q1、中央値、第 3 四分位数 Q3 などを簡単に確認できる
- [Q1 – 1.5IQR, Q3 + 1.5IQR] の外側にあるデータは外れ値
- 複数の変量の分布を同時に比べるときに便利

ボックスプロット



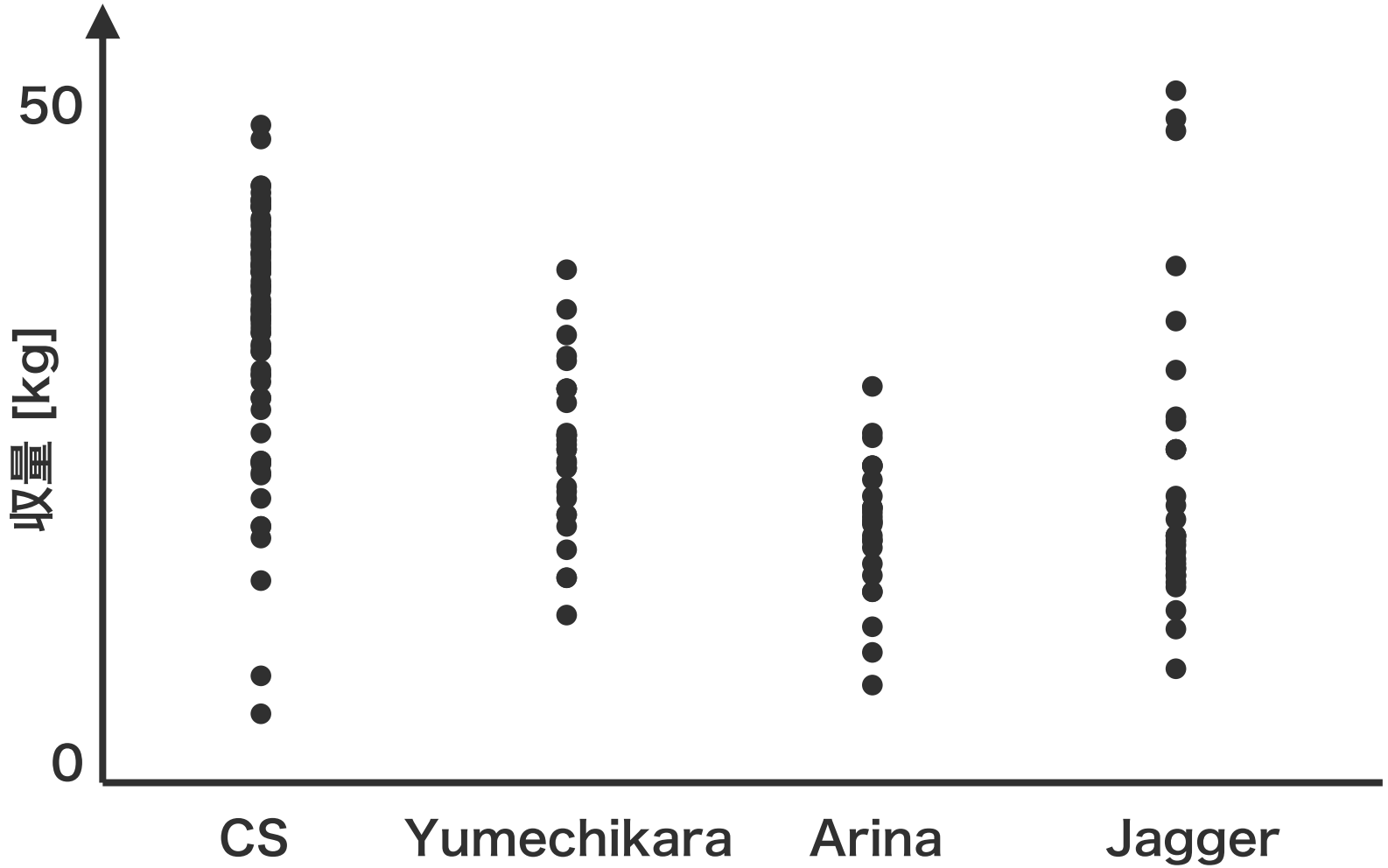

```
import numpy as  np
import matplotlib.pyplot as plt
np.random.seed(2018)

x1 = np.random.normal(10, 2, 20)



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


ax.boxplot([x1], labels=['leaf'])



ax.set_xlabel('tissue')
ax.set_ylabel('dry weight [g]')
ax.set_ylim(0, 20)
fig.show()
```

ボックスプロット



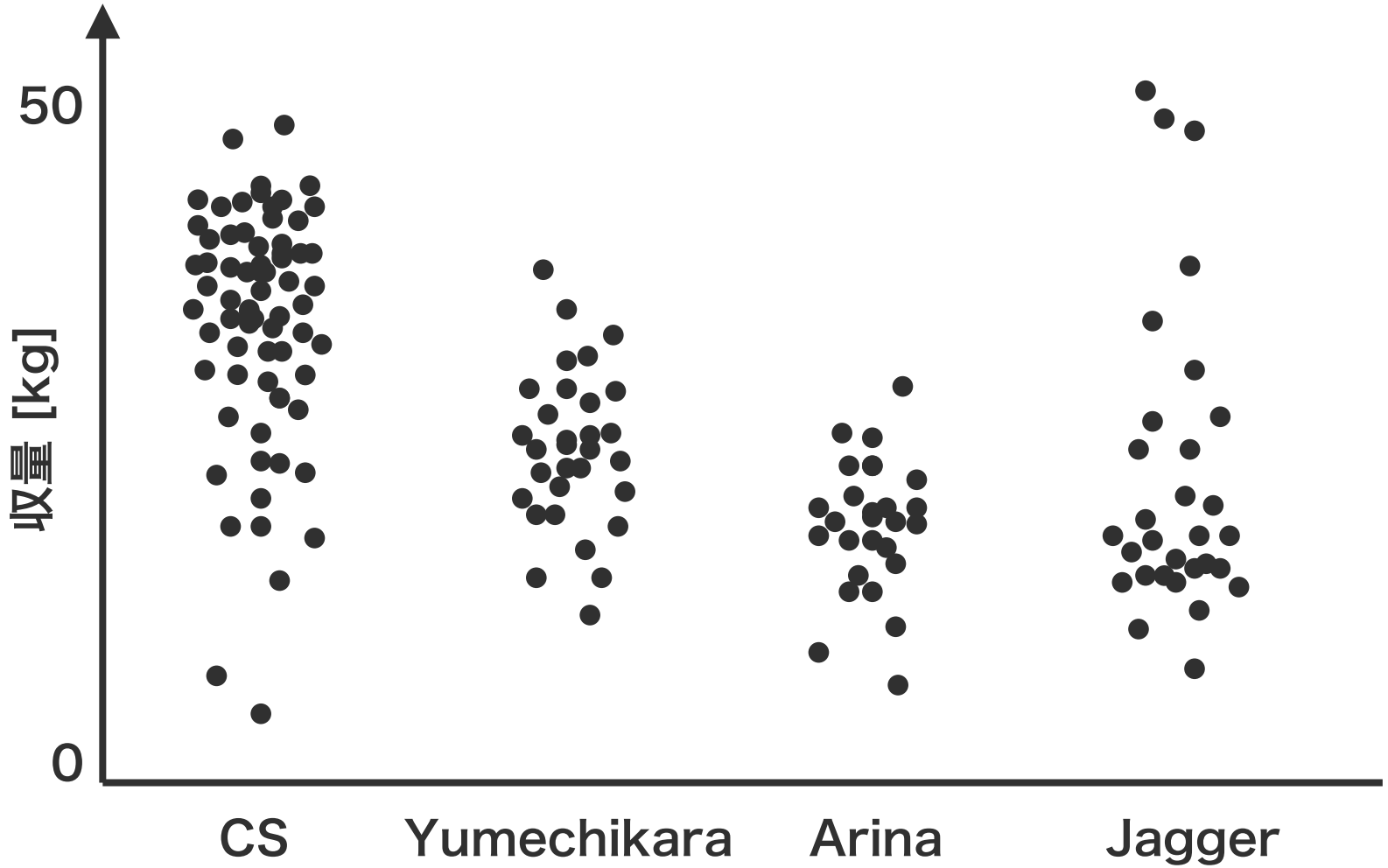

```
import numpy as  np
import matplotlib.pyplot as plt
np.random.seed(2018)

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()


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



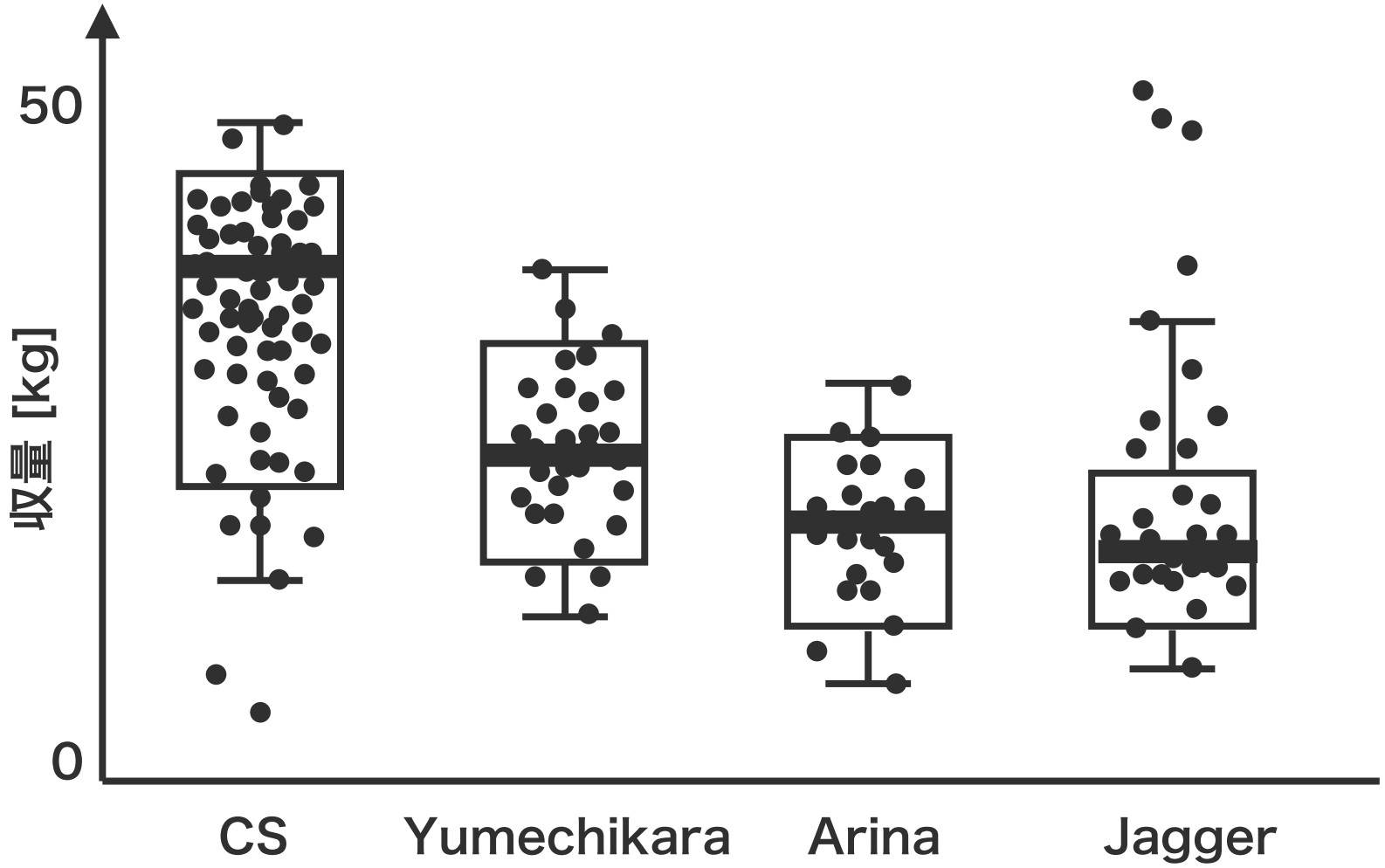
- 連続値データを可視化するためのグラフ、観測値をそのままプロットする
- データの多い所に点が重なるため、点を左右にランダムにずらしてプロットする
- ボックスプロットと合わせて使われる場合もある

jitter



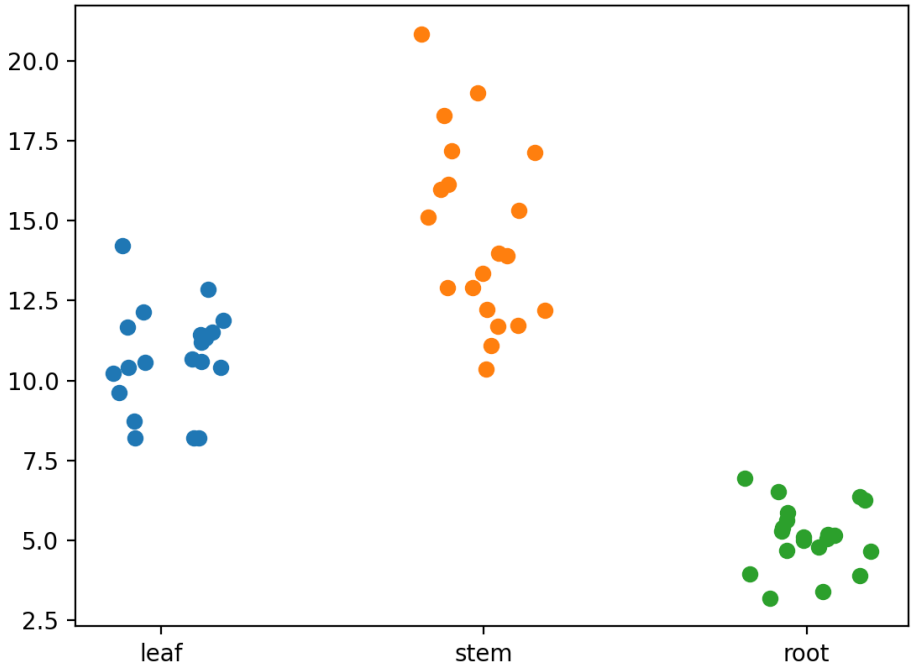
- 連続値データを可視化するためのグラフ、観測値をそのままプロットする
- データの多い所に点が重なるため、点を左右にランダムにずらしてプロットする
- ボックスプロットと合わせて使われる場合もある

jitter



- 連続値データを可視化するためのグラフ、観測値をそのままプロットする
- データの多い所に点が重なるため、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

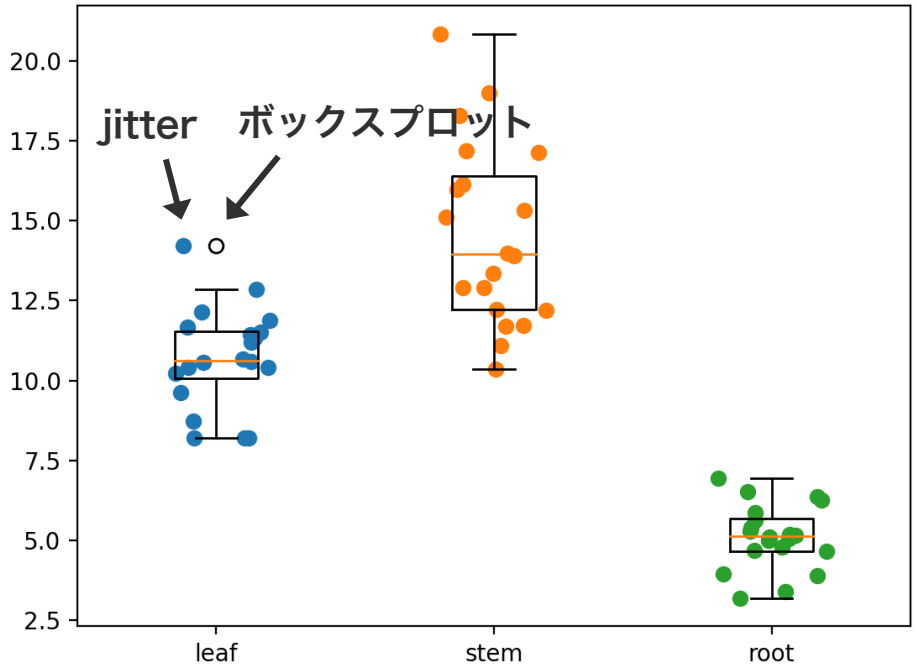
jitter




```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(112358)

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()
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()
```

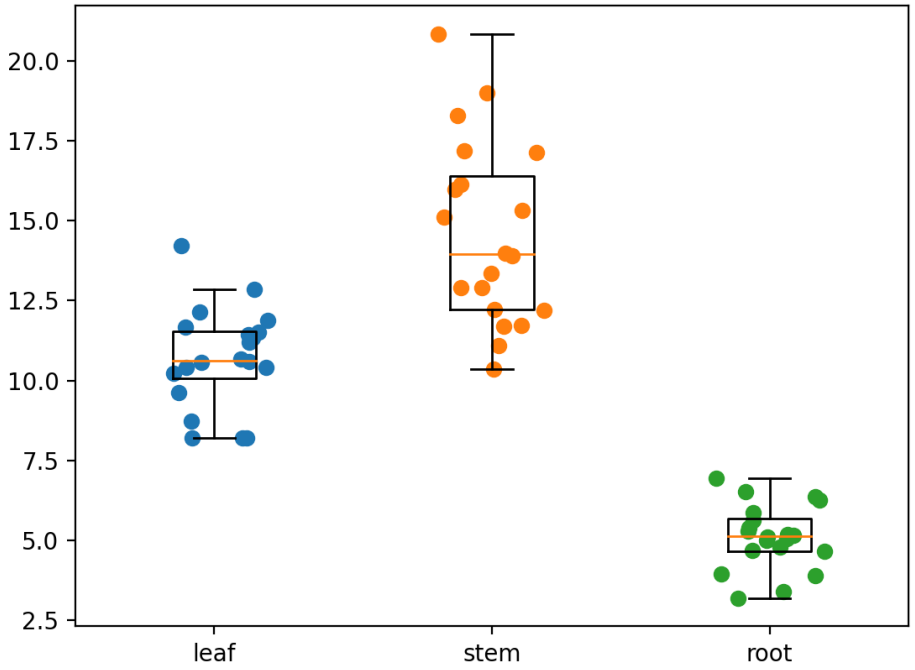
y1、y2、y3 のデータの y 座標をそのままにして、x 座標に乱数を加えて左右にずらす。

jitter




```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(112358)

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()
ax.scatter(x1, y1)
ax.scatter(x2, y2)
ax.scatter(x3, y3)
ax.boxplot([y1, y2, y3],
           labels=['leaf', 'stem', 'root'])
fig.show()
```

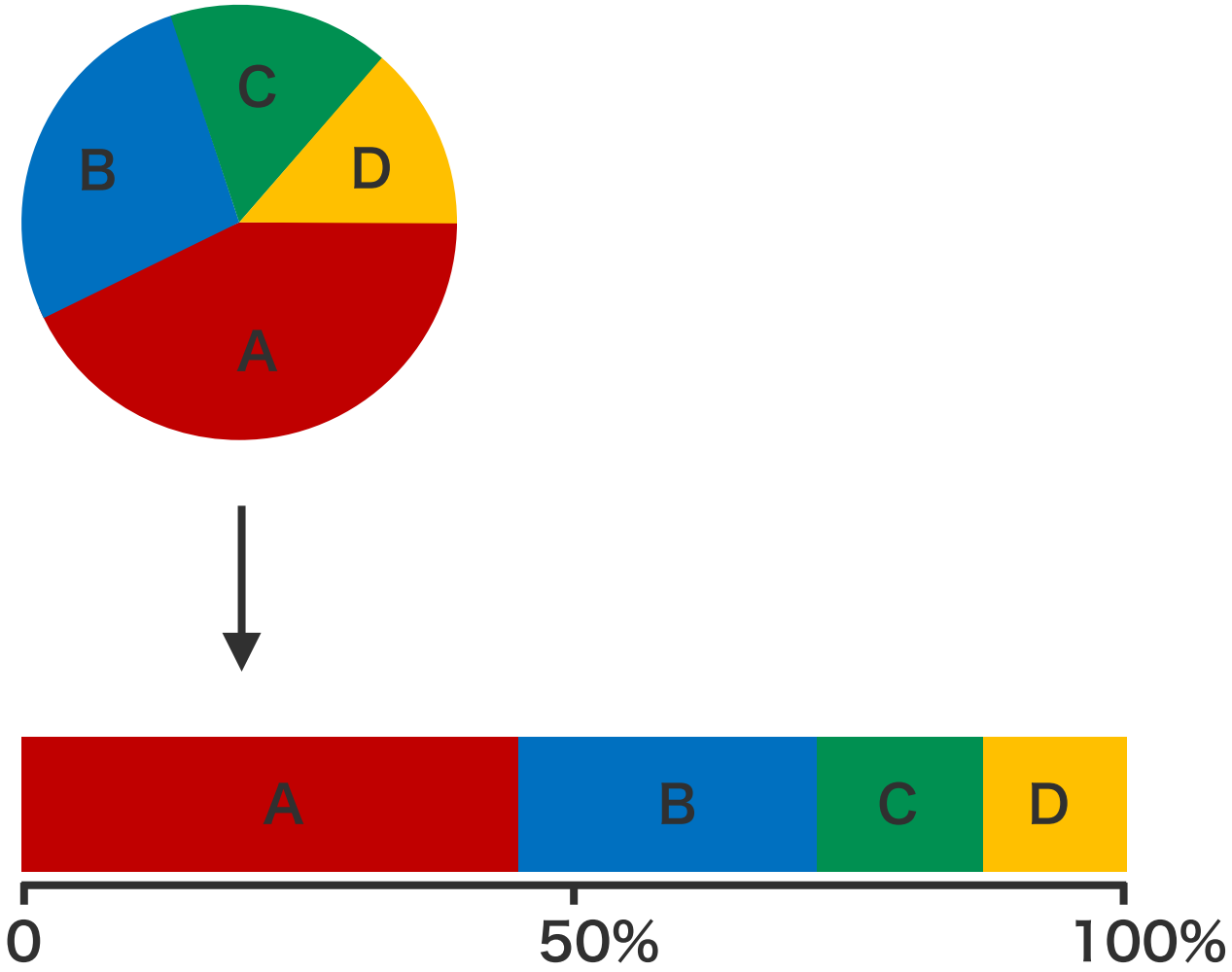
jitter




```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(112358)

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()
ax.scatter(x1, y1)
ax.scatter(x2, y2)
ax.scatter(x3, y3)
ax.boxplot([y1, y2, y3], showfliers=False,
           labels=['leaf', 'stem', 'root'])
fig.show()
```

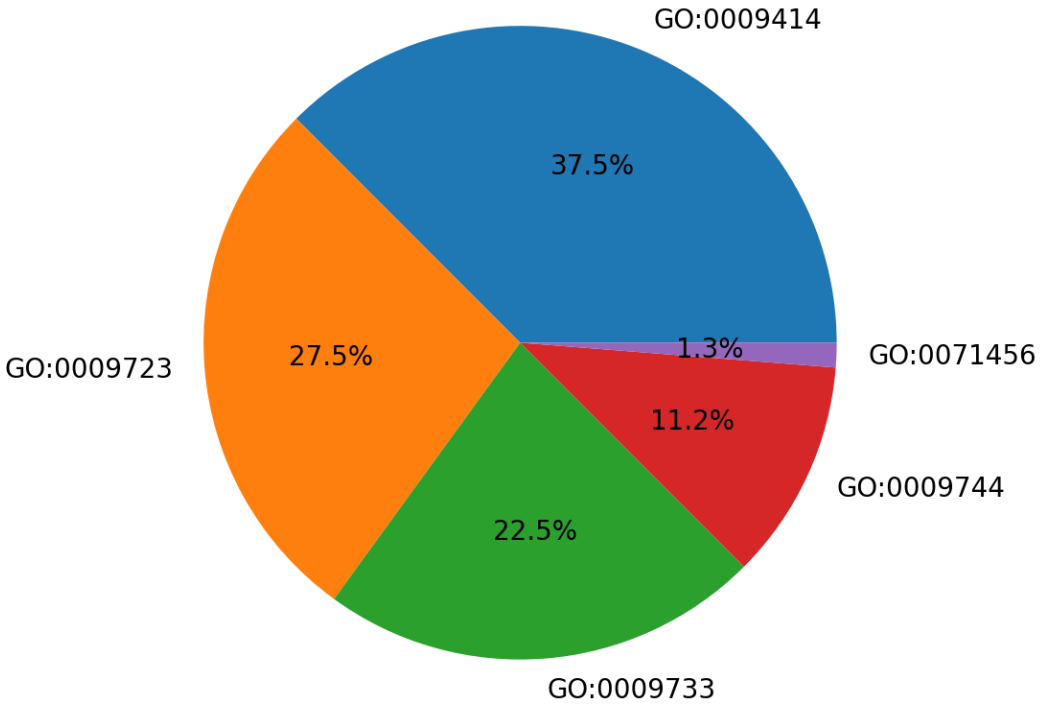
showfliers=False を指定することで、boxplot メソッドは外れ値を描かなくなる。

円グラフ



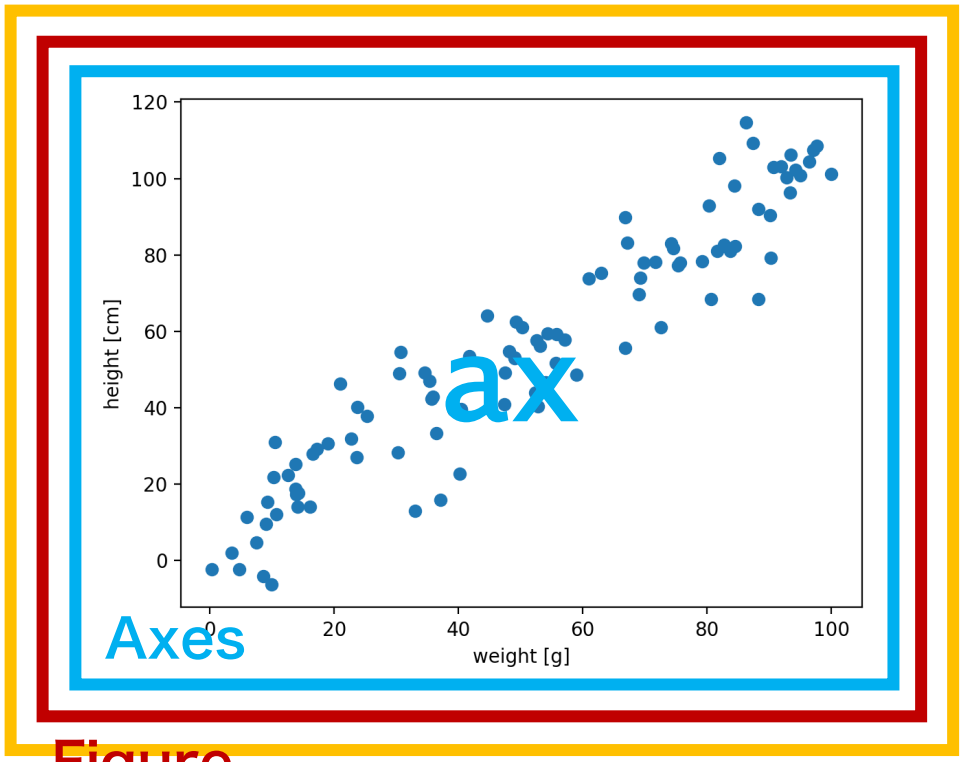
- 円グラフは誤解されやすいため、他のグラフで代替できる場合は、なるべく円グラフを使わない方が無難
- 人を騙す目的であれば、積極的に円グラフを用いるとよい
- さらに騙し効果を上げるために、3D 円グラフ、歪んだ円グラフあるいは合計が 100% にならない円グラフなどを用いるとよい

円グラフ



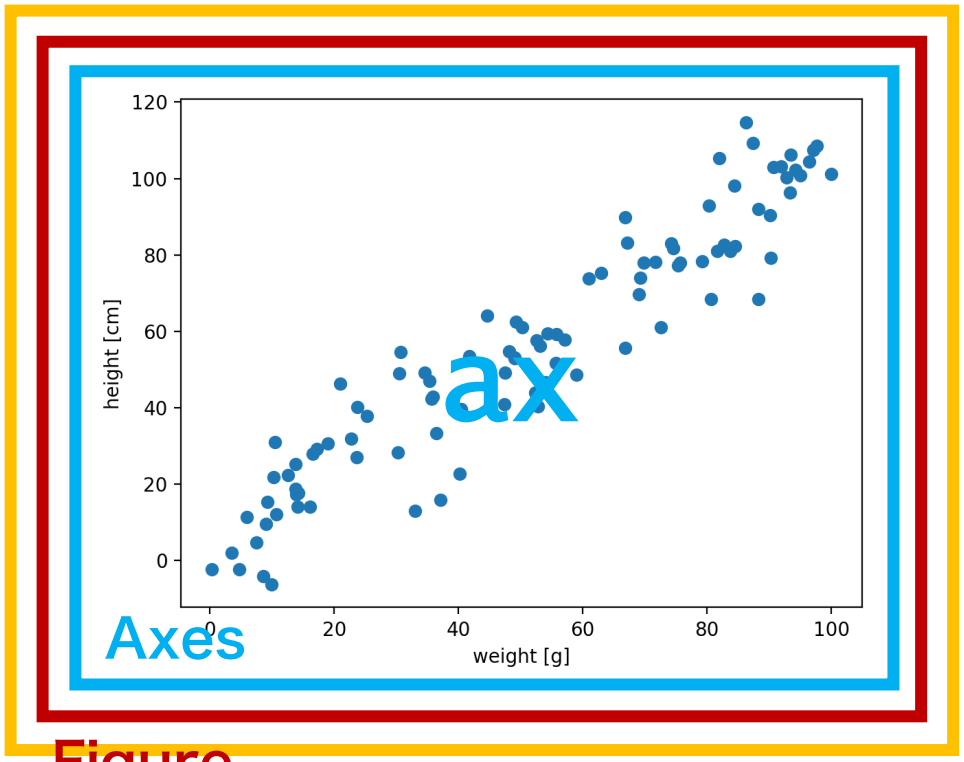

```
import matplotlib.pyplot as plt
import numpy as np

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()

ax.pie(y, labels=x, autopct="%1.1f%%")

ax.axis('equal')

fig.show()
```

subplot



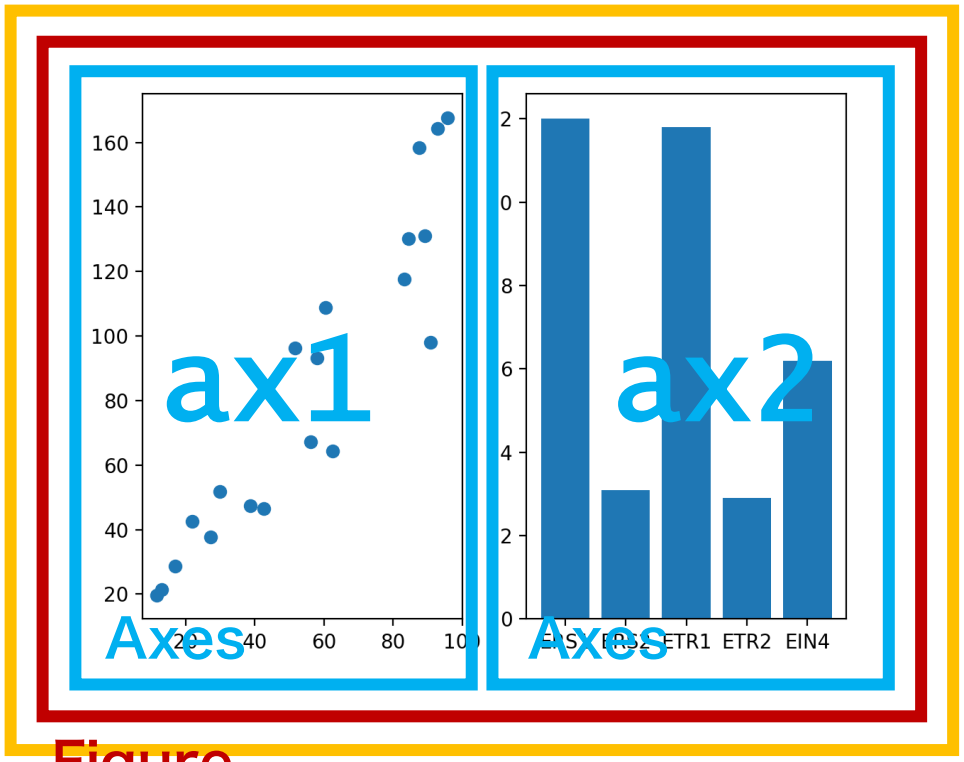

```
import matplotlib.pyplot as plt
import numpy as np

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()

ax.scatter(x, y)
ax.set_xlabel('weight [g]')
ax.set_ylabel('height [cm]')
fig.show()
```

subplot



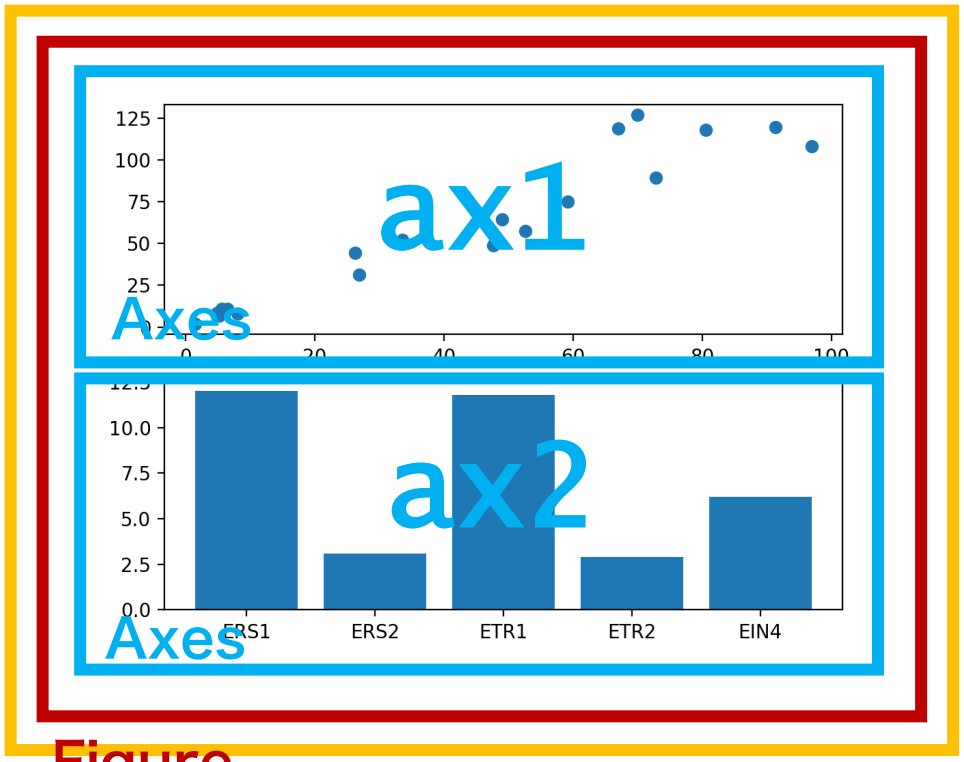

```
import matplotlib.pyplot as plt
import numpy as np

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()
```

subplot



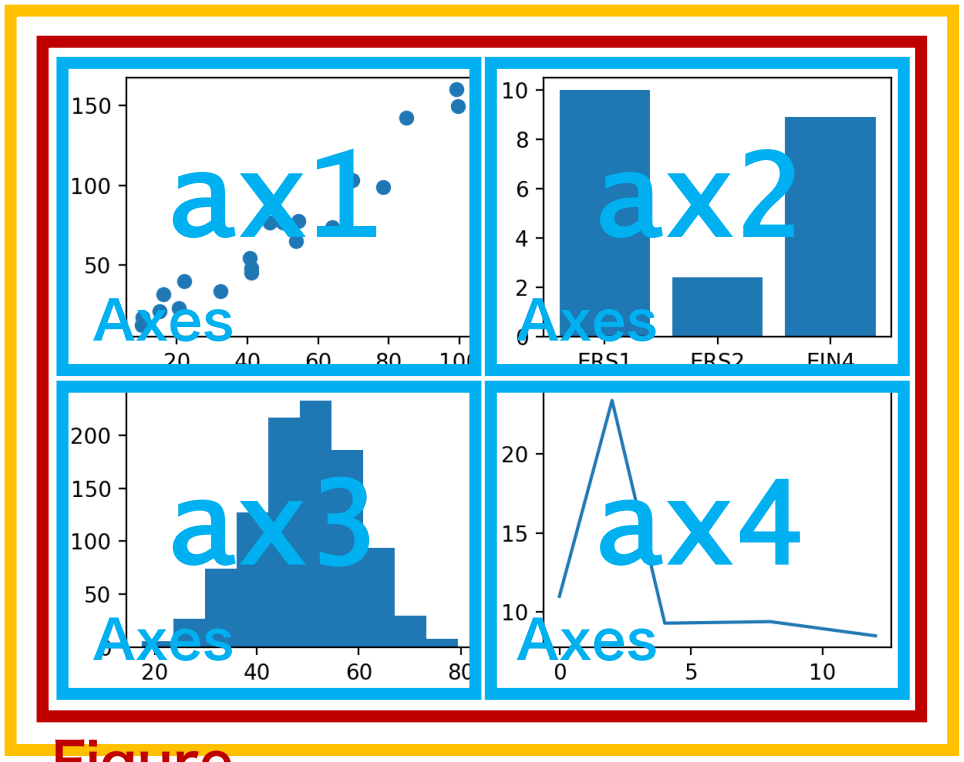

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
fig = plt.figure()
x1 = np.random.uniform(0, 100, 20)
y1 = x1 * np.random.uniform(1, 2, 20)

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()
```

subplot



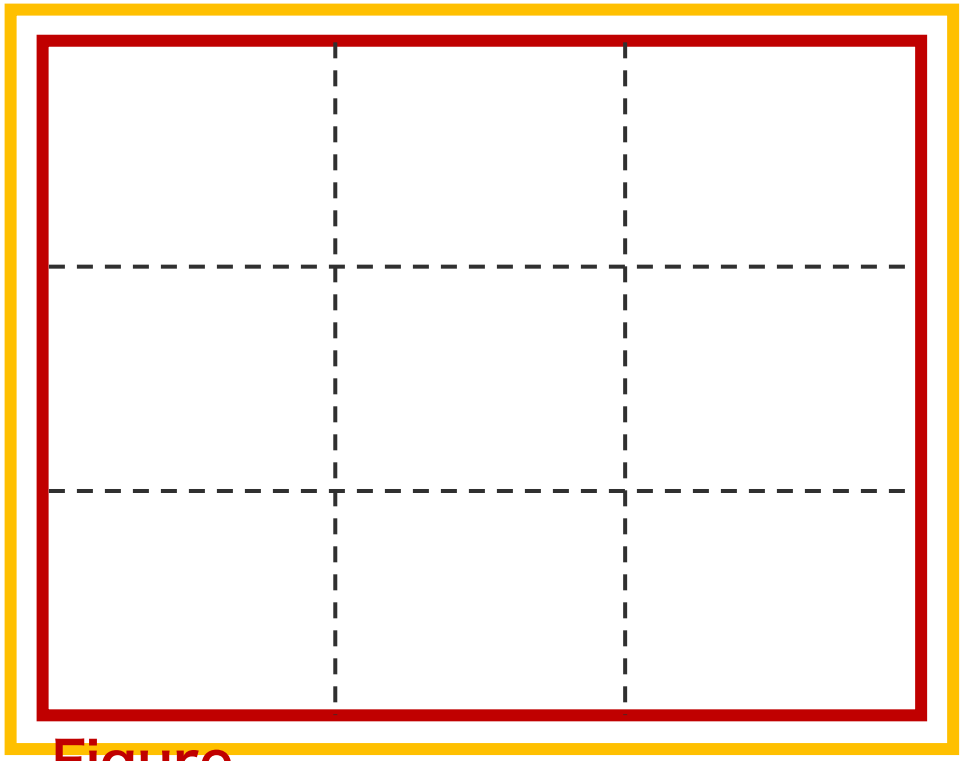

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
fig = plt.figure()
x1 = np.random.uniform(0, 100, 20)
y1 = x1 * np.random.uniform(1, 2, 20)

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()
```

subplot




```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
x1 = np.random.uniform(0, 100, 20)
y1 = x1 * np.random.uniform(1, 2, 20)
x2 = np.array(['ERS1', 'ERS2', 'EIN4'])
y2 = np.array([10.0, 2.4, 8.9])
x3 = np.random.normal(50, 10, 1000)
x4 = np.array([0, 2, 4, 8, 12])
y4 = np.array([11.0, 23.4, 9.3, 9.4, 8.5])
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()
```

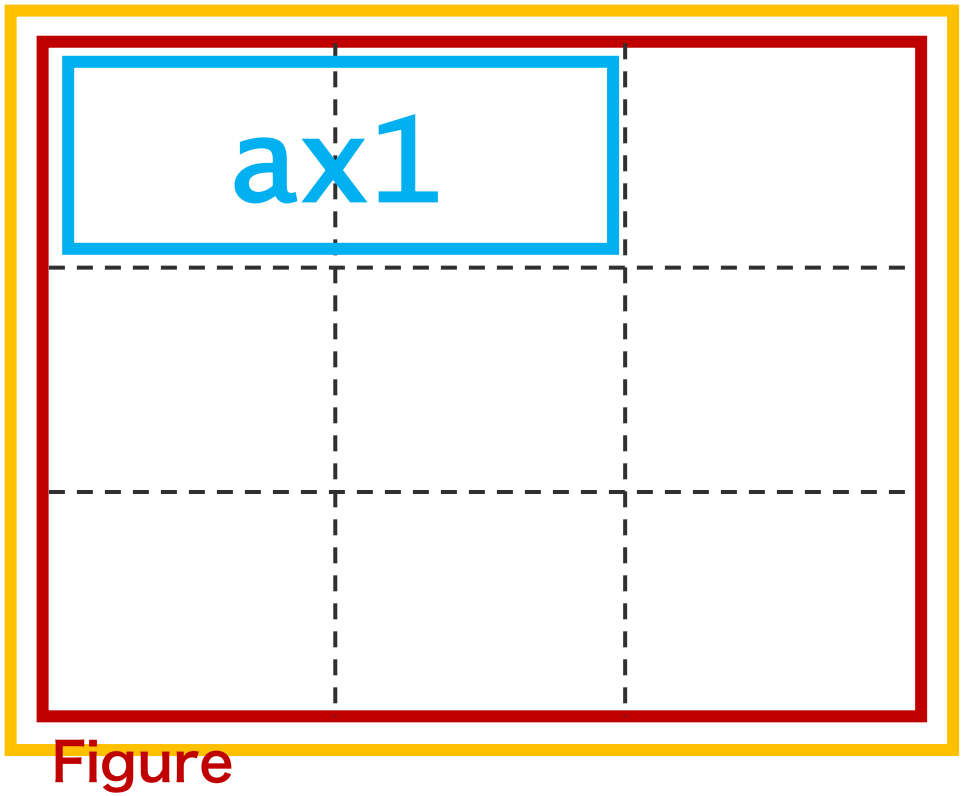
gridspec



gridspec を使用すると、描画領域をグリッド状に分けたのちに、隣接するグリッド同士を結合させることによって、描画領域を任意の形に分割できる。

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

fig = plt.figure()
gs = fig.add_gridspec(3, 3)
```

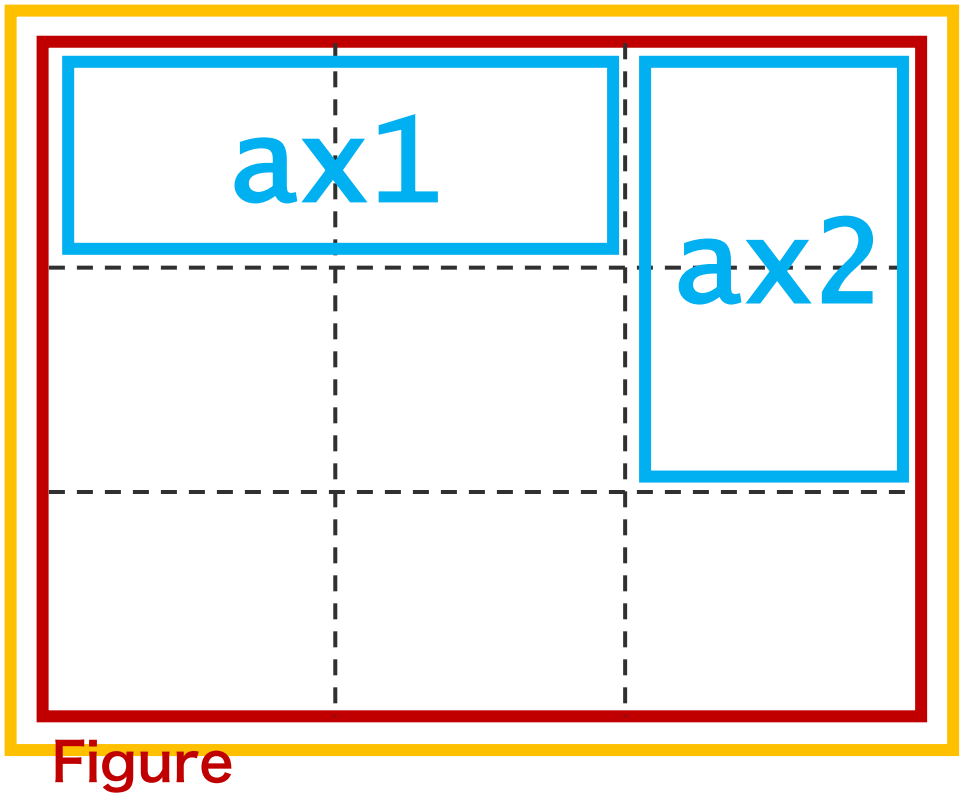
gridspec



gridspec を使用すると、描画領域をグリッド状に分けたのちに、隣接するグリッド同士を結合させることによって、描画領域を任意の形に分割できる。

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

fig = plt.figure()
gs = fig.add_gridspec(3, 3)

ax1 = fig.add_subplot(gs[0, 0:2])
```

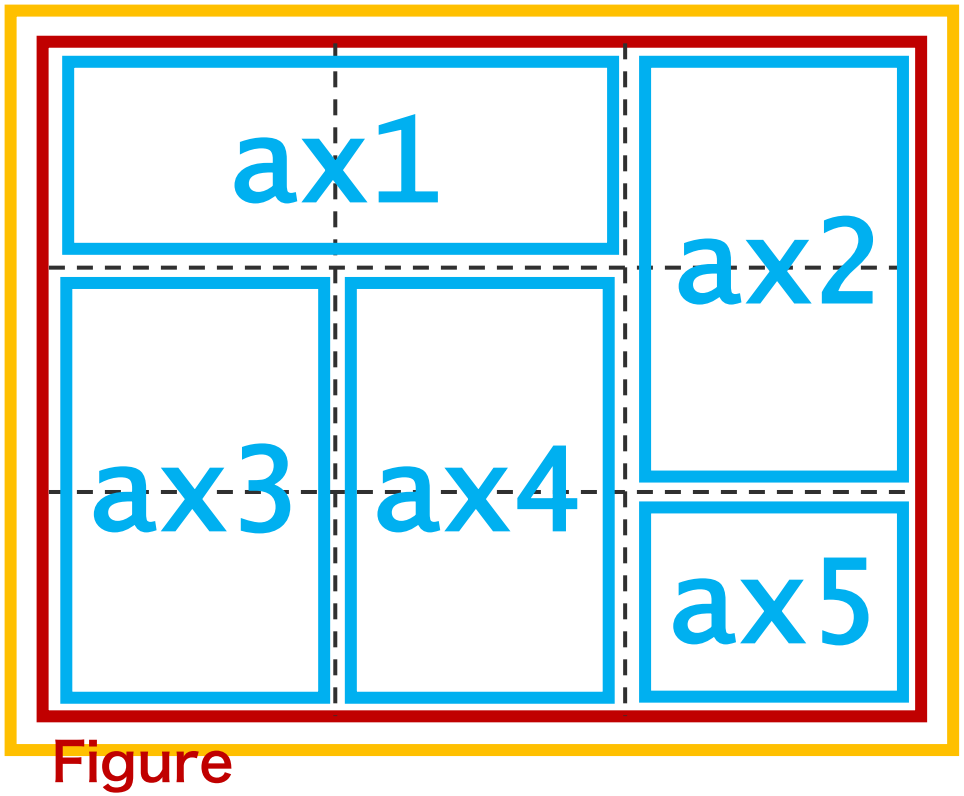
gridspec




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

fig = plt.figure()
gs = fig.add_gridspec(3, 3)

ax1 = fig.add_subplot(gs[0, 0:2])
ax2 = fig.add_subplot(gs[0:2, 2])
```

gridspec を使用すると、描画領域をグリッド状に分けたのちに、隣接するグリッド同士を結合させることによって、描画領域を任意の形に分割できる。

gridspec



gridspec を使用すると、描画領域をグリッド状に分けたのちに、隣接するグリッド同士を結合させることによって、描画領域を任意の形に分割できる。

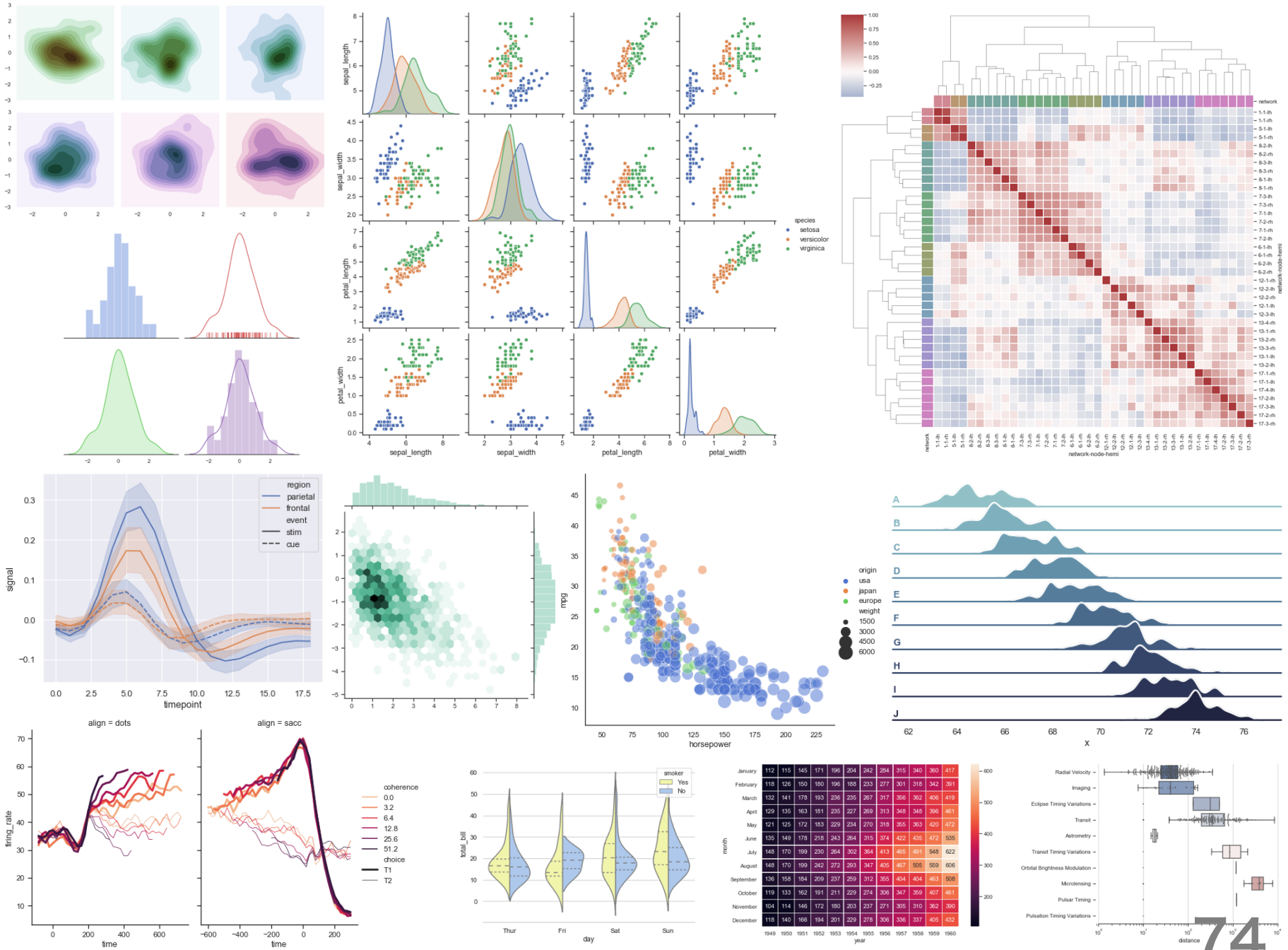
```
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

fig = plt.figure()
gs = fig.add_gridspec(3, 3)

ax1 = fig.add_subplot(gs[0, 0:2])
ax2 = fig.add_subplot(gs[0:2, 2])
ax3 = fig.add_subplot(gs[1:3, 0])
ax4 = fig.add_subplot(gs[1:3, 1])
ax5 = fig.add_subplot(gs[2, 2])

ax1.plot([0, 1], [10, 20])
ax2.plot([0, 1], [10, 20])
ax3.plot([0, 1], [10, 20])
ax4.plot([0, 1], [10, 20])
ax5.plot([0, 1], [10, 20])

plt.tight_layout()
plt.show()
```

可視化

- matplotlib
- seaborn

seaborn

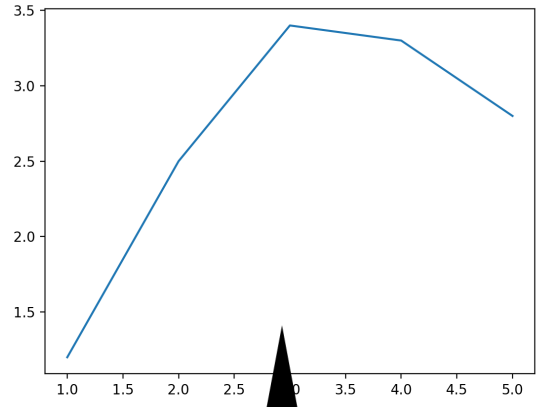
Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.



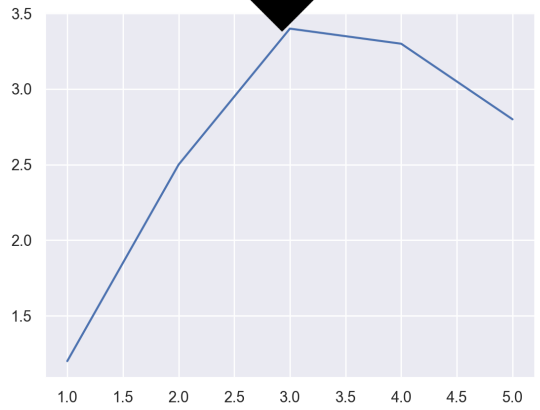
seaborn スタイルシート

matplotlib グラフのカラーテーマ（スタイルシート）を seaborn 風に変更する場合、seaborn をインポートしたのち、seaborn.set() を実行する。

matplotlib



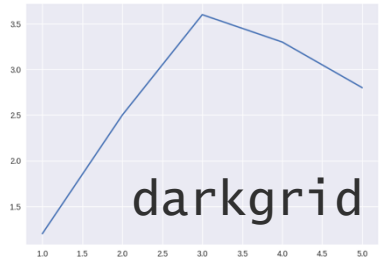
seaborn

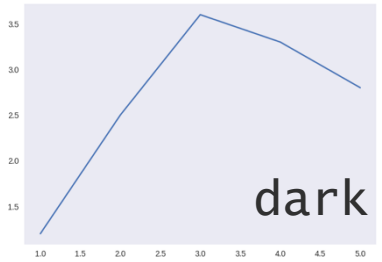


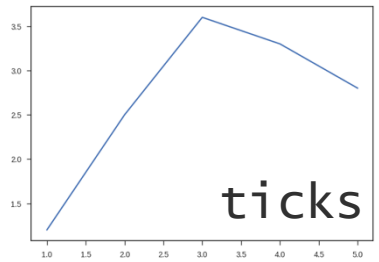

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()


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()
```

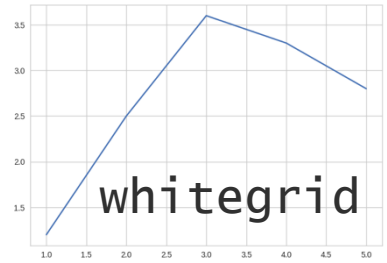
seaborn スタイルシート

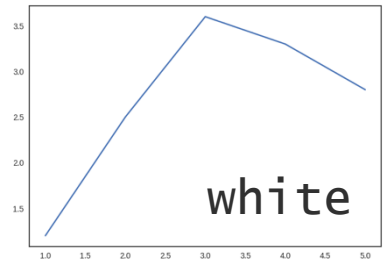
seaborn には 5 種類の基本スタイルシートが用意されている。set_style 関数で、スタイルシートを変更できる。











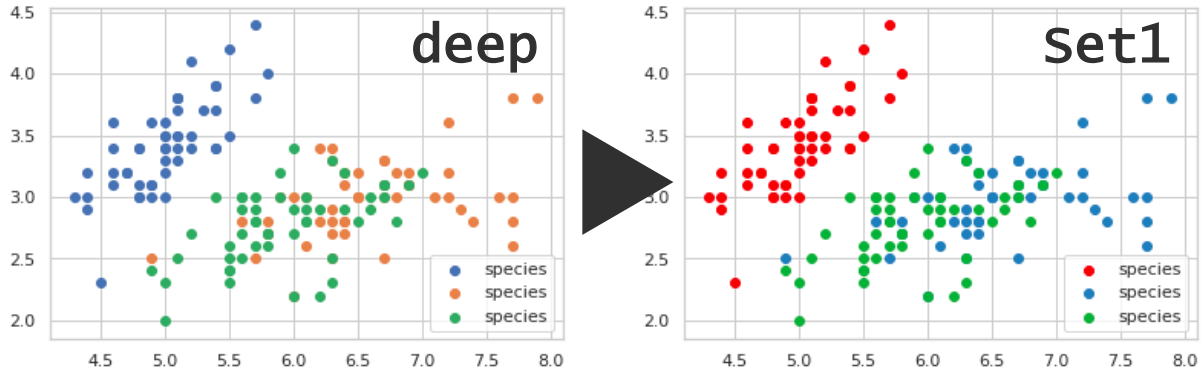

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
sns.set_style('whitegrid')


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

グラフの色の組み合わせはカラーパレットと呼ばれている。set_palette 関数でカラーパレットを変更できる。

deep

Set1

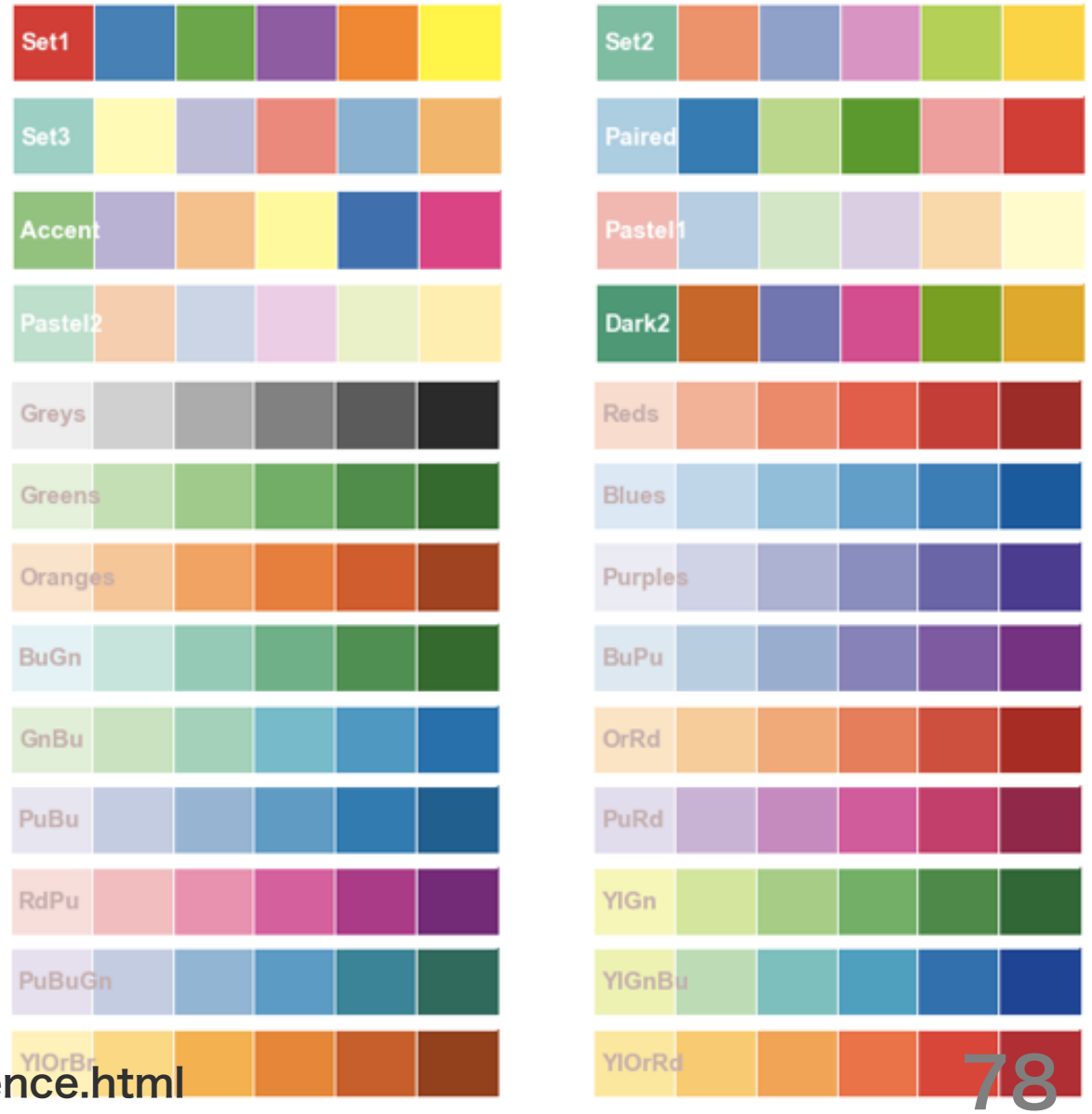



```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
sns.set_style('whitegrid')
sns.set_palette('Set1')

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

seaborn の set_palette で指定できるカラーパレットは、matplotlib の colormaps で定義されている。
詳細は、matplotlib のウェブサイトを参照。

deep

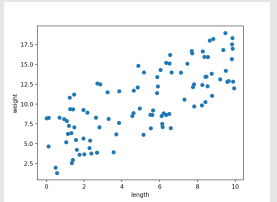


https://seaborn.pydata.org/tutorial/color_palettes.html
https://matplotlib.org/examples/color/colormaps_reference.html

seaborn

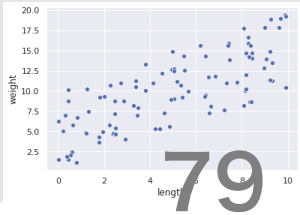
matplotlib

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



x = np.random.uniform(0, 10, 100)
y = x + 10 * np.random.uniform(0, 1, 100)



fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.scatter(x, y)
ax.set_xlabel('length')
ax.set_ylabel('weight')

fig.show()
```



seaborn

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

x = np.random.uniform(0, 10, 100)
y = x + 10 * np.random.uniform(0, 1, 100)

df = pd.DataFrame({'length': x,
                   'weight': y})

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax = sns.scatterplot(x='length',
                     y='weight',
                     data=df, ax=ax)

fig.show()
```

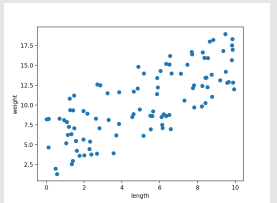


seaborn

matplotlib

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


x = np.random.uniform(0, 10, 100)
y = x + 10 * np.random.uniform(0, 1, 100)



fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax.scatter(x, y)
ax.set_xlabel('length')
ax.set_ylabel('weight')

fig.show()
```

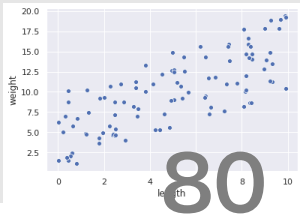
x 座標と y 座標を用意して可視化関数に直接代入してグラフを描く。



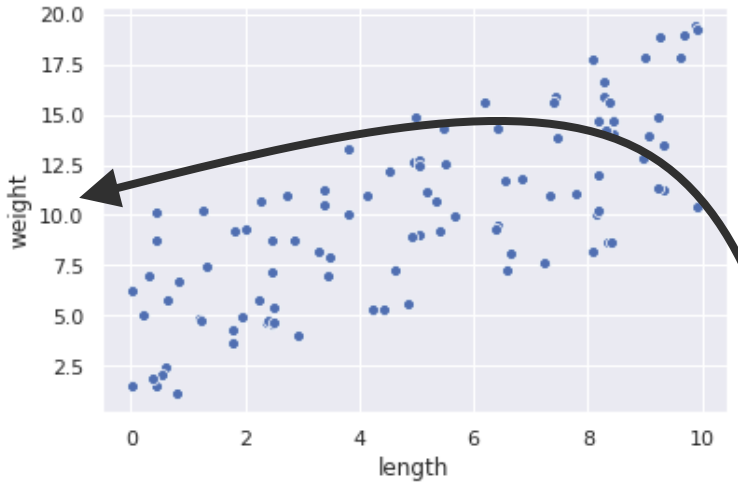
seaborn

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

x = np.random.uniform(0, 10, 100)
y = x + 10 * np.random.uniform(0, 1, 100)

df = pd.DataFrame({'length': x,
                   'weight': y})

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax = sns.scatterplot(x='length',
                     y='weight',
                     data=df, ax=ax)

fig.show()
```

データをデータフレーム型で用意し可視化関数に代入する。関数の中で x 軸と y 軸を指定してグラフを描く。



散布図



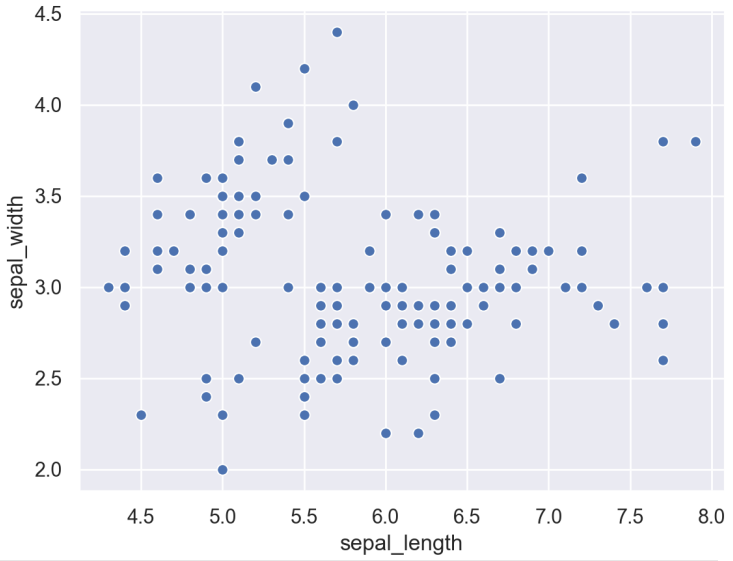
y='weight'

x='length'

length	weight
5.4	11.1
2.8	7.5
1.5	0.7
8.5	14.2
:	:

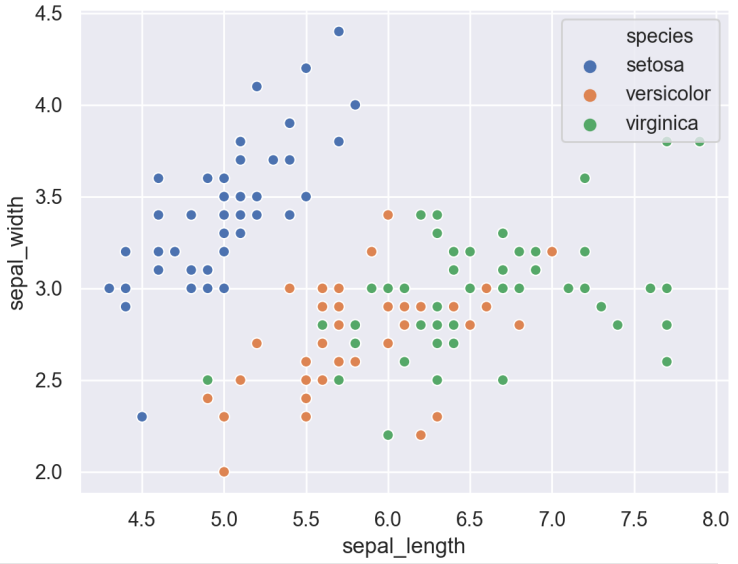
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

a = np.random.uniform(0, 10, 100)
b = a + 10 * np.random.uniform(0, 1, 100)

df = pd.DataFrame({'length': a,
                   'weight': b})

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax = sns.scatterplot(x='length',
                     y='weight',
                     data=df, ax=ax)

fig.show()
```

散布図




```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

df = sns.load_dataset("iris")
df.head()
#    sepal_length  sepal_width  petal_length  petal_width species
# 0           5.1          3.5           1.4          0.2  setosa
# 1           4.9          3.0           1.4          0.2  setosa
# 2           4.7          3.2           1.3          0.2  setosa
# 3           4.6          3.1           1.5          0.2  setosa
# 4           5.0          3.6           1.4          0.2  setosa

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

ax = sns.scatterplot(x='sepal_length', y='sepal_width',
                     data=df, ax=ax)
plt.show()
```

散布図



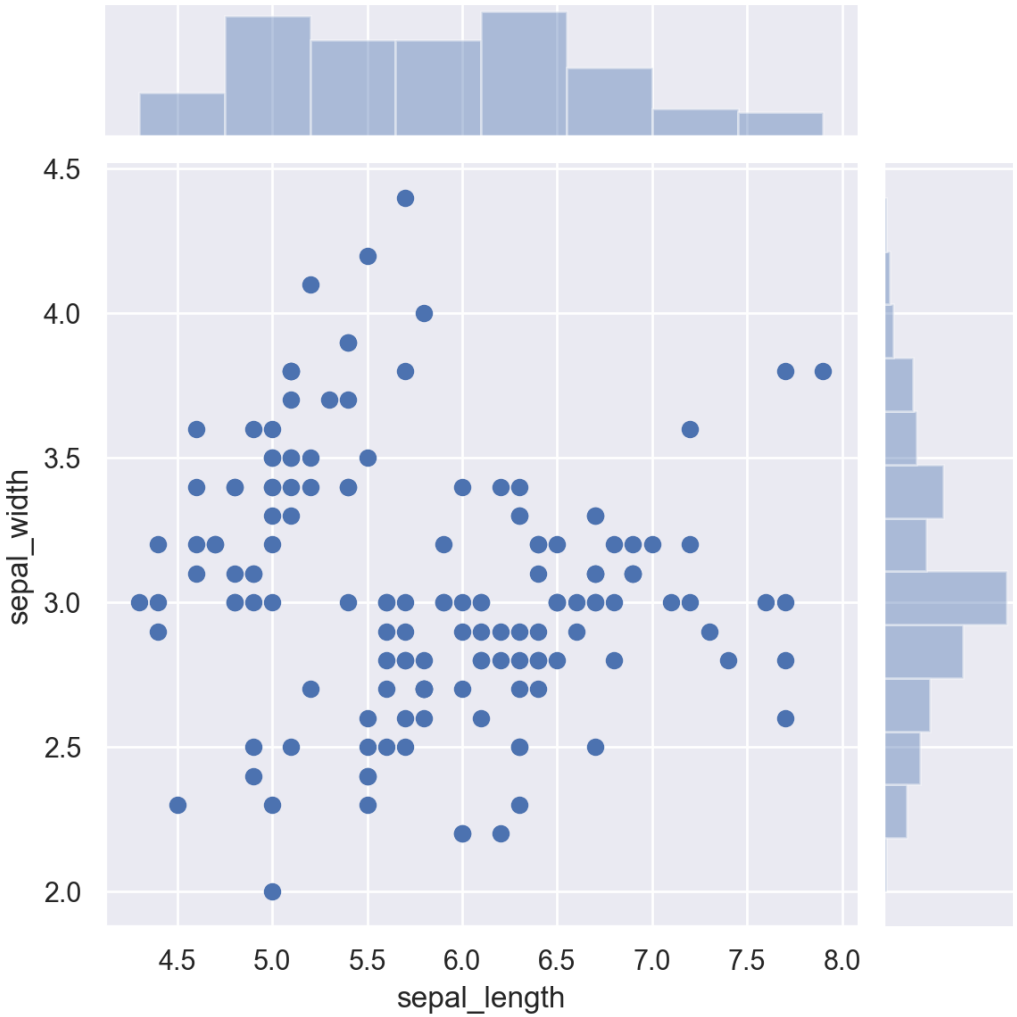

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

df = sns.load_dataset("iris")
df.head()
#    sepal_length  sepal_width  petal_length  petal_width species
# 0           5.1          3.5           1.4          0.2  setosa
# 1           4.9          3.0           1.4          0.2  setosa
# 2           4.7          3.2           1.3          0.2  setosa
# 3           4.6          3.1           1.5          0.2  setosa
# 4           5.0          3.6           1.4          0.2  setosa

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

ax = sns.scatterplot(x='sepal_length', y='sepal_width', hue='species',
                     data=df, ax=ax)
plt.show()
```

seaborn 可視化関数

基本的なグラフを描く関数は seaborn でも用意されている。これらの関数の、seaborn と matplotlib の対応は以下の表のようになっている。

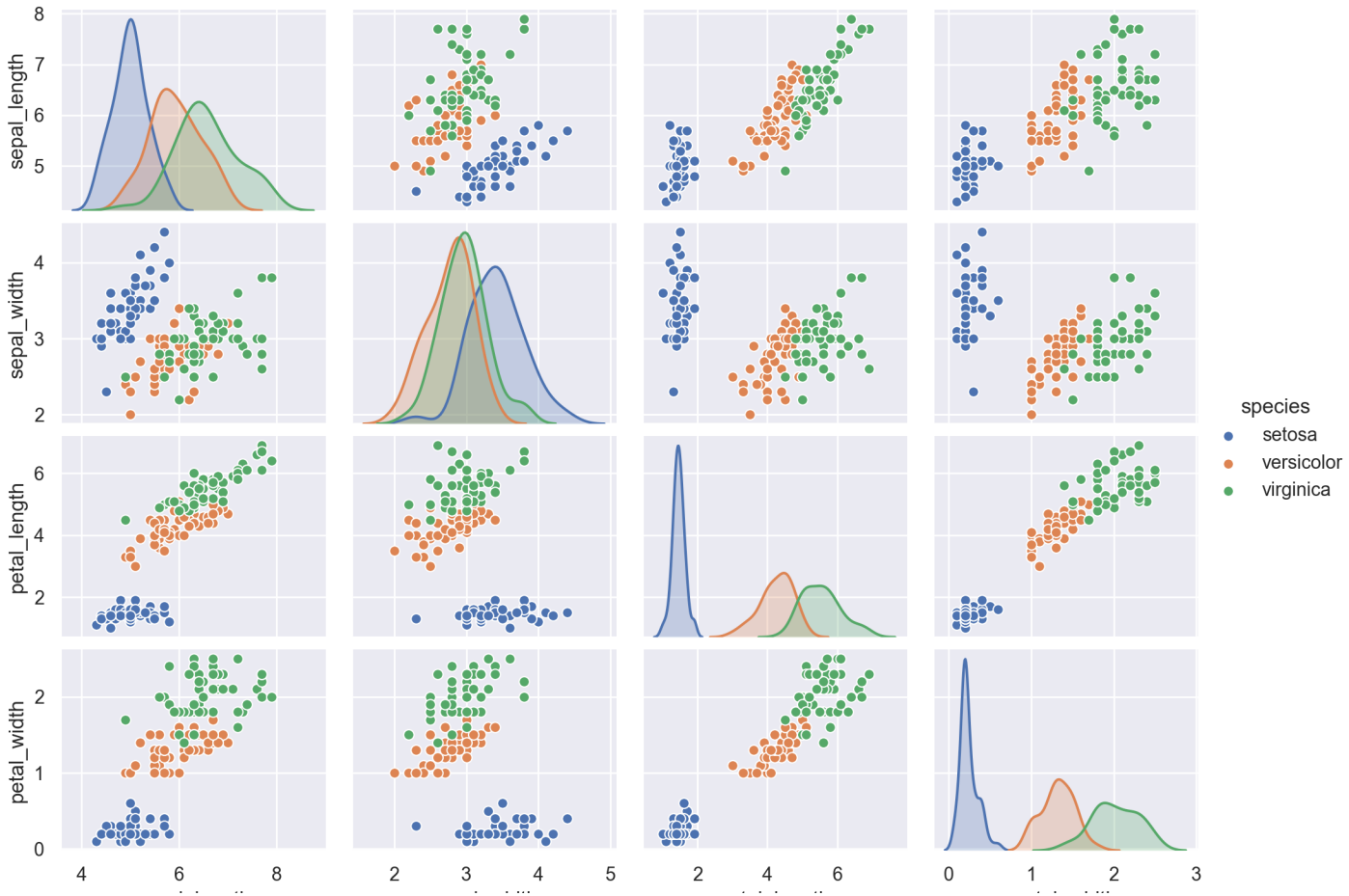
グラフ	matplotlib	seaborn
線グラフ	`ax.plot`	`sns.lineplot`
散布図	`ax.scatter`	`sns.scatterplot`
棒グラフ	`ax.bar`	`sns.barplot`
ヒストグラム	`ax.hist`	`sns.distplot`
ボックスプロット	`ax.boxplot`	`sns.boxplot`
ヒートマップ	`ax.imshow` `ax.pcolor`	`sns.heatmap` `sns.clustermap`

ヒストグラム付き散布図



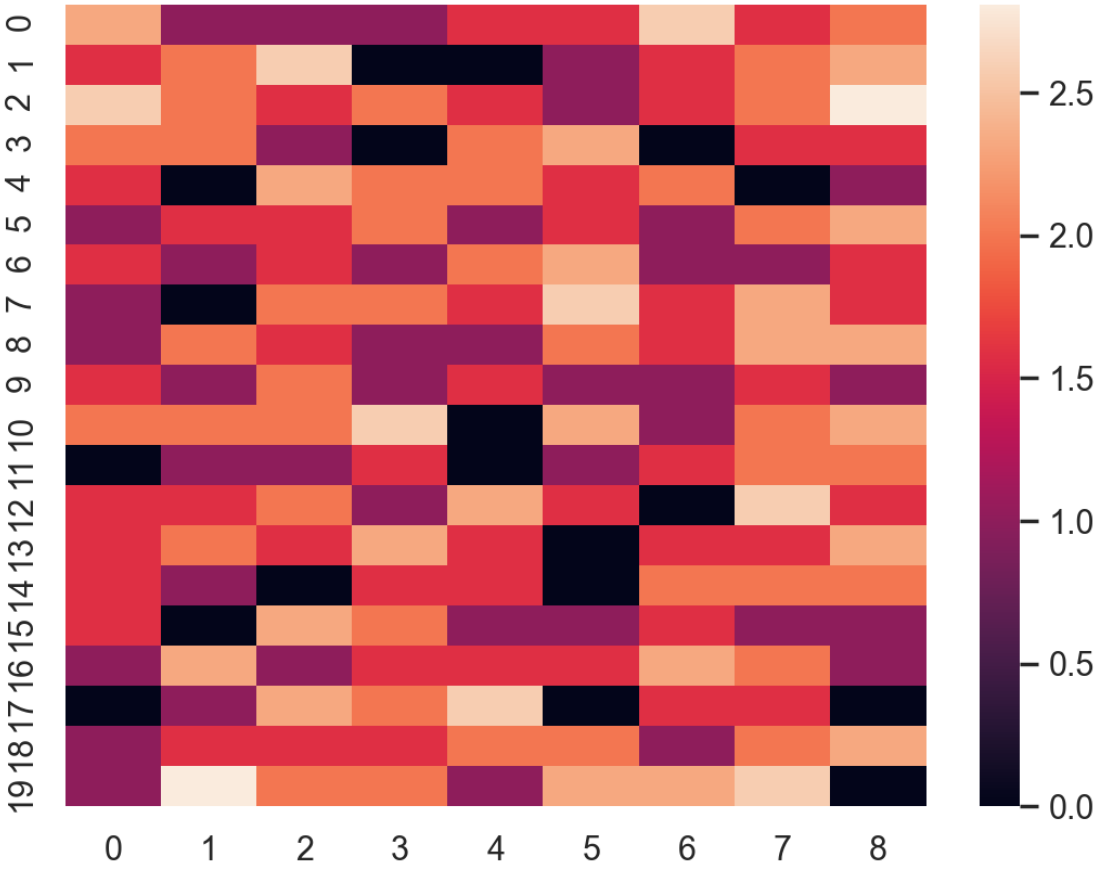

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

df = sns.load_dataset("iris")

sns.jointplot(x='sepal_length',
                y='sepal_width', data=df)
plt.show()
```

ペアプロット



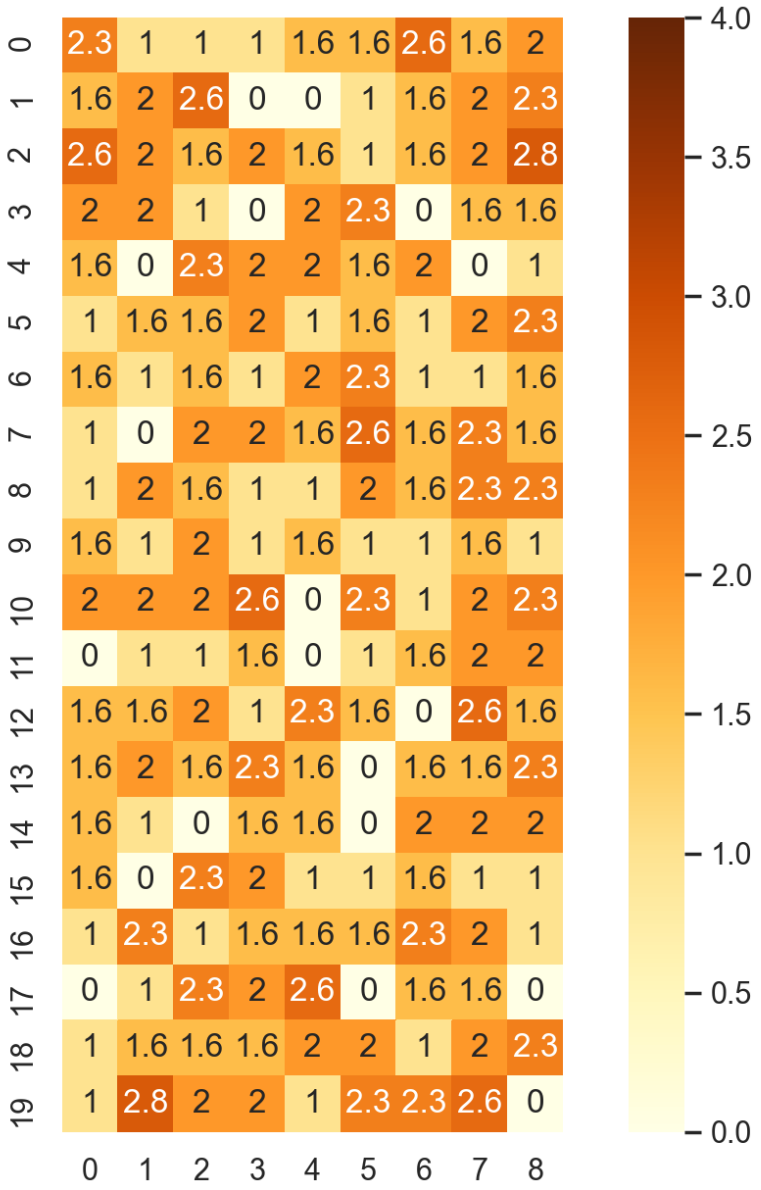

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

df = sns.load_dataset("iris")
df.head()

sns.pairplot(df, hue='species')
plt.show()
```

ヒートマップ



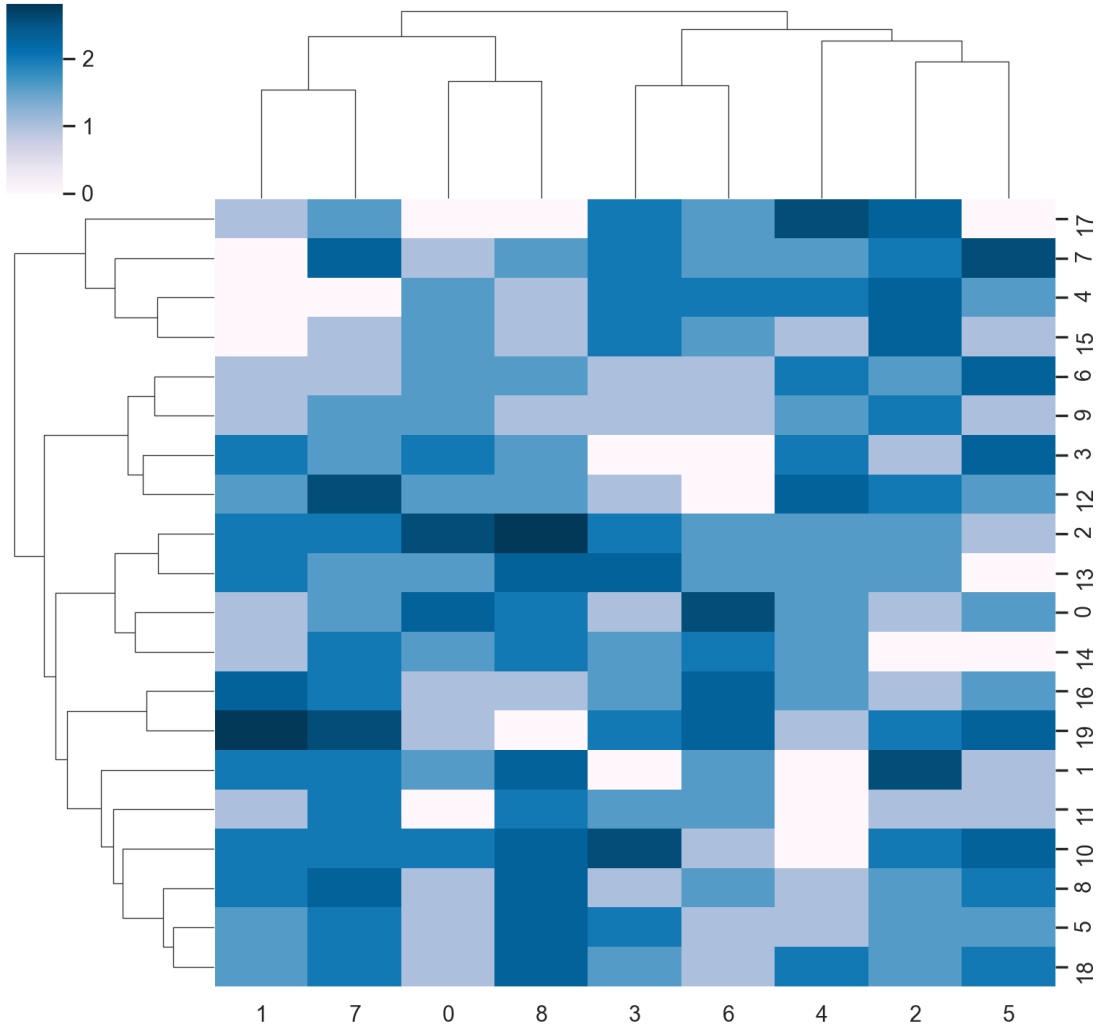

```
import numpy as np
import seaborn as sns

np.random.seed(12345)


x = np.random.binomial(100, 0.02, 180)
x = x. reshape((20, 9))
x = np.log2(x + 1)

sns.heatmap(x)




plt.show()
```

ヒートマップ



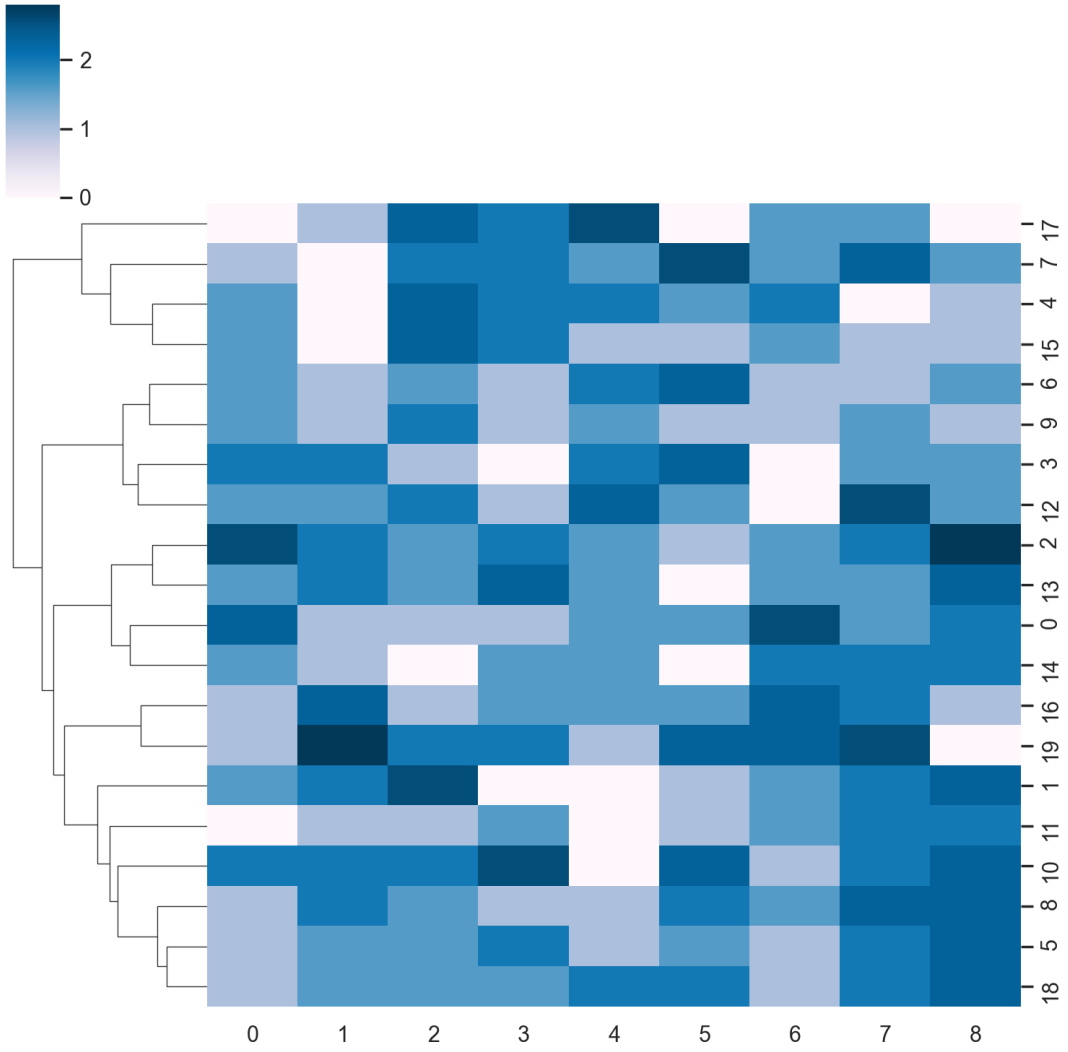

```
import numpy as np
import seaborn as sns

np.random.seed(12345)


x = np.random.binomial(100, 0.02, 180)
x = x. reshape((20, 9))
x = np.log2(x + 1)

sns.heatmap(x, annot=True,
               square=True,
               cmap='YlOrBr',
               vmin = 0, vmax = 4)



plt.show()
```

ヒートマップ




```
import numpy as np
import seaborn as sns

np.random.seed(12345)


x = np.random.binomial(100, 0.02, 180)
x = x. reshape((20, 9))
x = np.log2(x + 1)

sns.clustermap(x,
                    cmap='PuBu',
                    method='ward',
                    metric='euclidean')



plt.show()
```

ヒートマップ




```
import numpy as np
import seaborn as sns

np.random.seed(12345)


x = np.random.binomial(100, 0.02, 180)
x = x. reshape((20, 9))
x = np.log2(x + 1)

sns.clustermap(x,
                    cmap='PuBu',
                    method='ward',
                    metric='euclidean',
                    col_cluster=False)


plt.show()
```

農学生命情報科学特論Ⅰ

4

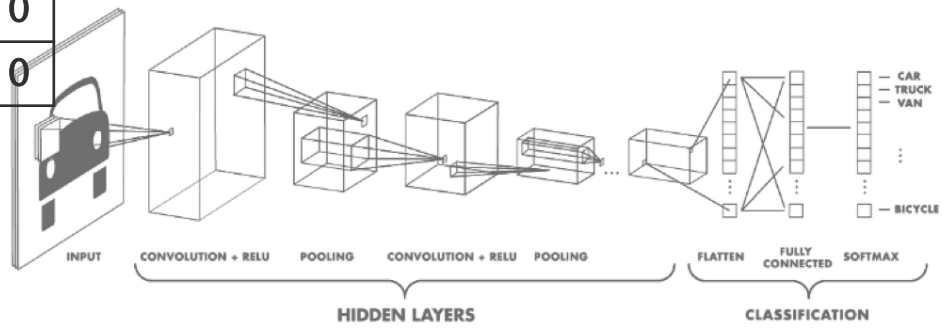
- 可視化
- 深層学習研究事例

塩基配列の one-hot encoding

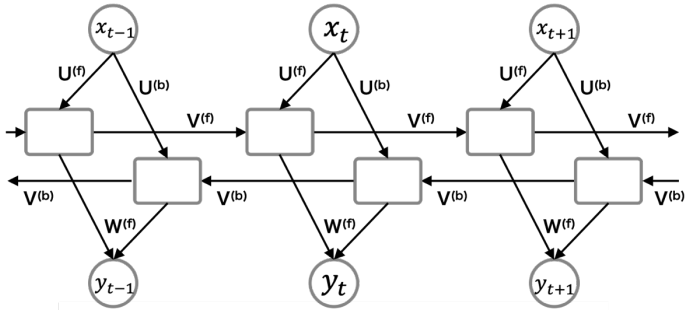
入力　AGTCATTGCA

1	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	0	0	1	0
0	1	0	0	0	0	0	1	0	0
0	0	1	0	0	1	1	0	0	0

CNN → 出力

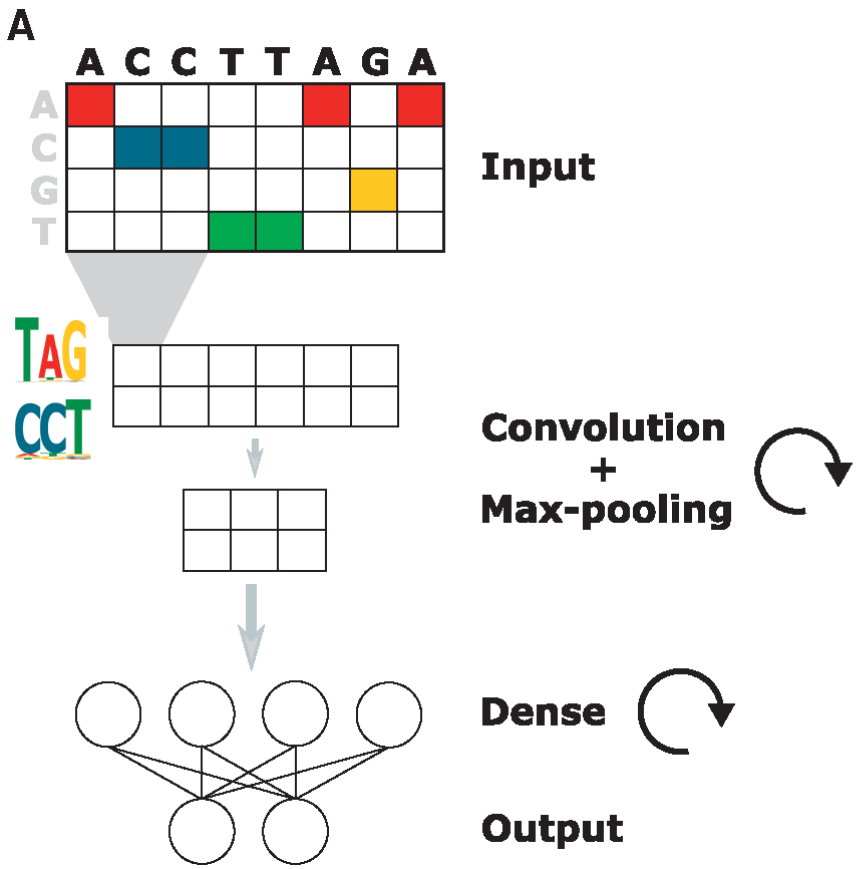


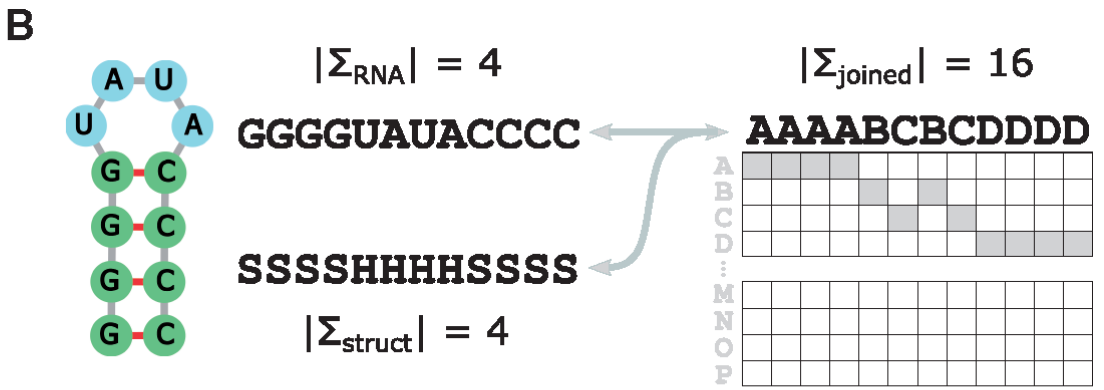
A G T C A T T G C A → RNN → 出力

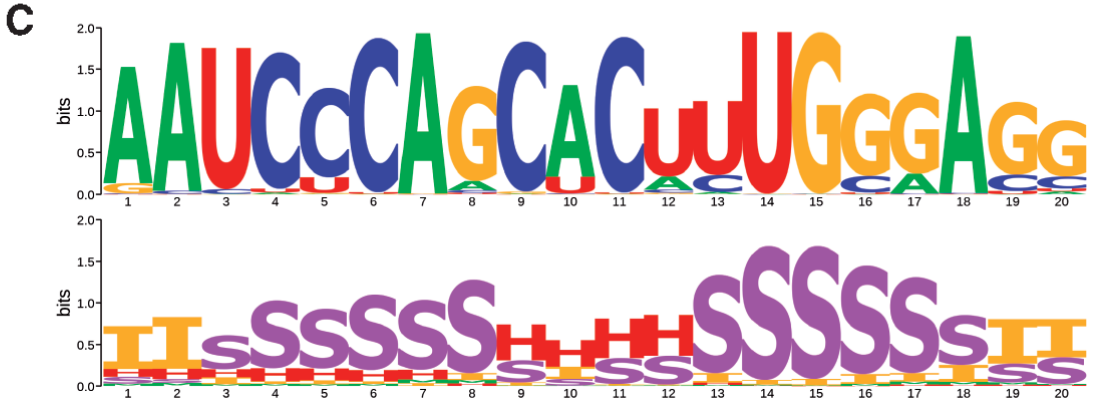


研究事例 / genome sequences

pysster: classification of biological sequences by learning sequence and structure motifs with convolutional neural networks. Bioinformatics 2018, 17(1):3035–3037
https://doi.org/10.1093/bioinformatics/bty222

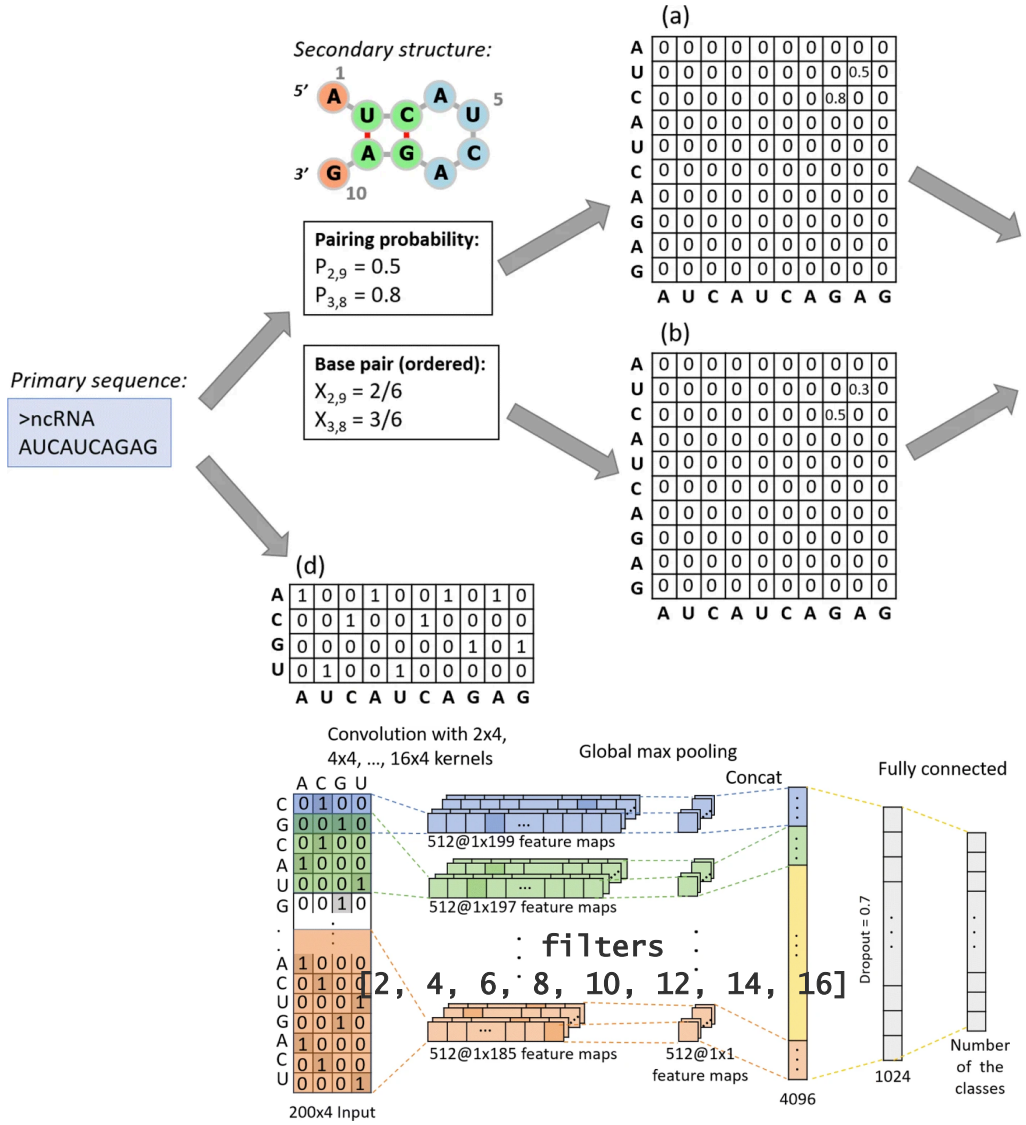


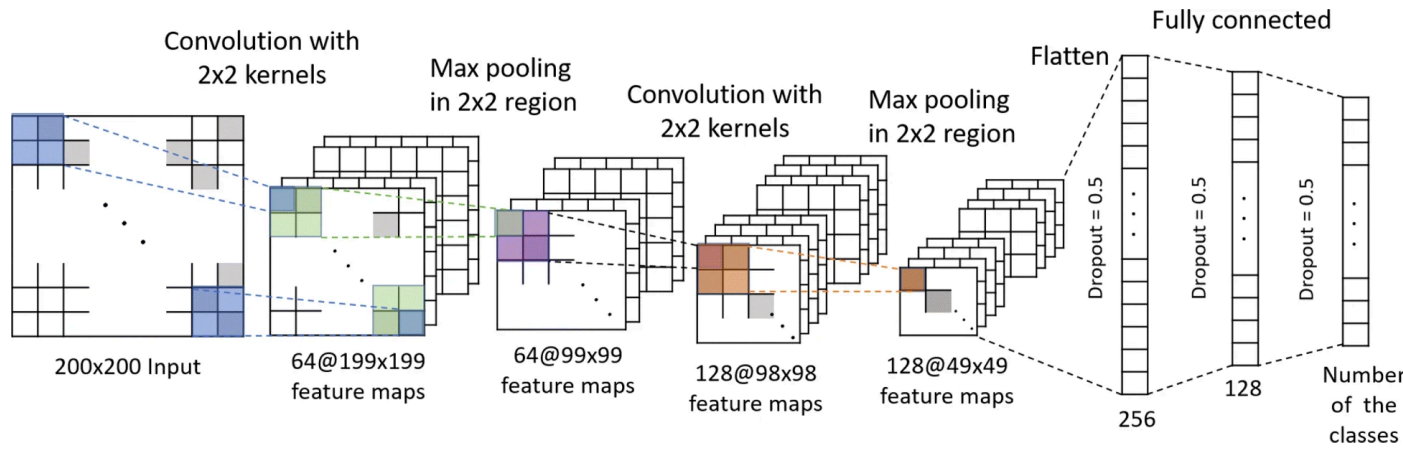




研究事例 / genome sequences

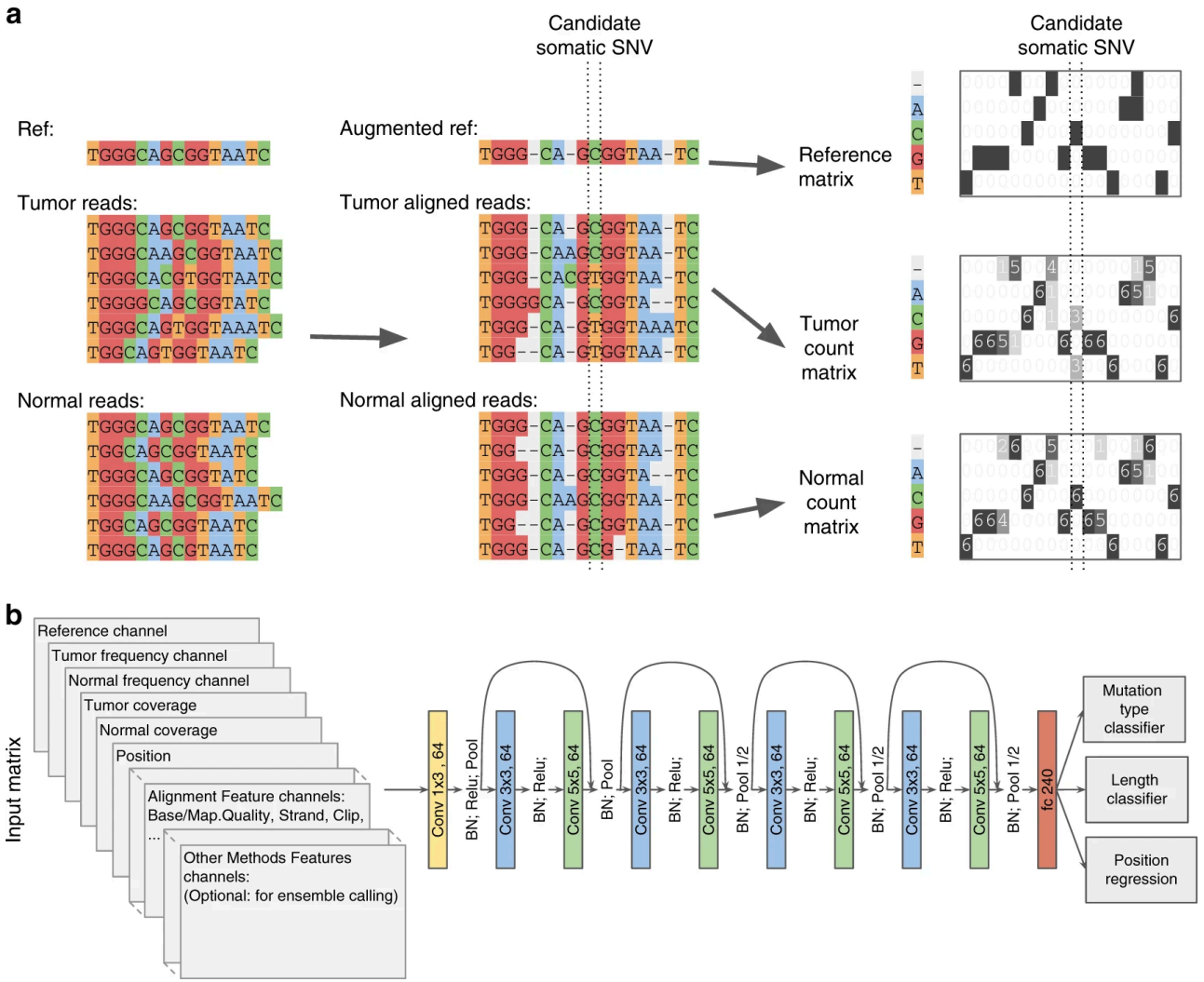
Fast and accurate microRNA search using CNN. BMC Bioinformatics 2019, 20:646
https://doi.org/10.1186/s12859-019-3279-2





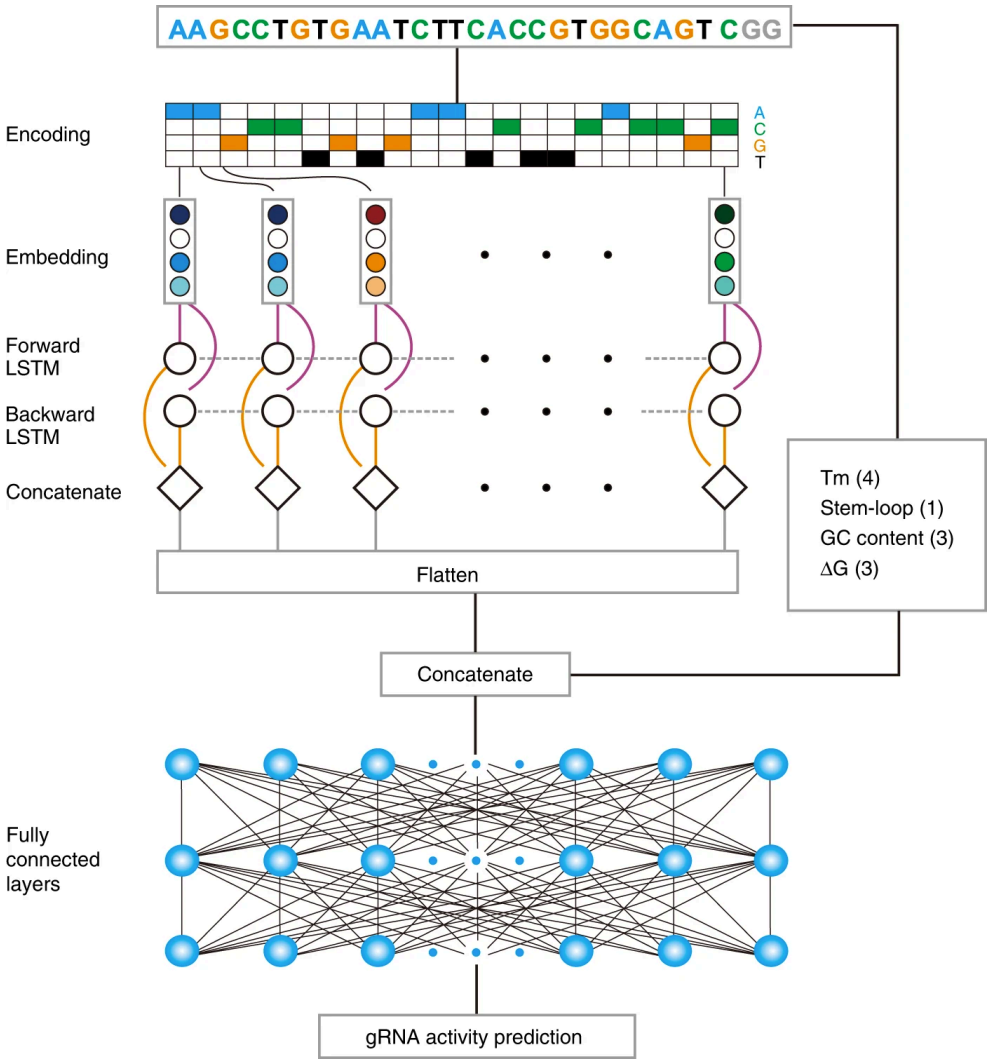
研究事例 / genome sequences

Deep convolutional neural networks for accurate somatic mutation detection. Nat Commun 2019, 10:1041
https://doi.org/10.1038/s41467-019-09027-x



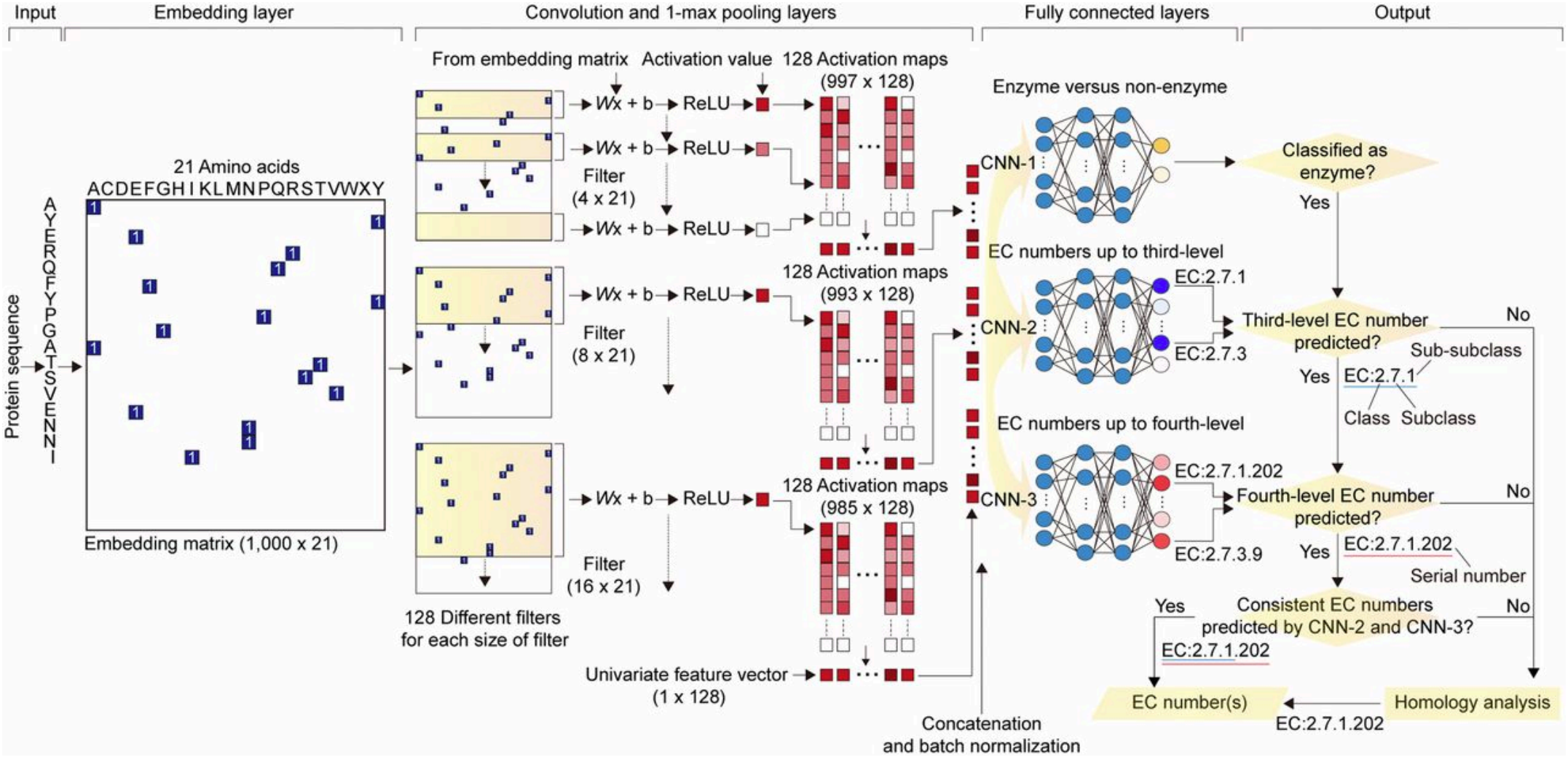
研究事例 / genome sequences

Optimized CRISPR guide RNA design for two high-fidelity Cas9 variants by deep learning. Nat Commun 2019, 10:4284
https://doi.org/10.1038/s41467-019-12281-8



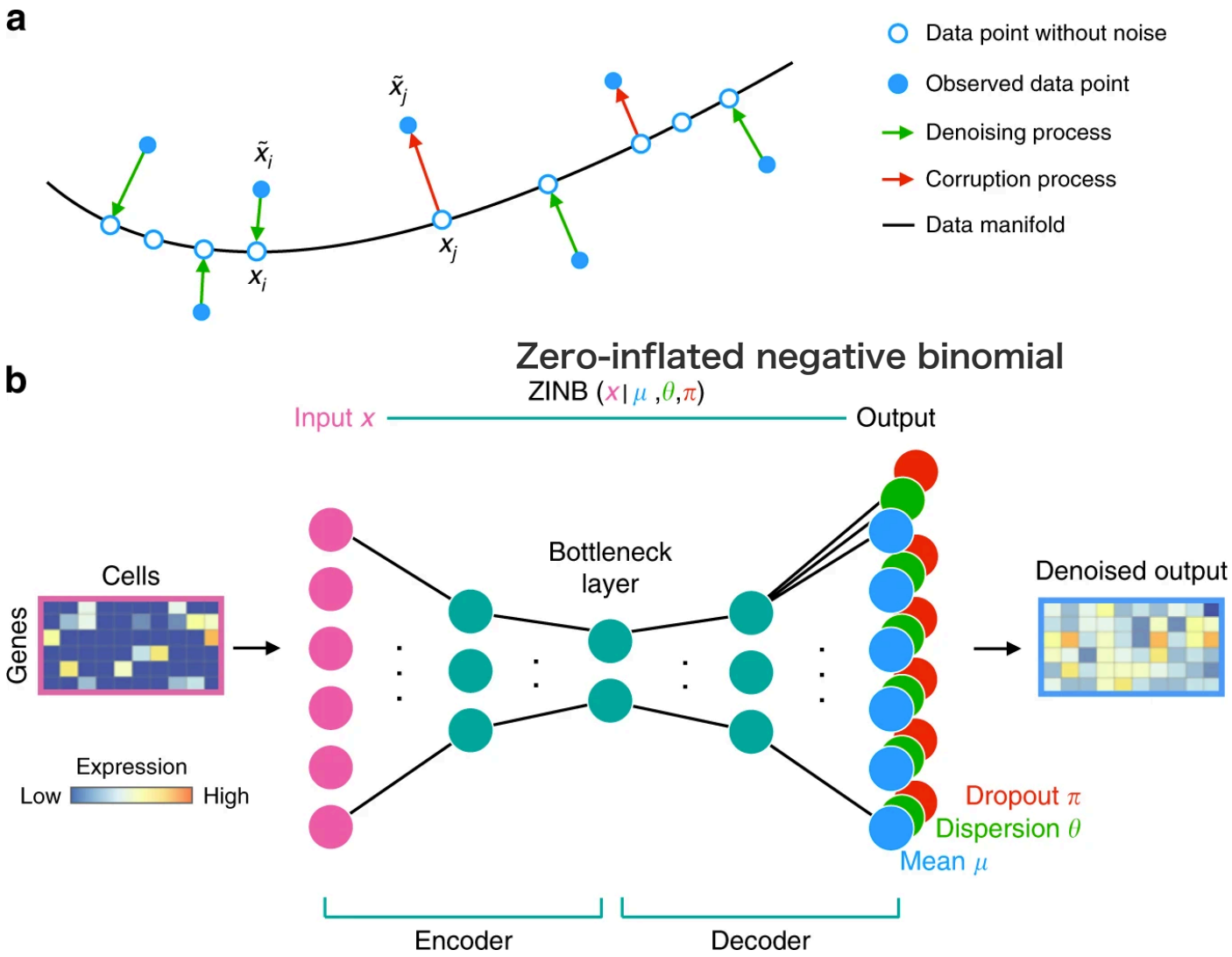
研究事例 / amino acid sequences

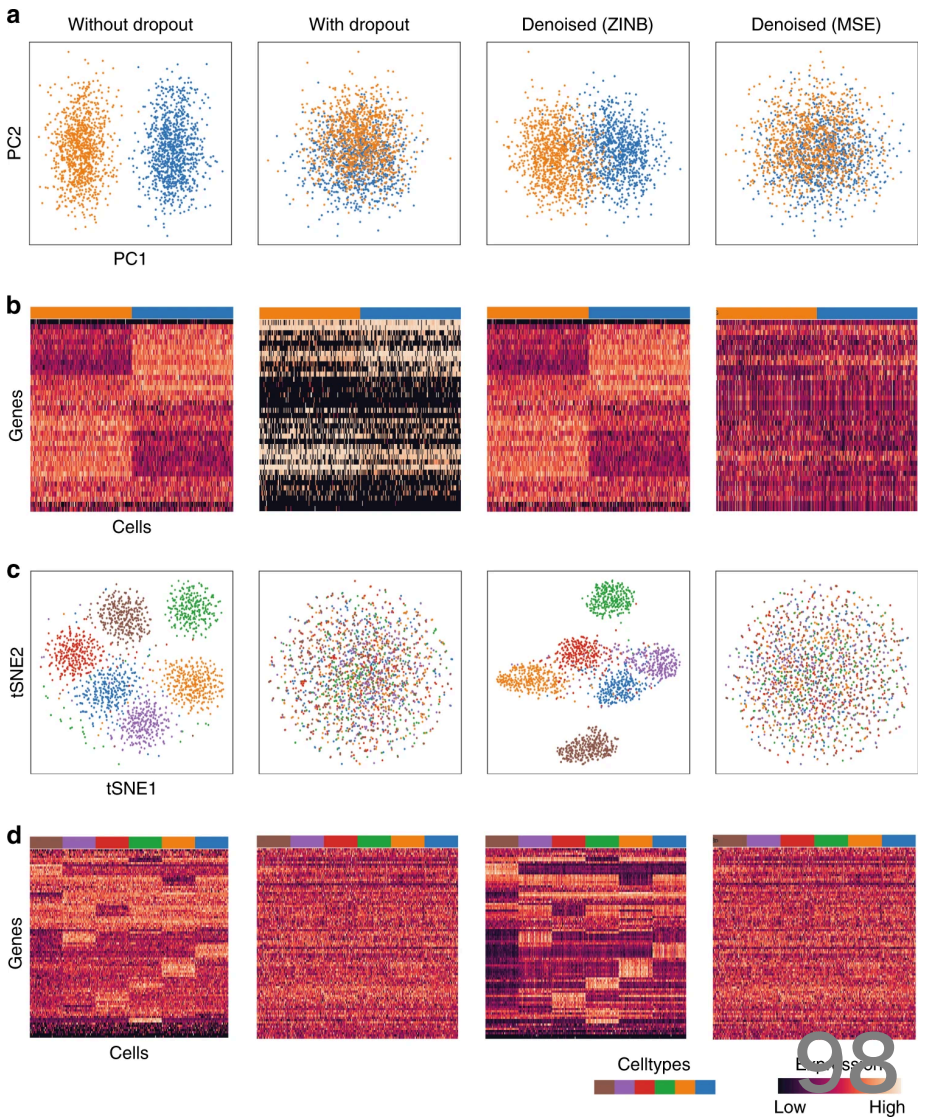
Deep learning enables high-quality and high-throughput prediction of enzyme commission numbers. PNAS 2019, 116(28): 13996-14001
https://doi.org/10.1073/pnas.1821905116



研究事例 / scRNA-Seq

Single-cell RNA-seq denoising using a deep count autoencoder. Nat Commun 2019, 10:390
https://doi.org/10.1038/s41467-018-07931-2





研究事例 / scRNA-Seq

Clustering single-cell RNA-seq data with a model-based deep learning approach. Nat Mach Intell 2019, 1:191-198
https://doi.org/10.1038/s42256-019-0037-0