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、農学生命科学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

OCT
03 13:15–16:30

Python 基礎

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。

OCT
10 13:15–16:30

テキストデータ処理

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。

OCT
17 13:15–16:30

データ分析

第 3 回目の授業では、Python ライブラリー（NumPy や Pandas）を利用して、CSV ファイルの処理などのデータ分析やデータ可視化を中心に取り上げる。

OCT
24 13:15–16:30

スマート農業

Python のライブラリー（PyTorch 等）を利用して、深層学習による物体分類や物体検出モデルを実装する例を示す。

農学生命情報科学特論Ⅰ

3

- Pandas
- データ可視化

python™

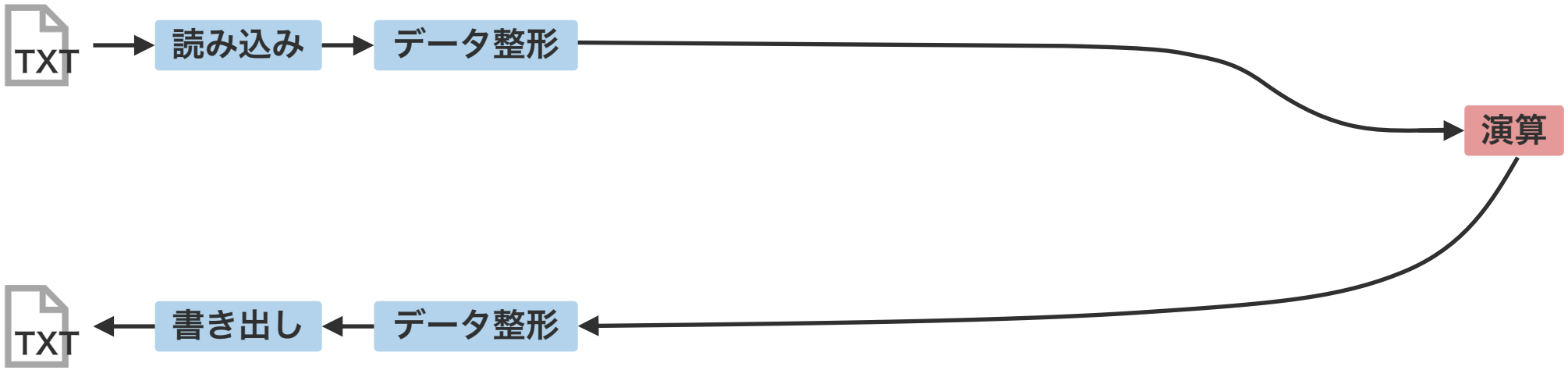
アプリ開発やシステム管理など様々な用途で利用できるような汎用機能を提供する。

- リスト
- 2 次元リスト

NumPy

整形されたデータに対して、情報量をできるだけ落とさずに、高速に演算を行う機能を提供する。

- 配列
- 2 次元配列



python™

pandas

NumPy

アプリ開発やシステム管理など様々な用途で利用できるような汎用機能を提供する。

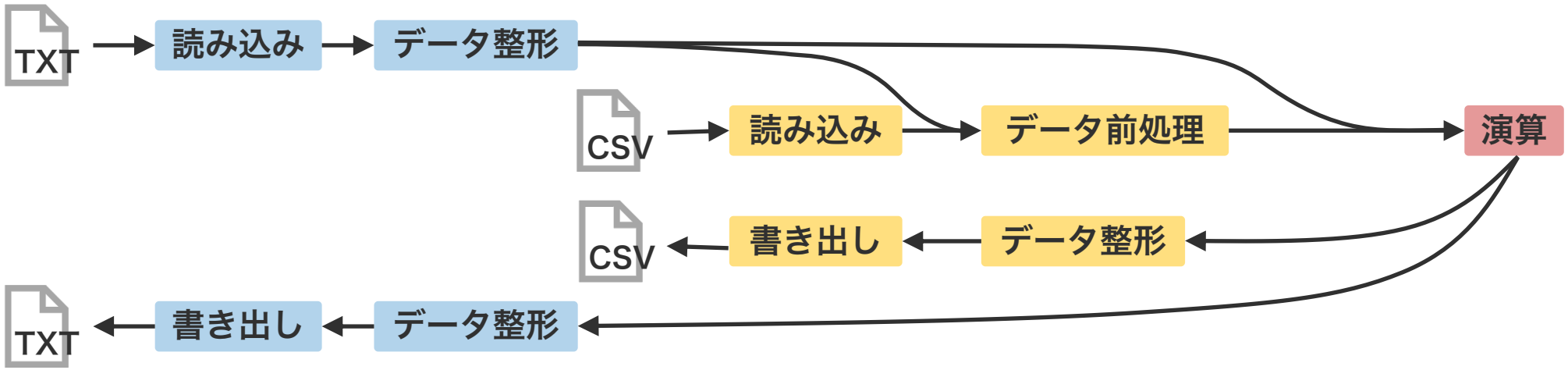
- リスト
- 2 次元リスト

CSV ファイルの読み書きや表データの操作や整形などに特化した機能を提供する。

- シリーズ
- データフレーム

整形されたデータに対して、情報量をできるだけ落とさずに、高速に演算を行う機能を提供する。

- 配列
- 2 次元配列

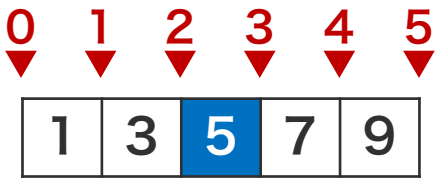


Pandas

- ◯ シリーズ
- ◯ データフレーム
- ◯ 表データ処理

シリーズ

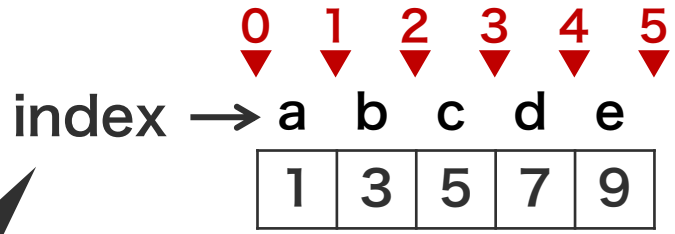
Pandas のシリーズは、1 次元の配列データを扱うときに使用する。pd.Series 関数にリストを代入して作成する。シリーズから要素を取り出すときは、位置番号を指定して取り出す。




```
import pandas as pd

x = pd.Series([1, 3, 5, 7, 9])



x[2]
# 5
```

シリーズ

シリーズは、リストと異なり、各値を位置番号と index の両方で管理している。シリーズを pd.Series 関数で作成するときに、index が自動的に作られるが、自ら指定して作ることもできる。

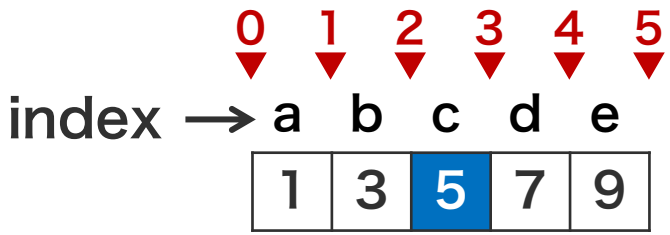


シリーズの各要素に位置番号の他に index と呼ばれる索引が付けられる。

```
import pandas as pd

x = pd.Series([1, 3, 5, 7, 9])
x
# 0    1
# 1    3
# 2    5
# 3    7
# 4    9
# dtype: int64

x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd', 'e'])
x
# a    1
# b    3
# c    5
# d    7
# e    9
# dtype: int64
```

シリーズ

シリーズを作成するとき、文字列を index に指定した場合、その文字列でシリーズの各要素を取得できるようになる。




```
import pandas as pd

x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd', 'e'])
x
# a    1
# b    3
# c    5
# d    7
# e    9
# dtype: int64


x[2]
# 5


x['c']
# 5
```

シリーズ

シリーズは、値と index の両方のデータを保持している。シリーズから値だけを NumPy の 1 次元配列として取得したい場合は、x.values のように取得する。また、シリーズから index だけを取得したい場合は、x.index を使用する。

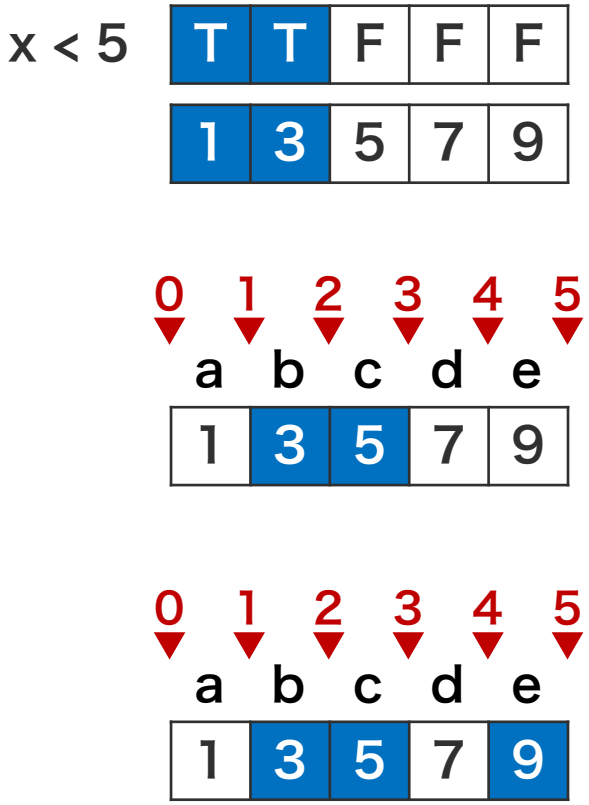
```
import pandas as pd

x = pd.Series([1, 3, 5, 7, 9],
      index=['a', 'b', 'c', 'd', 'e'])

x
# a    1
# b    3
# c    5
# d    7
# e    9
# dtype: int64

x.values
# array([1, 3, 5, 7, 9])

x.index
# Index(['a', 'b', 'c', 'd', 'e'],
#       dtype='object')
```

シリーズ

シリーズから要素を取得するとき、位置番号と index で取得できるほか、NumPy のようにスライスしたり、フィルター（ブーリアンベクトル）を使用して取得したりすることもできる。

```
import pandas as pd
x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd',
'e'])

x[x < 5]




x[1:3]




k = ['b', 'c', 'e']
x[k]
```

シリーズ




```
import pandas as pd
x = pd.Series([1, 3, 5, 7, 9],
              index=['a', 'b', 'c', 'd',
'e'])

x[x < 5]
# a    1
# b    3
# dtype: int64

x[1:3]
# b    3
# c    5
# dtype: int64

k = ['b', 'c', 'e']
x[k]
# b    3
# c    5
# e    9
# dtype: int64
```

問題 P1-1

赤色で書かれているオブジェクトが保持している値を答えよ。

```
import pandas as pd
x = pd.Series([1, 3, 5, 7, 9],
        index=['a', 'b', 'c', 'd', 'e'])


x[['a', 'c', 'e']]




keep1 = (1 < x)
keep2 = (x < 7)
x[keep1 & keep2]
```

```
import pandas as pd
x = pd.Series([1, 3, 5, 7, 9],
        index=['a', 'b', 'c', 'd', 'e'])

x['a'] = 2
x[0] = 4
x.values


x[x > 4] = 6
x.values


x[x % 2 == 0] = 1
x.values
```

シリーズ同士の計算

シリーズ同士は NumPy の配列と同様に四則演算が可能である。計算対象のシリーズの長さが同一でない場合は、見かけ上、短い方に欠損値を埋め込んだ上で計算される。また、シリーズとスカラーの計算も定義されている。この場合、スカラーが自動的に、シリーズと同じ長さに展開されて、計算が行われる（ブロードキャスト）。

```
import pandas as pd

x = pd.Series([1, 3, 5, 9])
y = pd.Series([2, 4, 6, 8])
z = pd.Series([1, 1])
w = 2

a = x + y
a.values
#array([ 3,  7, 11, 17])


b = x - z
b.values
# array([ 0.,  2., nan, nan])

c = x * w
c.values
# array([ 2,  6, 10, 18])
```

シリーズ同士の計算

シリーズに index がついている場合、計算は index に基づいて計算される。どちらか一方のシリーズにしか存在しない index の場合、存在しない方を欠損値として扱う。

```
import pandas as pd


x = pd.Series([1, 3, 5, 7],
              index=['a', 'b', 'c', 'd'])
y = pd.Series([2, 4, 6, 8],
              index=['d', 'c', 'b', 'a'])
a = x + y
a.values
# array([9, 9, 9, 9])


x = pd.Series([1, 3, 5, 7],
              index=['a', 'b', 'c', 'd'])
y = pd.Series([2, 4, 6, 8],
              index=['a', 'b', 'd', 'c'])
a = x * y
a.values
# array([ 2,  12, 40, 42)
```

計算後の結果は index 順に並べ替えられる。x * y と y * x の計算結果は同じである。

シリーズ同士の計算

シリーズに index がついている場合、計算は index に基づいて計算される。どちらか一方のシリーズにしか存在しない index の場合、存在しない方を欠損値として扱う。

```
import pandas as pd


x = pd.Series([1, 3, 5, 7],
        index=['a', 'b', 'd', 'e'])

y = pd.Series([2, 4, 6, 8],
        index=['a', 'b', 'c', 'e'])


a = x + y
a.values
# array([ 3.,  7., nan, nan, 15.])


b = x * y
b.values
# array([ 2., 12., nan, nan, 56.])
```

要約統計量

Pandas のシリーズに対して、平均、分散、中央値などの要約統計量を計算するメソッドが多く用意されている。

```
import numpy as np
import pandas as pd
x = pd.Series([1, 2, 3, 4, 5])

x.count()
# 5

x.min()
# 1

x.max()
# 5

x.idxmax()
# 4

x.quantile(0.25)
# 2.0
```

```
x.sum()
# 15

x.mean()
# 3.0

x.median()
# 3.0

x.var()
# 2.5

x.std()
# 1.5811388300841898

x.cumsum().values
# array([ 1,  3,  6, 10, 15])
```

欠損値 / dropna

Pandas のシリーズに欠損値・非数値（np.nan）を含めることができる。欠損値は、Pandas で用意されたメソッドを使って取り除いたり、その位置を調べたりすることができる。

メソッド	動作
dropna	シリーズ中の np.nan を取り除く。
fillna	シリーズ中の np.nan を指定した値で置き換える。
isnull	シリーズ中の各要素が np.nan かどうかを True と False で表す。
notnull	is.null と反対の動作を行う。

欠損値除去後のシリーズの要素数が変化するので、複数のシリーズを同時に解析するとき、要素数の違いによりブロードキャスト機能が働き、想定外の計算が行われる可能があることに注意。

```
import numpy as np
import pandas as pd

x = pd.Series([1, 3, np.nan, 7,
np.nan])
x
# a     1.0
# b     3.0
# c     NaN
# d     7.0
# e     NaN
# dtype: float64

y = x.dropna()
y
# 0 1.0
# 1 3.0
# 3 7.0
# dtype: float64
```

欠損値 / fillna

Pandas のシリーズに欠損値・非数値（np.nan）を含めることができる。欠損値は、Pandas で用意されたメソッドを使って取り除いたり、その位置を調べたりすることができる。

メソッド	動作
dropna	シリーズ中の np.nan を取り除く。
fillna	シリーズ中の np.nan を指定した値で置き換える。
isnull	シリーズ中の各要素が np.nan かどうかを True と False で表す。
notnull	is.null と反対の動作を行う。

合理的な根拠なしに、すべての欠損値を特定の値に置き換えてはならない。fillna メソッドを使用するときは十分に注意すること。

```
import numpy as np
import pandas as pd

x = pd.Series([1, 3, np.nan, 7,
np.nan])


y = x.fillna(0)
y
# 0 1.0
# 1 3.0
# 2 0.0
# 3 7.0
# 4 0.0
# dtype: float64
```

欠損値 / isnull

Pandas のシリーズに欠損値・非数値（np.nan）を含めることができる。欠損値は、Pandas で用意されたメソッドを使って取り除いたり、その位置を調べたりすることができる。

メソッド	動作
dropna	シリーズ中の np.nan を取り除く。
fillna	シリーズ中の np.nan を指定した値で置き換える。
isnull	シリーズ中の各要素が np.nan かどうかを True と False で表す。
notnull	is.null と反対の動作を行う。

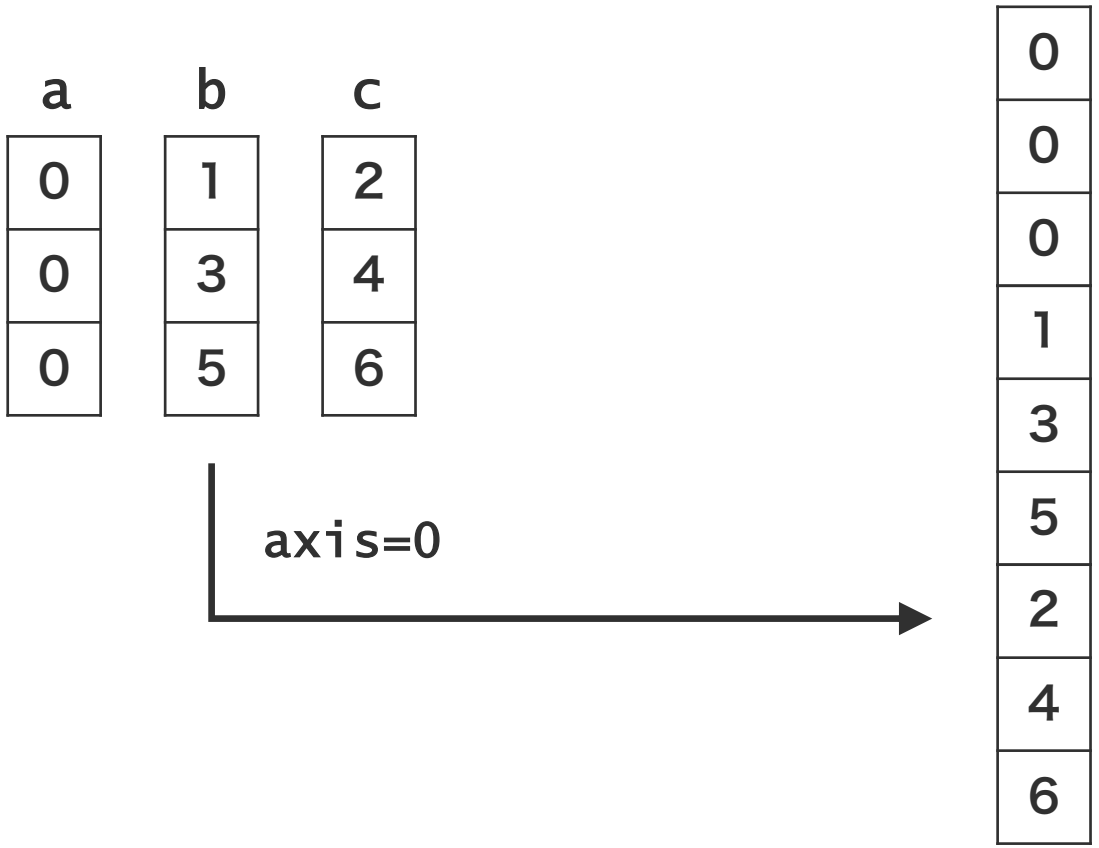
```
import numpy as np
import pandas as pd

x = pd.Series([1, 3, np.nan, 7,
np.nan])


y = x.isnull()
y
# 0 False
# 1 False
# 2 True
# 3 False
# 4 True
# dtype: bool
```

Pandas

- シリーズ
- データフレーム
- 表データ処理

データフレーム / pd.concat

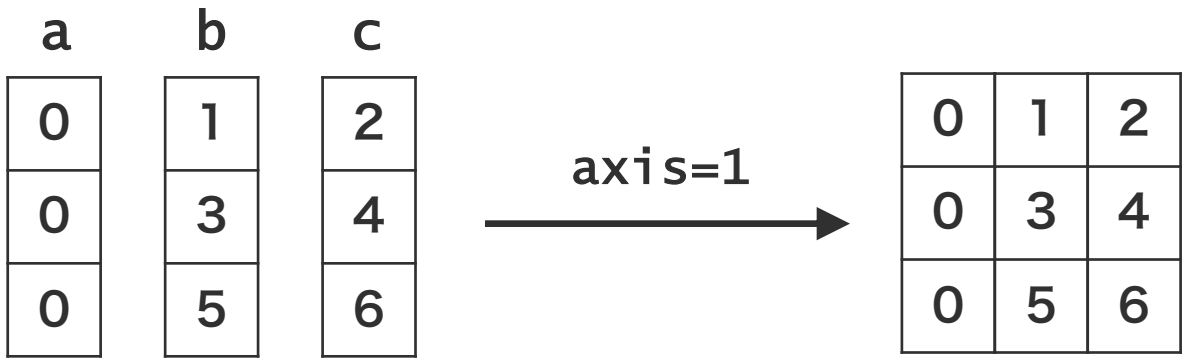
Pandas では、行列型のデータをデータフレームと呼ぶ。データフレームは、シリーズを行方向あるいは列方向に束ねることで作成される。シリーズ同士を束ねるとき pd.concat 関数を使用する。




```
import pandas as pd
a = pd.Series([0, 0, 0])
b = pd.Series([1, 3, 5])
c = pd.Series([2, 4, 6])

x = pd.concat([a, b, c])
x
# 0    0
# 1    0
# 2    0
# 0    1
# 1    3
# 2    5
# 0    2
# 1    4
# 2    6
# dtype: int64

x.__class__.__name__
# 'Series'
```

データフレーム / pd.concat

pd.concat 関数の axis 引数を指定することで、複数のシリーズを列方向に束ねることもできる。




```
import pandas as pd
a = pd.Series([0, 0, 0])
b = pd.Series([1, 3, 5])
c = pd.Series([2, 4, 6])

y = pd.concat([a, b, c], axis=1)
y
#    0  1  2
# 0  0  1  2
# 1  0  3  4
# 2  0  5  6

y.__class__.__name__
# 'DataFrame'
```

データフレーム / ディクショナリ

リストを値として保存しているディクショナリを、Pandas のデータフレームに変換することもできる。このとき、ディクショナリのキーは、データフレームの列名に変換される。

```
import pandas as pd


d = {'buna' : [1, 0, 1, 0, 1],
     'kashi': [1, 3, 5, 7, 9],
     'nara' : [0, 2, 4, 6, 8]}


df = pd.DataFrame(d)
df
#    buna kashi  nara
# 0     1     1     0
# 1     0     3     2
# 2     1     5     4
# 3     0     7     6
# 4     1     9     8
```

データフレーム / ディクショナリ

シリーズを値として保存しているディクショナリも Pandas のデータフレームに変換することもできる。ディクショナリのキーは、データフレームの列名に変換される。また、シリーズの index は、データフレームの index （行名）に変換される。

```
import pandas as pd

w1 = pd.Series([1, 0, 1, 0, 1],
        index=['a', 'b', 'c', 'd',
'e'])
w2 = pd.Series([1, 3, 5, 7, 9],
        index=['a', 'b', 'c', 'd',
'e'])
w3 = pd.Series([0, 2, 4, 6, 8],
        index=['a', 'b', 'c', 'd',
'e'])

d = {'buna': w1, 'kashi': w2,
     'nara': w3}
df = pd.DataFrame(d)
df
#    buna kashi  nara
# a     1     1     0
# b     0     3     2
# c     1     5     4
# d     0     7     6
# e     1     9     8
```

データフレーム / ディクショナリ

index を含むシリーズの場合、データフレームはそれらの index に基づいて作られる。データフレームは index で並べ替えられるため、その順番は必ずしもシリーズの順番を反映しないことに注意。

```
import pandas as pd

w1 = pd.Series([1, 0, 1, 0, 1],
        index=['b', 'a', 'c', 'd',
'f'])
w2 = pd.Series([1, 3, 5, 7, 9],
        index=['a', 'b', 'c', 'f',
'e'])
w3 = pd.Series([0, 2, 4, 6, 8],
        index=['c', 'b', 'a', 'd',
'e'])

d = {'buna': w1, 'kashi': w2,
     'nara': w3}
df = pd.DataFrame(d)
df
#    buna  kashi  nara
# a   0.0    1.0   4.0
# b   1.0    3.0   2.0
# c   1.0    5.0   0.0
# d   0.0    NaN   6.0
```

データフレーム / 二次元リスト

二次元リストを作成し、それをデータフレームに変換することもできる。この際に、データフレームの列名（columns）と行名（index）が自動的に振られる。なお、データフレームを作成するときに、columns と index 引数を利用することで、列名と行名を自由につけることができる。

```
import pandas as pd
d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]])
d
#    0  1  2  3
# 0 11 12 13 14
# 1 21 22 23 24
# 2 31 32 33 34

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])
d
#    C1 C2 C3 C4
# R1 11 12 13 14
# R2 21 22 23 24
# R3 31 32 33 34
```

データフレーム / 行名と列名

データフレームの index と columns は、あとから変更することができる。データフレームの index と columns 属性に直接新しい名前を代入することで変更できる。

```
import pandas as pd

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])
d
#     C1  C2  C3  C4
# R1  11  12  13  14
# R2  21  22  23  24
# R3  31  32  33  34

d.index = ['x', 'y', 'z']
d.columns = ['h', 'i', 'j', 'k']
d
#    h  i  j  k
# x 11 12 13 14
# y 21 22 23 24
# z 31 32 33 34
```

特定の列または行の名前だけを変更したいとき、Pandas の rename メソッドを使用すると便利である。

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.rename.html

データフレーム要素参照 / iloc

データフレームから要素を取得する方法として、位置番号を利用しても、行名・列名を利用しても取得できる。位置番号で取得する場合は、iloc を使用し、0 から始まる位置番号を与える。

	C1	C2	C3	C4
R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

```
import pandas as pd

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])


d.iloc[0, :]
# C1 11
# C2 12
# C3 13
# C4 14

d.iloc[:, 2]
# R1 13
# R2 23
# R3 33
```

データフレーム要素参照 / iloc

	C1	C2	C3	C4
R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

```
import pandas as pd

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])


d.iloc[0:2, 1:4]
#     C2 C3 C4
# R1 12 13 14
# R2 22 23 24



d.iloc[[0, 2], [1, 3]]
#     C2 C4
# R1 12 14
# R3 32 34
```

NumPy の動作と異なるので、両者を混同しないように。

データフレーム要素参照 / loc

データフレームから要素を行名または列名で取得するときは、loc を使用する。

	C1	C2	C3	C4
R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

```
import pandas as pd

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])


d.loc['R1', :]
# C1 11
# C2 12
# C3 13
# C4 14

d.loc[:, 'C3']
# R1 13
# R2 23
# R3 33
```

データフレーム要素参照 / loc

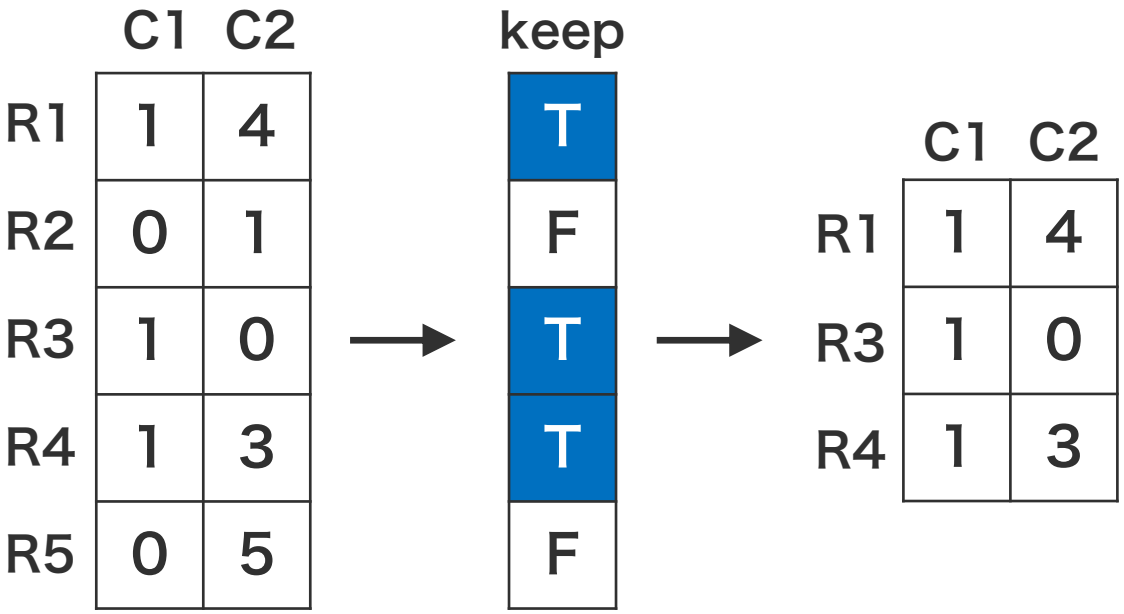
	C1	C2	C3	C4
R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

R1	11	12	13	14
R2	21	22	23	24
R3	31	32	33	34

```
import pandas as pd

d = pd.DataFrame([[11, 12, 13, 14],
                  [21, 22, 23, 24],
                  [31, 32, 33, 34]],
        index=['R1', 'R2', 'R3'],
      columns=['C1', 'C2', 'C3', 'C4'])


d.loc[['R1', 'R2'], 'C2':'C4']
#     C2 C3 C4
# R1 12 13 14
# R2 22 23 24



d.loc[['R1', 'R3'], ['C2', 'C4']]
#     C2 C4
# R1 12 14
# R3 32 34
```

データフレーム要素参照

データフレームもシリーズと同様に、True と False からなるフィルター（ブーリアンベクトル）を使って要素を取得することができる。フィルターを使用する場合は、loc または iloc の両方を使用することができる。




```
import pandas as pd

d = pd.DataFrame([[1, 4],
                  [0, 1],
                  [1, 0],
                  [1, 3],
                  [0, 5]],
   index=['R1', 'R2', 'R3', 'R4',
'R5'],
 columns=['C1', 'C2'])


keep = (d.loc[:, 'C1'] > 0)

d.loc[keep, :]
#     C1 C2
# R1  1  4
# R3  1  0
# R4  1  3
```

データフレーム要素参照

参考

前出の .iloc および .loc 以外にも、.iat および .at を利用した要素参照や列名・行名属性を用いた参照方法などがある。

```
import pandas as pd

df = pd.DataFrame([[1, 4], [0, 1],
                   [1, 0], [1, 3],
                   [0, 5]],

index=['R1','R2','R3','R4','R5'],
  columns=['C1','C2'])

df.iloc[2:4, 0]
df.loc[['R1', 'R3'], ['C1']]
df.iat[0, 1]
df.at['R2', 'C2']
df[0:1]
df[['C1', 'C2']]
df.C2
```

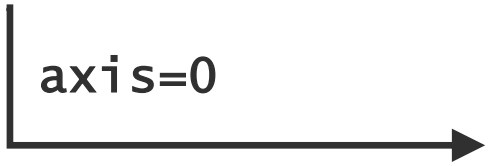
df.iloc	整数からなる位置番号を指定して、該当位置の要素を取得する。複数列や複数行の要素をまとめて取得できる。
df.loc	列名あるいは行名を指定して、該当位置の要素を取得する。複数列や複数行の要素をまとめて取得できる。
df.iat	.iloc と同じ使い方をする。ただし、1 つの要素しか取得できない。
df.at	.loc と同じ使い方をする。ただし、1 つの要素しか取得できない。
df[0:1]	行の位置番号をスライス表記で指定して、該当行の要素を取得する。複数行の要素をまとめて取得できる。
df[['C1']]	列の名前をリストで指定して、該当列の要素を取得する。複数列の要素をまとめて取得できる。ただし、スライス表記は使用できない。
df.C1	列の名前をオブジェクトの属性として、該当列の要素を取得することができる。

データフレーム / pd.concat

pd.concat 関数を使用することで、複数のデータフレームを結合させて 1 つのデータフレームにまとめることができる。pd.concat 関数の axis 引数を通して結合する次元方向を指定できる。

11	12	13	14
21	22	23	24

31	32	33	34
41	42	43	44



11	12	13	14
21	22	23	24
31	32	33	34
41	42	43	44

```
import pandas as pd

d1 = pd.DataFrame([[11, 12, 13, 14],
                   [21, 22, 23, 24]])
d2 = pd.DataFrame([[31, 32, 33, 34],
                   [41, 42, 43, 44]])

df = pd.concat([d1, d2])
df
#    0  1  2  3
# 0 11 12 13 14
# 1 21 22 23 24
# 2 31 32 33 34
# 3 41 42 43 44
```

データフレーム / pd.concat

pd.concat 関数を使用することで、複数のデータフレームを結合させて 1 つのデータフレームにまとめることができる。pd.concat 関数の axis 引数を通して結合する次元方向を指定できる。

11	12	13	14
21	22	23	24

31	32	33	34
41	42	43	44

11	12	13	14	31	32	33	34
21	22	23	24	41	42	43	44

```
import pandas as pd

d1 = pd.DataFrame([[11, 12, 13, 14],
                   [21, 22, 23, 24]])
d2 = pd.DataFrame([[31, 32, 33, 34],
                   [41, 42, 43, 44]])

df = pd.concat([d1, d2], axis=1)
df
#    0  1  2  3  0  1  2  3
# 0 11 12 13 14 31 32 33 34
# 1 21 22 23 24 41 42 43 44
```

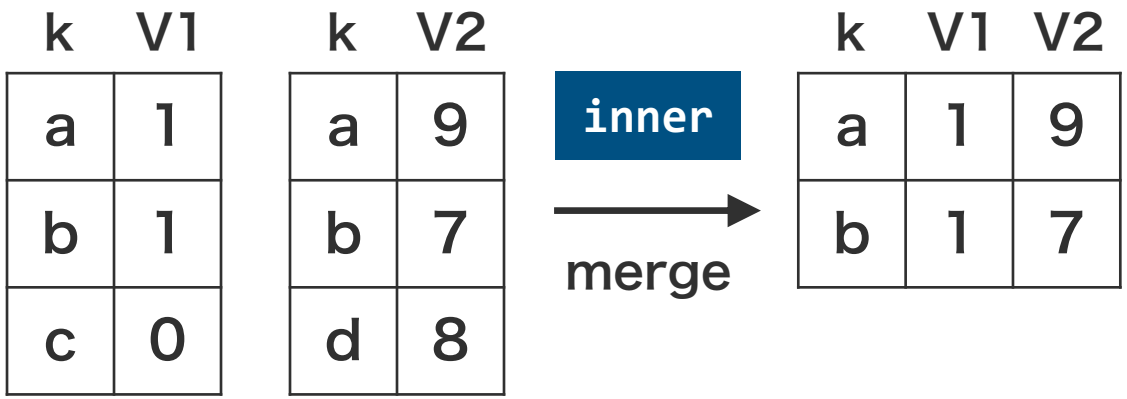
データフレーム / pd.concat

データフレームに index または列名が存在するとき、データフレーム同士が index と列名に基づいて結合される。結合後のデータフレームの行と列の並び順に十分に注意すること。

```
import pandas as pd

d1 = pd.DataFrame([[11, 12, 13, 14],
                   [21, 22, 23, 24]],
        index=['A', 'B'],
        columns=['a', 'b', 'c', 'd'])

d2 = pd.DataFrame([[31, 32, 33, 34],
                   [41, 42, 43, 44]],
        index=['X', 'Y'],
        columns=['a', 'b', 'c', 'e'])


df = pd.concat([d1, d2])
df
#     a  b  c    d    e
# A 11 12 13 14.0  NaN
# B 21 22 23 24.0  NaN
# X 31 32 33  NaN 34.0
# Y 41 42 43  NaN 44.0
```

データフレーム / pd.merge

複数のデータフレームを、特定の列の値に基づいて、マージするすことができる。このとき pd.merge 関数を使用する。



データフレームの結合を行うとき、キーとなる列に重複要素が含まれると、予期しない挙動になる場合があるため、十分に注意すること。

```
import pandas as pd

d1 = pd.DataFrame([['a', 1],
                   ['b', 1],
                   ['c', 0]],
                   columns=['k', 'V1'])

d2 = pd.DataFrame([['a', 9],
                   ['b', 7],
                   ['d', 8]],
                   columns=['k', 'V2'])


d = pd.merge(d1, d2)   # how='inner'
d
#   k V1 V2
# 0 a  1  9
# 1 b  1  7
```

データフレーム / merge

k	V1
a	1
b	1
c	0

k	V2
a	9
b	7
d	8

outer

merge

k	V1	V2
a	1	9
b	1	7
c	0	NaN
d	NaN	8

```
import pandas as pd

d1 = pd.DataFrame([['a', 1],
                   ['b', 1],
                   ['c', 0]],
                   columns=['k', 'V1'])

d2 = pd.DataFrame([['a', 9],
                   ['b', 7],
                   ['d', 8]],
                   columns=['k', 'V2'])


d = pd.merge(d1, d2, how='outer')
d
#    k   V1   V2
# 0  a  1.0  9.0
# 1  b  1.0  7.0
# 2  c  0.0  NaN
# 3  d  NaN  8.0
```

データフレーム / merge

k	V1
a	1
b	1
c	0

k	V2
a	9
b	7
d	8

left

merge

k	V1	V2
a	1	9
b	1	7
c	0	NaN

```
import pandas as pd

d1 = pd.DataFrame([['a', 1],
                   ['b', 1],
                   ['c', 0]],
                   columns=['k', 'V1'])

d2 = pd.DataFrame([['a', 9],
                   ['b', 7],
                   ['d', 8]],
                   columns=['k', 'V2'])


d = pd.merge(d1, d2, how='left')
d
#    k  V1   V2
# 0  a   1  9.0
# 1  b   1  7.0
# 2  c   0  NaN
```

データフレーム / merge

k	V1
a	1
b	1
c	0

k	V2
a	9
b	7
d	8

right

merge

k	V1	V2
a	1	9
b	1	7
d	NaN	8

```
import pandas as pd

d1 = pd.DataFrame([['a', 1],
                   ['b', 1],
                   ['c', 0]],
                   columns=['k', 'V1'])

d2 = pd.DataFrame([['a', 9],
                   ['b', 7],
                   ['d', 8]],
                   columns=['k', 'V2'])


d = pd.merge(d1, d2, how='right')
d
#    k   V1  V2
# 0  a  1.0   9
# 1  b  1.0   7
# 2  d  NaN   8
```

データフレーム / merge

マージ対象のデータフレームの基準列の列名が異なる場合は、pd.merge 関数の left_on および right_on 引数で指定する。

k	V1
a	1
b	1
c	0

f	V2
a	9
b	7
d	8

merge

k	V1	f	V2
a	1	a	9
b	1	b	7
c	0	NaN	NaN
NaN	NaN	d	8

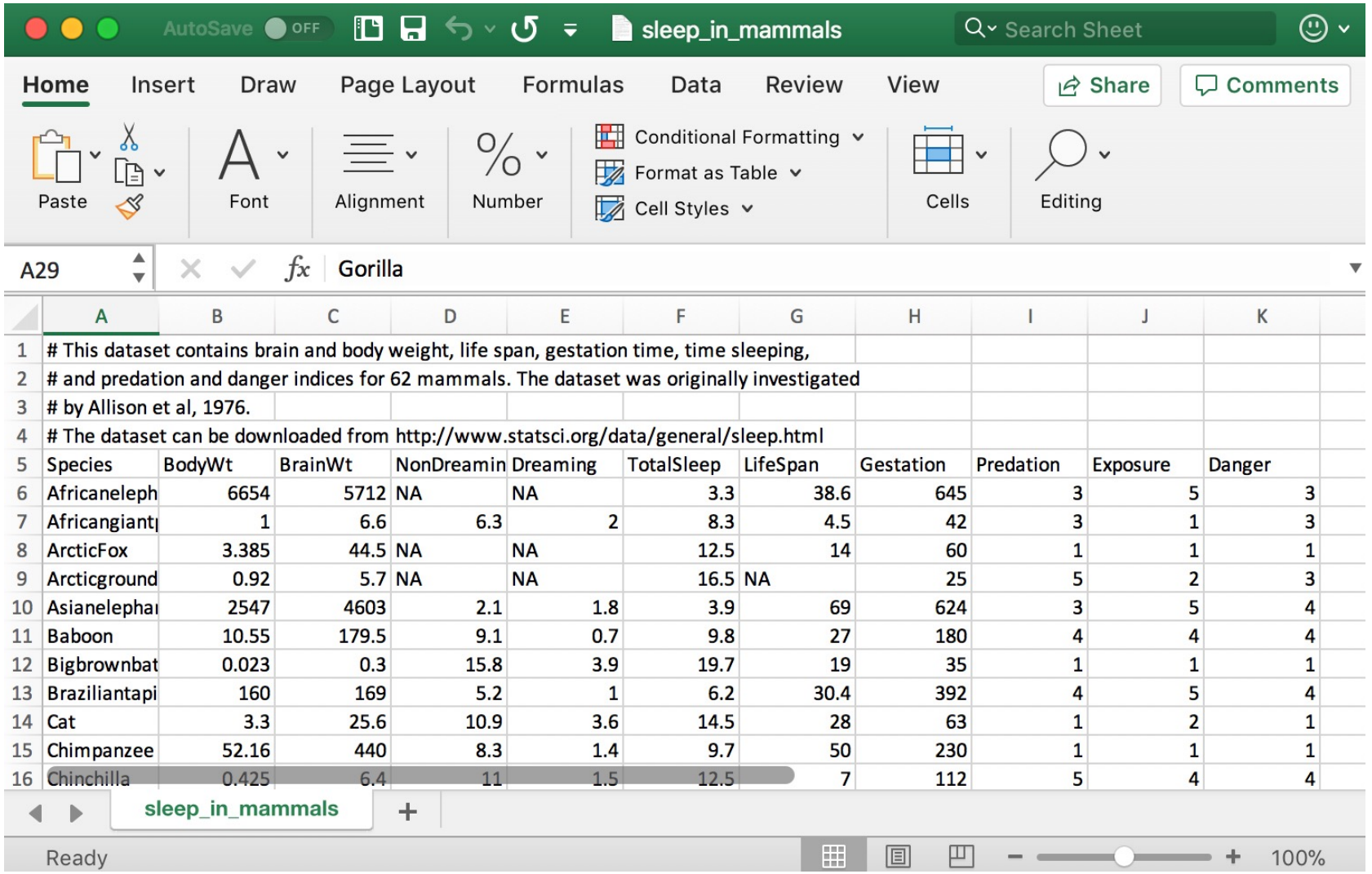
```
import pandas as pd

d1 = pd.DataFrame([['a', 1],
                   ['b', 1],
                   ['c', 0]],
                   columns=['k', 'V1'])

d2 = pd.DataFrame([['a', 9],
                   ['b', 7],
                   ['d', 8]],
                   columns=['f', 'V2'])

d = pd.merge(d1, d2, how='outer',
             left_on='k', right_on='f')
d
#     k   V1    f   V2
# 0   a  1.0    a  9.0
# 1   b  1.0    b  7.0
# 2   c  0.0  NaN  NaN
# 3 NaN  NaN    d  8.0
```

Pandas

- シリーズ
- データフレーム
- **表データ処理**

表データ

生物学で取り扱うデータは、一般的に、1 サンプル 1 行で記載されている。また、各行は複数の項目からなり、サンプルの属性が記載されいてる。このようなデータは、一般的にタブ区切りファイル（TSV）またはカンマ区切りファイル（CSV）のテキストファイルで保存される。



	A	B	C	D	E	F	G	H	I	J	K
1	# This dataset contains brain and body weight, life span, gestation time, time sleeping,										
2	# and predation and danger indices for 62 mammals. The dataset was originally investigated										
3	# by Allison et al, 1976.										
4	# The dataset can be downloaded from http://www.statsci.org/data/general/sleep.html										
5	Species	BodyWt	BrainWt	NonDreamin	Dreaming	TotalSleep	LifeSpan	Gestation	Predation	Exposure	Danger
6	Africaneleph	6654	5712	NA	NA	3.3	38.6	645	3	5	3
7	Africangiant	1	6.6	6.3	2	8.3	4.5	42	3	1	3
8	ArcticFox	3.385	44.5	NA	NA	12.5	14	60	1	1	1
9	Arcticground	0.92	5.7	NA	NA	16.5	NA	25	5	2	3
10	Asianelephar	2547	4603	2.1	1.8	3.9	69	624	3	5	4
11	Baboon	10.55	179.5	9.1	0.7	9.8	27	180	4	4	4
12	Bigbrownbat	0.023	0.3	15.8	3.9	19.7	19	35	1	1	1
13	Braziliantapi	160	169	5.2	1	6.2	30.4	392	4	5	4
14	Cat	3.3	25.6	10.9	3.6	14.5	28	63	1	2	1
15	Chimpanzee	52.16	440	8.3	1.4	9.7	50	230	1	1	1
16	Chinchilla	0.425	6.4	11	1.5	12.5	7	112	5	4	4

表データ

コメント

データ

```
# This dataset contains brain and body weight, life span, gestation time,
# and predation and danger indices for 62 mammals. The dataset was origin
# by Allison et al, 1976.
# The dataset can be downloaded from http://www.statsci.org/data/general/
Species BodyWt  BrainWt NonDreaming     Dreaming         TotalSleep       L
Africanelephant 6654    5712    NA      NA      3.3      38.6    645      3
Africangiantpouchedrat  1       6.6     6.3     2        8.3     4.5      4
ArcticFox       3.385   44.5    NA      NA      12.5     14      60       1
Arcticgroundsquirrel    0.92    5.7     NA      NA       16.5    NA       2
Asianelephant   2547    4603    2.1     1.8     3.9      69      624      3
Baboon  10.55   179.5   9.1     0.7     9.8      27      180     4        4
Bigbrownbat     0.023   0.3     15.8    3.9     19.7     19      35       1
Braziliantapir  160     169     5.2     1       6.2      30.4    392      4
Cat     3.3     25.6    10.9    3.6     14.5     28      63      1        2
Chimpanzee      52.16   440     8.3     1.4     9.7      50      230      1
Chinchilla      0.425   6.4     11      1.5     12.5     7       112      5
Cow     465     423     3.2     0.7     3.9      30      281     5        5
```

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

表データ

属性（特徴量）

ヘッダー →

サンプル

Species	BodyWt	BrainWt	NonDreaming	Dreaming		TotalSleep		L
Africanelephant	6654	5712	NA	NA	3.3	38.6	645	3
Africangiantpouchedrat	1	欠損値	6.6	6.3	2	8.3	4.5	4
ArcticFox	3.385	44.5	NA	NA	12.5	14	60	1
Arcticgroundsquirrel		0.92	5.7	NA	NA	16.5	NA	2
Asianelephant	2547	4603	2.1	1.8	3.9	69	624	3
Baboon	10.55	179.5	9.1	0.7	9.8	27	180	4
Bigbrownbat		0.023	0.3	15.8	3.9	19.7	19	1
Braziliantapir		160	169	5.2	1	6.2	30.4	392
Cat	3.3	25.6	10.9	3.6	14.5	28	63	1
Chimpanzee		52.16	440	8.3	1.4	9.7	50	230
Chinchilla		0.425	6.4	11	1.5	12.5	7	112
Cow	465	423	3.2	0.7	3.9	30	281	5

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

表データ読み込み

Pandas の read_table 関数を使うことで、CSV または TSV データを読み込むことができる。

```
import pandas as pd

f = 'sleep_in_mammals.txt'

d = pd.read_csv(f)
```

```
 # This dataset contains brain and body weight   life span   gestation time   time sleeping  Unnamed: 4
0   # and predation and danger indices for 62 mamm...         NaN              NaN             NaN         NaN
1                                 # by Allison et al      1976.0              NaN             NaN         NaN
2   # The dataset can be downloaded from http://ww...         NaN              NaN             NaN         NaN
3   Species\tBodyWt\tBrainWt\tNonDreaming\tDreamin...         NaN              NaN             NaN         NaN
4   Africanelephant\t6654\t5712\tNA\tNA\t3.3\t38.6...         NaN              NaN             NaN         NaN
..                                                ...         ...              ...             ...         ...
61  Treehyrax\t2\t12.3\t4.9\t0.5\t5.4\t7.5\t200\t3...         NaN              NaN             NaN         NaN
62  Treeshrew\t0.104\t2.5\t13.2\t2.6\t15.8\t2.3\t4...         NaN              NaN             NaN         NaN
63  Vervet\t4.19\t58\t9.7\t0.6\t10.3\t24\t210\t4\t...         NaN              NaN             NaN         NaN
64  Wateropossum\t3.5\t3.9\t12.8\t6.6\t19.4\t3\t14...         NaN              NaN             NaN         NaN
65  Yellow-belliedmarmot\t4.05\t17\tNA\tNA\tNA\t13...         NaN              NaN             NaN         NaN
```

read_csv 関数のオプションを正しく指定しないと、古いバージョンの Pandas ではエラーが起き、新しいバージョンの Pandas ではエラーは起きないがデータを正しく認識できない。

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

表データ読み込み

Pandas の read_table 関数を使うことで、CSV または TSV データを読み込むことができる。データを正しく読み込むには、read_table 関数に、コメント行を明示し、ヘッダー行の有無、区切り文字の種類を正しく指定する必要がある。

```
import pandas as pd

f = 'sleep_in_mammals.txt'

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

バックスラッシュは特別な意味を持つ文字である。バックスラッシュの後に続く1文字は、特別な意味を持つ。例えば 't' は英文字の t を表すが、'\t' はタブを表す。

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

表データ読み込み

```
import pandas as pd

f = 'sleep_in_mammals.txt'

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

d.shape
# (62, 11)

d.head()
#                  Species    BodyWt  BrainWt  ...  Predation  Exposure  Danger
# 0         Africanelephant  6654.000   5712.0  ...          3         5       3
# 1  Africangiantpouchedrat     1.000      6.6  ...          3         1       3
# 2              ArcticFox     3.385     44.5  ...          1         1       1
# 3    Arcticgroundsquirrel     0.920      5.7  ...          5         2       3
# 4           Asianelephant  2547.000   4603.0  ...          3         5       4
# [5 rows x 11 columns]
```

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

表データ読み込み

```
import pandas as pd

f = 'sleep_in_mammals.txt'

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

d.shape
# (62, 10)

d.head()
#                          BodyWt  BrainWt  ...  Exposure  Danger
# Species                                   ...
# Africanelephant        6654.000   5712.0  ...         5       3
# Africangiantpouchedrat    1.000      6.6  ...         1       3
# ArcticFox                 3.385     44.5  ...         1       1
# Arcticgroundsquirrel      0.920      5.7  ...         2       3
# Asianelephant          2547.000   4603.0  ...         5       4
# [5 rows x 10 columns]
```

特定の列をデータの一部ではなくて、行名として取り組むこともできる。

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

表データ / 行名と列名

```
import pandas as pd

f = 'sleep_in_mammals.txt'

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

d.index
# Index(['Africanelephant', 'Africangiantpouchedrat', 'ArcticFox',
#        'Arcticgroundsquirrel', 'Asianelephant', 'Baboon', 'Bigbrownbat',
#        'Braziliantapir', 'Cat', 'Chimpanzee', 'Chinchilla', 'Cow',
#        ...
#        'Treeshrew', 'Vervet', 'Wateropossum', 'Yellow-belliedmarmot'],
#       dtype='object', name='Species')

d.columns
# Index(['BodyWt', 'BrainWt', 'NonDreaming', 'Dreaming', 'TotalSleep',
#        'LifeSpan', 'Gestation', 'Predation', 'Exposure', 'Danger'],
#       dtype='object')
```

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

表データ / 要素の取得

```
import pandas as pd

f = 'sleep_in_mammals.txt'

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

d.iloc[0:3, 0:5]
#                          BodyWt  BrainWt  NonDreaming  Dreaming  TotalSleep
# Species
# Africanelephant         6654.000   5712.0          NaN       NaN         3.3
# Africangiantpouchedrat     1.000      6.6          6.3       2.0         8.3
# ArcticFox                  3.385     44.5          NaN       NaN        12.5

d.iloc[0:2, :]
#                          BodyWt  BrainWt  NonDreaming  ... Exposure  Danger
# Species                                                ...
# Africanelephant         6654.000   5712.0          NaN  ...        5       3
# Africangiantpouchedrat     1.000      6.6          6.3  ...        1       3
```

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

表データ / 要素の取得

```
import pandas as pd

f = 'sleep_in_mammals.txt'

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

species = ['Cat', 'Rat', 'Cow', 'Pig']
features = ['BodyWt', 'BrainWt', 'TotalSleep', 'LifeSpan']

d.loc[species, features]
#          BodyWt  BrainWt  TotalSleep  LifeSpan
# Species
# Cat        3.30     25.6        14.5      28.0
# Rat        0.28      1.9        13.2       4.7
# Cow      465.00    423.0         3.9      30.0
# Pig      192.00    180.0         8.4      27.0
```

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

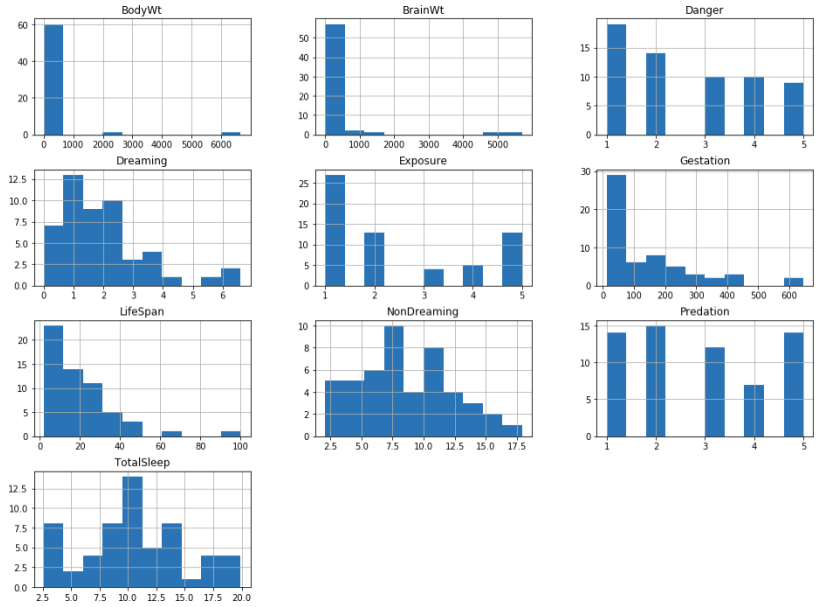
表データ読み込み / データ要約

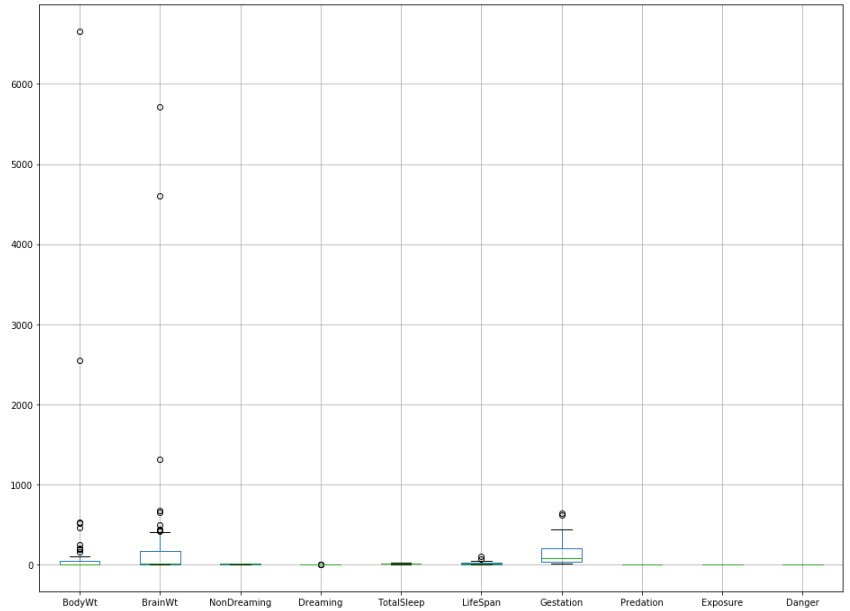
```
import pandas as pd

f = 'sleep_in_mammals.txt'

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

d.describe()
#              BodyWt      BrainWt  NonDreaming  ...  Predation   Exposure     Danger
# count    62.000000    62.000000    48.000000  ...  62.000000  62.000000  62.000000
# mean    198.789984   283.134194     8.672917  ...   2.870968   2.419355   2.612903
# std     899.158011   930.278942     3.666452  ...   1.476414   1.604792   1.441252
# min       0.005000     0.140000     2.100000  ...   1.000000   1.000000   1.000000
# 25%       0.600000     4.250000     6.250000  ...   2.000000   1.000000   1.000000
# 50%       3.342500    17.250000     8.350000  ...   3.000000   2.000000   2.000000
# 75%      48.202500   166.000000    11.000000  ...   4.000000   4.000000   4.000000
# max    6654.000000  5712.000000    17.900000  ...   5.000000   5.000000   5.000000
# [8 rows x 10 columns]
```

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

表データ読み込み / データ可視化

```
import pandas as pd

f = 'sleep_in_mammals.txt'

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

d.hist()
d.boxplot()
```





※ 一部の属性の値の範囲が大きいので、対数化してからグラフに描くと全体の傾向を掴めやすくなる。

※ matplotlib でグラフを描いた方が一般的であるので、ここでは Pandas の視覚化機能を取り上げない。

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

問題 P2-1

diversity_galapagos.txt を Pandas で読み込み、面積（Area）が最も大きい島の名前とその面積を答えよ。

```
import pandas as pd

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

ヒント

シリーズ s の最大値は max(s) で求めることができる。

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

問題 P2-2

diversity_galapagos.txt を Pandas で読み込み、 各島における面積あたりの種数を求めよ。

```
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

ファイル書き出し

Pandas では、データを CSV または TSV ファイルに書き出すとき、to_csv 関数（メソッド）を使う。この際に、区切り文字、行名の有無、列名の有無を指定することができる。また、欠損値を特定の文字に変換することもできる。

```
import pandas as pd
f = 'sleep_in_mammals.txt'

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

d.to_csv('o.txt', sep=',', header=False, index=False)
```

```
6654.0,5712.0,,,3.3,38.6,645.0,3,5,3
1.0,6.6,6.3,2.0,8.3,4.5,42.0,3,1,3
3.385,44.5,,,12.5,14.0,60.0,1,1,1
0.92,5.7,,,16.5,,25.0,5,2,3
2547.0,4603.0,2.1,1.8,3.9,69.0,624.0,3,5,4
10.55,179.5,9.1,0.7,9.8,27.0,180.0,4,4,4
0.023,0.3,15.8,3.9,19.7,19.0,35.0,1,1,1
160.0,169.0,5.2,1.0,6.2,30.4,392.0,4,5,4
3.3,25.6,10.9,3.6,14.5,28.0,63.0,1,2,1
52.16,440.0,8.3,1.4,9.7,50.0,230.0,1,1,1
```

ファイル書き出し

Pandas では、データを CSV または TSV ファイルに書き出すとき、to_csv 関数（メソッド）を使う。この際に、区切り文字、行名の有無、列名の有無を指定することができる。また、欠損値を特定の文字に変換することもできる。

```
import pandas as pd
f = 'sleep_in_mammals.txt'

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

d.to_csv('o.txt', sep=',', header=True, index=True)
```

```
Species,BodyWt,BrainWt,NonDreaming,Dreaming,TotalSleep,LifeSpan,Gestation,Predation,Exposure
Africanelephant,6654.0,5712.0,,,3.3,38.6,645.0,3,5,3
Africangiantpouchedrat,1.0,6.6,6.3,2.0,8.3,4.5,42.0,3,1,3
ArcticFox,3.385,44.5,,,12.5,14.0,60.0,1,1,1
Arcticgroundsquirrel,0.92,5.7,,,16.5,,25.0,5,2,3
Asianelephant,2547.0,4603.0,2.1,1.8,3.9,69.0,624.0,3,5,4
Baboon,10.55,179.5,9.1,0.7,9.8,27.0,180.0,4,4,4
Bigbrownbat,0.023,0.3,15.8,3.9,19.7,19.0,35.0,1,1,1
Braziliantapir,160.0,169.0,5.2,1.0,6.2,30.4,392.0,4,5,4
Cat,3.3,25.6,10.9,3.6,14.5,28.0,63.0,1,2,1
```

ファイル書き出し

Pandas では、データを CSV または TSV ファイルに書き出すとき、to_csv 関数（メソッド）を使う。この際に、区切り文字、行名の有無、列名の有無を指定することができる。また、欠損値を特定の文字に変換することもできる。

```
import pandas as pd
f = 'sleep_in_mammals.txt'

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

d.to_csv('o.txt', sep='\t', header=True, index=True)
```

```
Species BodyWt  BrainWt NonDreaming     Dreaming        TotalSleep      LifeSpan        Gestati
Africanelephant 6654.0  5712.0                  3.3     38.6    645.0   3       5       3
Africangiantpouchedrat  1.0     6.6     6.3     2.0     8.3     4.5     42.0    3       1
ArcticFox       3.385   44.5                    12.5    14.0    60.0    1       1       1
Arcticgroundsquirrel    0.92    5.7                     16.5            25.0    5       2
Asianelephant   2547.0  4603.0  2.1     1.8     3.9     69.0    624.0   3       5       4
Baboon  10.55   179.5   9.1     0.7     9.8     27.0    180.0   4       4       4
Bigbrownbat     0.023   0.3     15.8    3.9     19.7    19.0    35.0    1       1       1
Braziliantapir  160.0   169.0   5.2     1.0     6.2     30.4    392.0   4       5       4
Cat     3.3     25.6    10.9    3.6     14.5    28.0    63.0    1       2       1
```

ファイル書き出し

Pandas では、データを CSV または TSV ファイルに書き出すとき、to_csv 関数（メソッド）を使う。この際に、区切り文字、行名の有無、列名の有無を指定することができる。また、欠損値を特定の文字に変換することもできる。

```
import pandas as pd
f = 'sleep_in_mammals.txt'

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

d.to_csv('o.txt', sep='\t', header=True, index=True)
         na_rep= 'NA')
```

```
Species BodyWt  BrainWt NonDreaming     Dreaming        TotalSleep      LifeSpan        Gestati
Africanelephant 6654.0  5712.0  NA      NA      3.3     38.6    645.0   3       5       3
Africangiantpouchedrat  1.0     6.6     6.3     2.0     8.3     4.5     42.0    3       1
ArcticFox       3.385   44.5    NA      NA      12.5    14.0    60.0    1       1       1
Arcticgroundsquirrel    0.92    5.7     NA      NA      16.5    NA      25.0    5       2
Asianelephant   2547.0  4603.0  2.1     1.8     3.9     69.0    624.0   3       5       4
Baboon  10.55   179.5   9.1     0.7     9.8     27.0    180.0   4       4       4
Bigbrownbat     0.023   0.3     15.8    3.9     19.7    19.0    35.0    1       1       1
Braziliantapir  160.0   169.0   5.2     1.0     6.2     30.4    392.0   4       5       4
Cat     3.3     25.6    10.9    3.6     14.5    28.0    63.0    1       2       1
```

空白がNAに置換される

農学生命情報科学特論Ⅰ

3

- Pandas
- データ可視化

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。

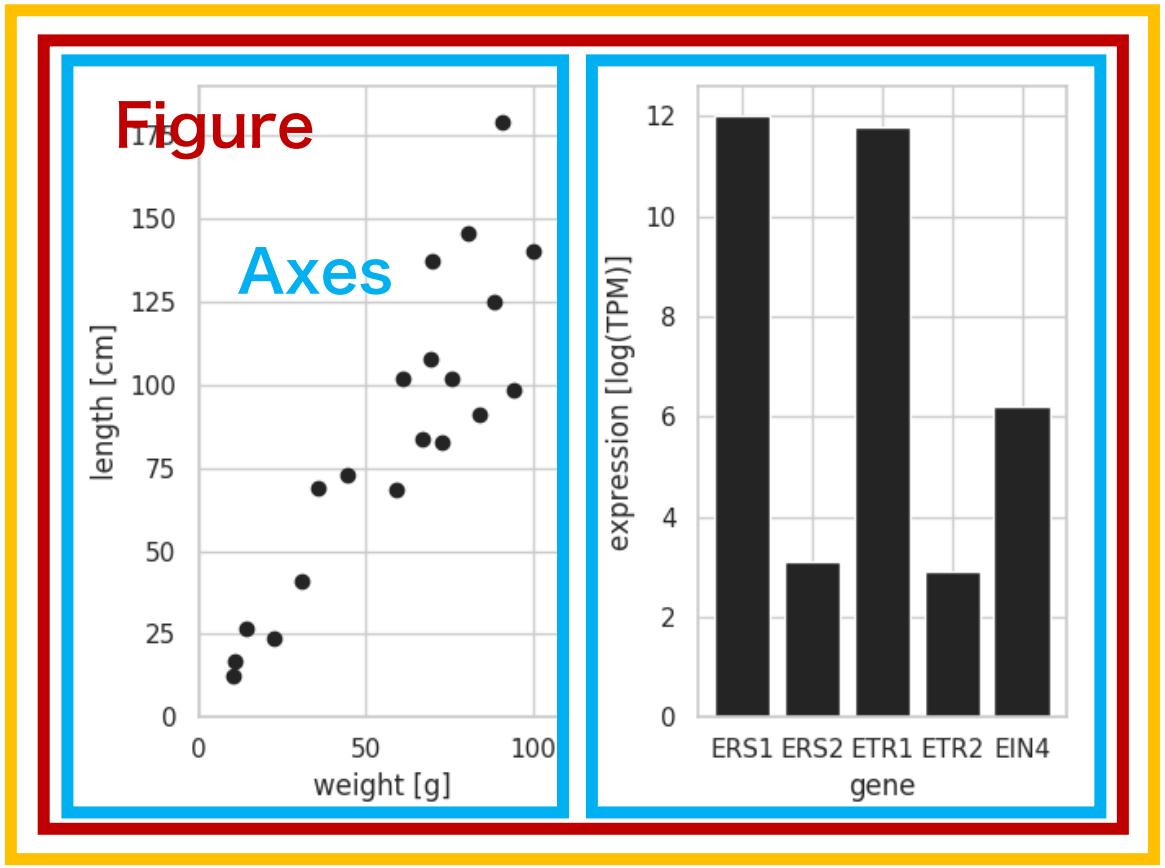
object-oriented interface

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()
```

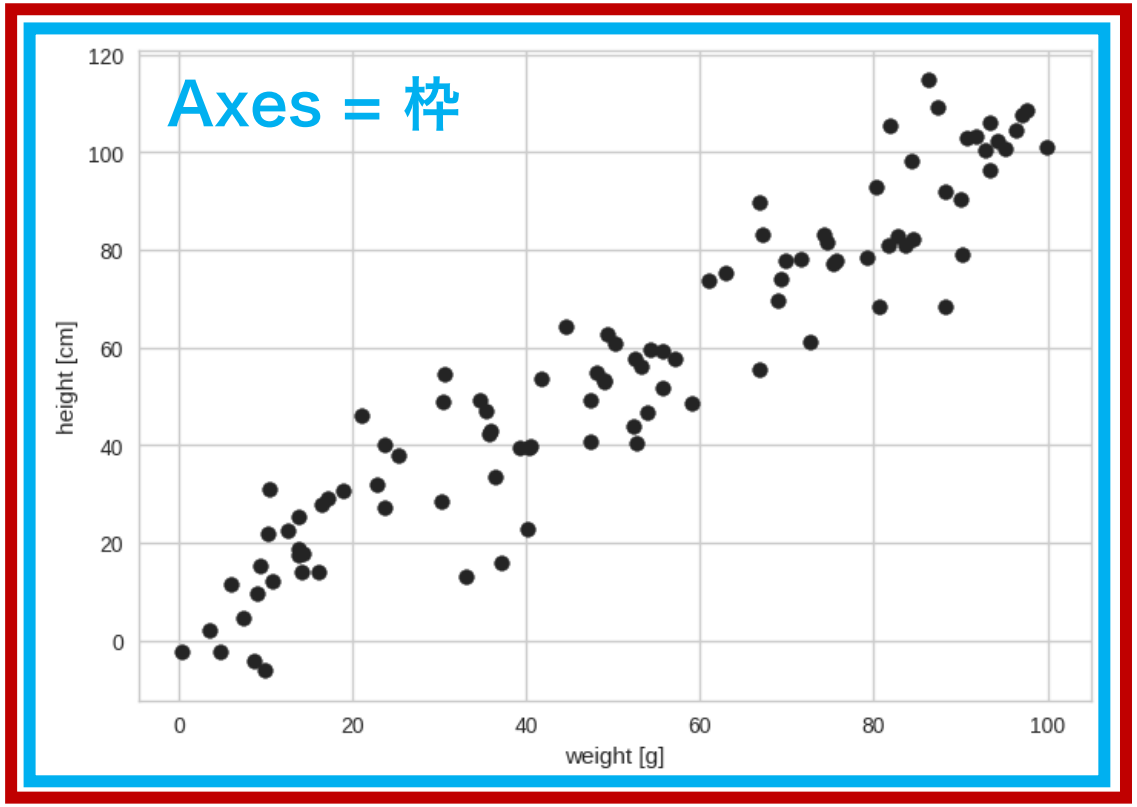
state-based interface

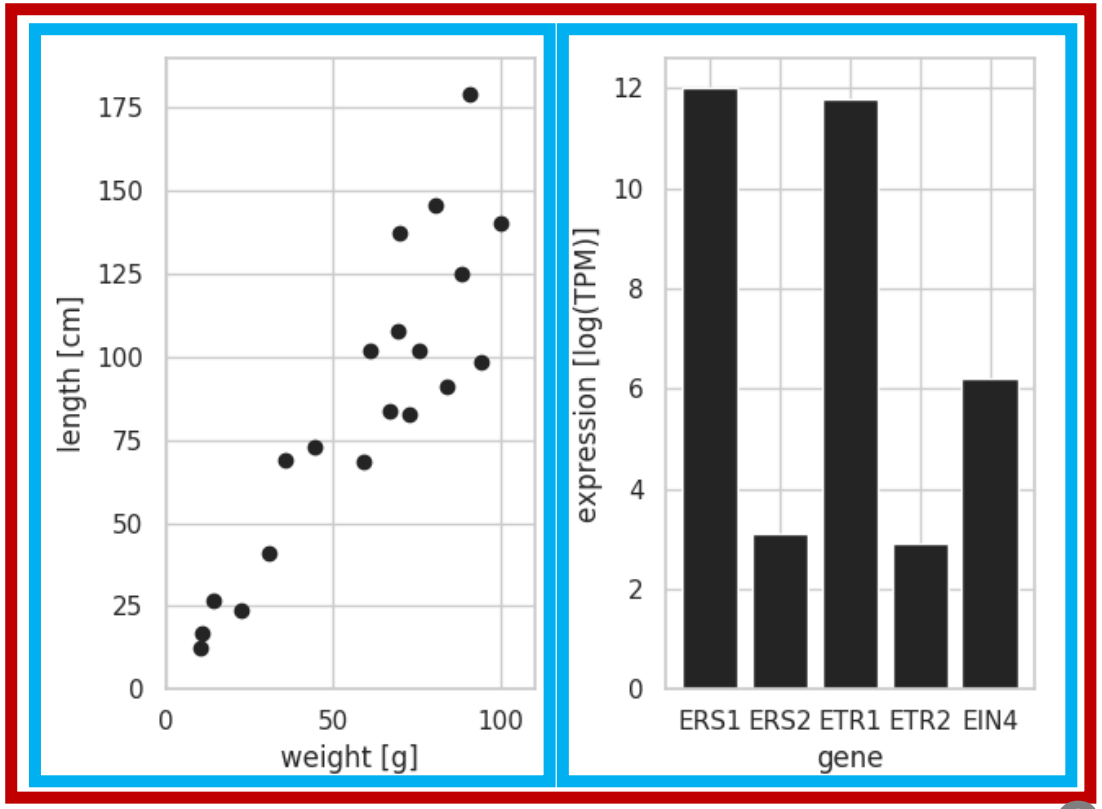
```
import numpy as np
import matplotlib.pyplot as plt

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

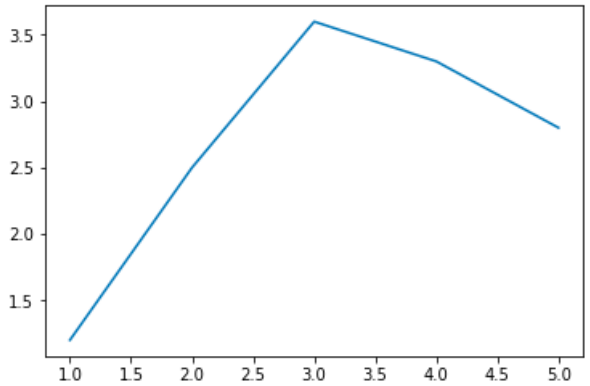




object-oriented interface

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。

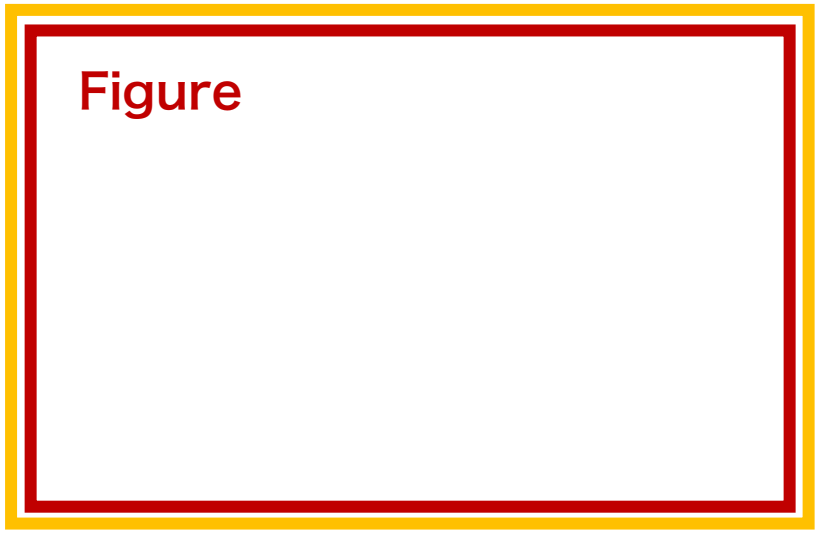
▶

```
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

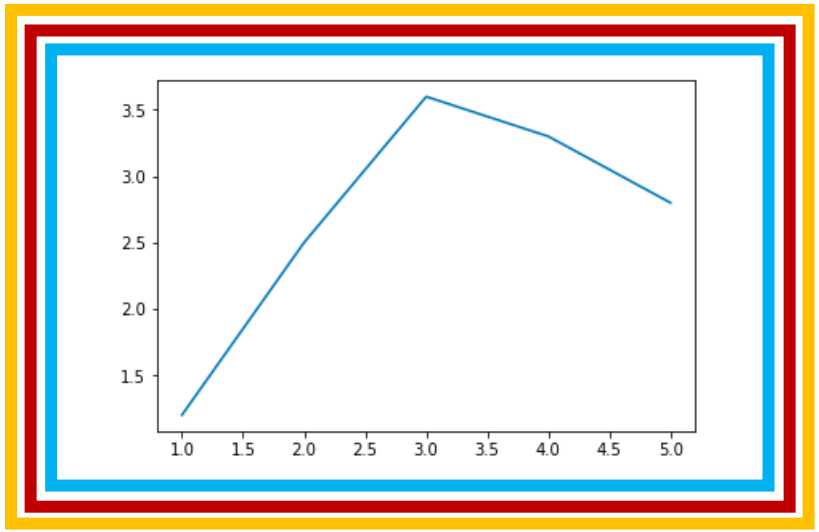
1. matplotlib.pyplot モジュールの機能を呼び出す。pyplot 描画デバイスが用意される。
2. Figure クラスのオブジェクト（領域）を用意する。
3. Figure 領域の中に、さらに Axes クラスのオブジェクト（領域）を作成する。
4. Axes 領域に線グラフを描く。

```
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。

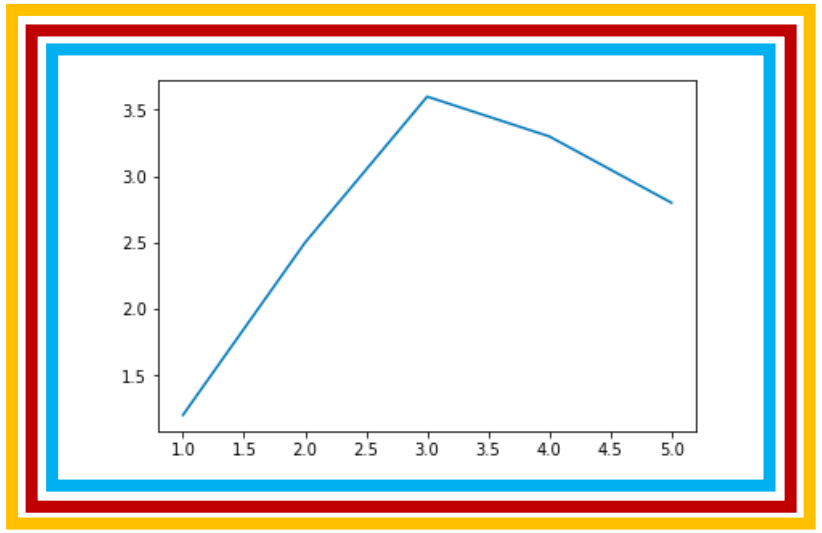
```
import numpy as np
import matplotlib.pyplot as plt

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。

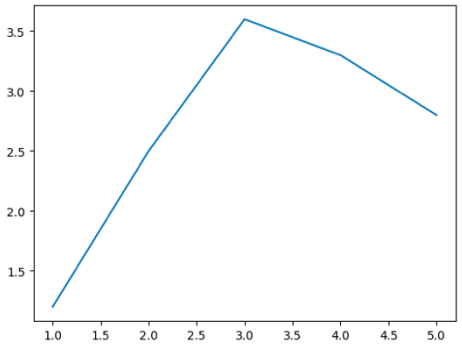
グラフ	object-oriented	state-based
線グラフ	`ax.plot`	`plt.plot`
散布図	`ax.scatter`	`plt.scatter`
棒グラフ	`ax.bar`	`plt.bar`
ヒストグラム	`ax.hist`	`plt.hist`
ボックスプロット	`ax.boxplot`	`plt.boxplot`

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。

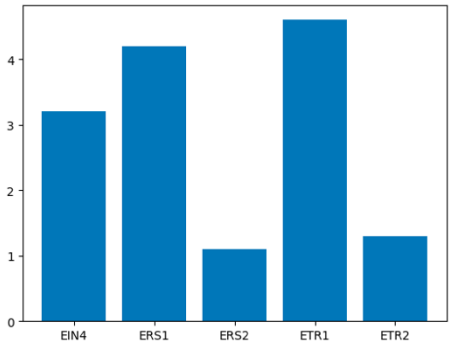
グラフ	object-oriented	state-based
グラフのタイトル	`ax.set_title`	`plt.title`
目盛りの比率	`ax.set_aspect`	`plt.axes().set_aspect`
グラフの凡例	`ax.legend`	`plt.legend`
x 軸の表示範囲	`ax.set_xlim`	`plt.xlim`
x 軸のラベル	`ax.set_xlabel`	`plt.xlabel`
x 軸の目盛りの表示位置	`ax.set_xticks`	`plt.xticks`
x 軸の目盛りの値	`ax.set_xticklabels`	
x 軸の目盛りの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

基本グラフ



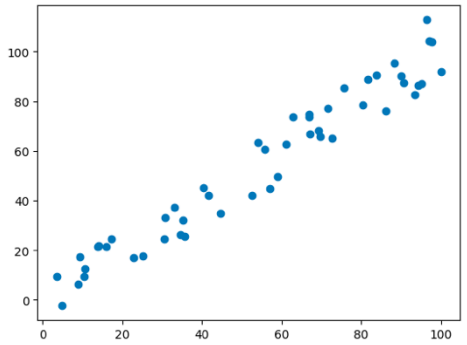
line chart

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。



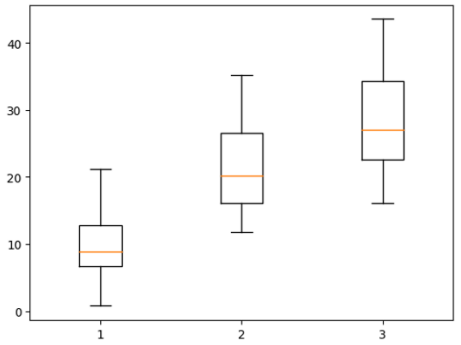
bar chart

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。



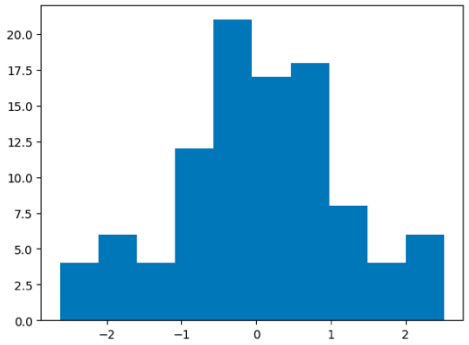
scatter chart

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

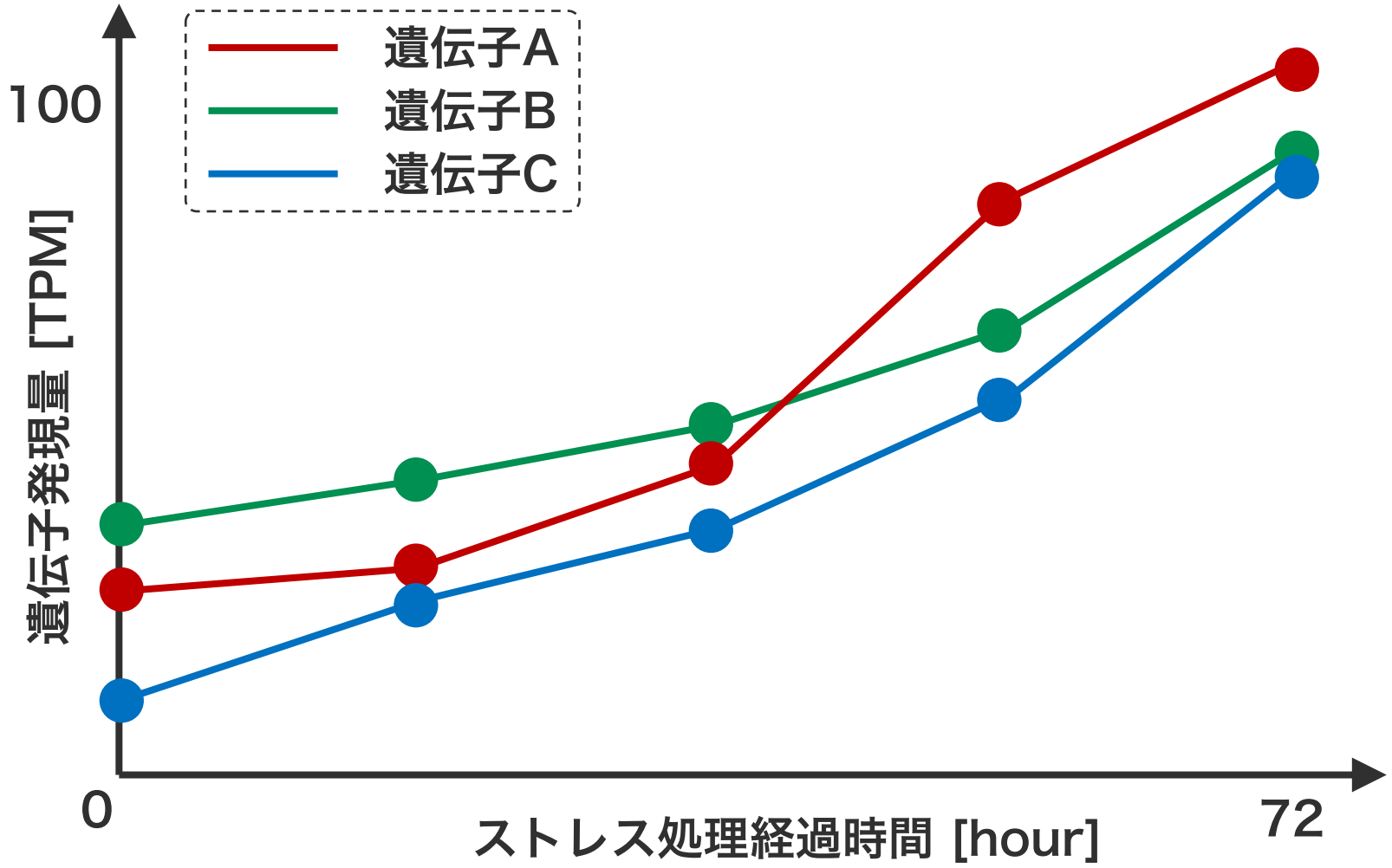
ボックスプロットは、複数の連続量データの分布の特徴を可視化する目的で使われる。



histogram

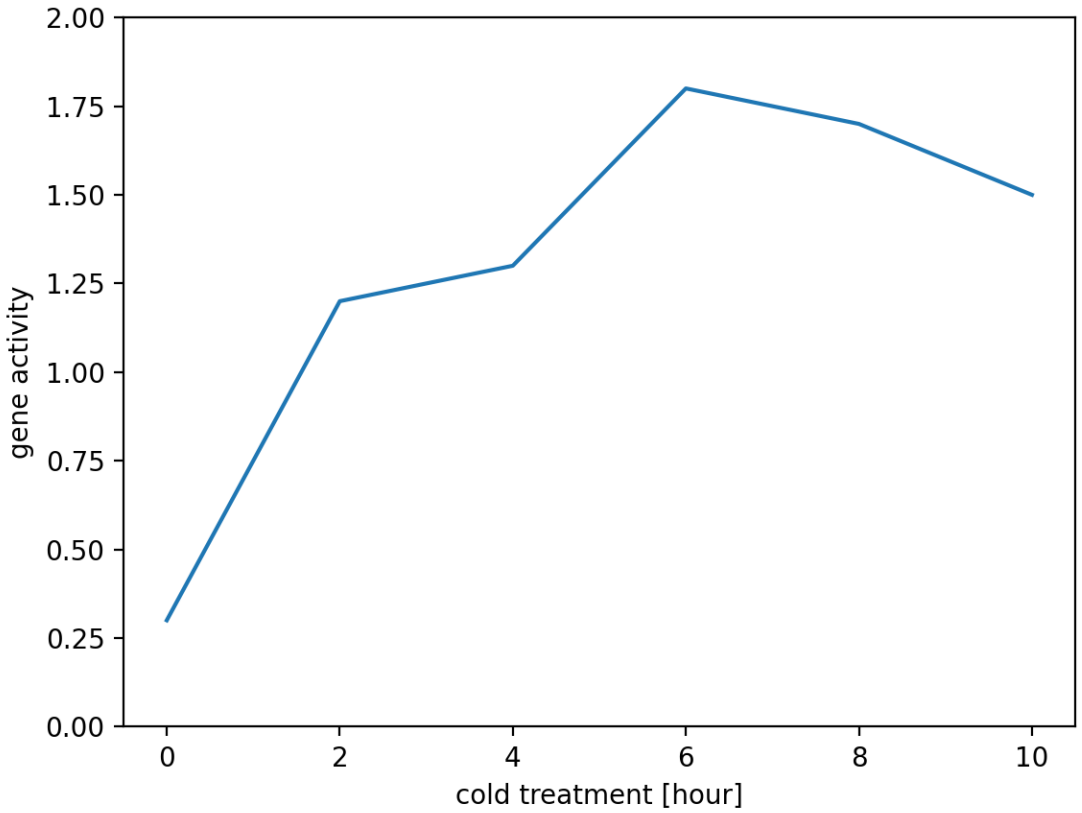
ヒストグラムは、1 変量の連続値データの分布を可視化する目的で使われる。

線グラフ



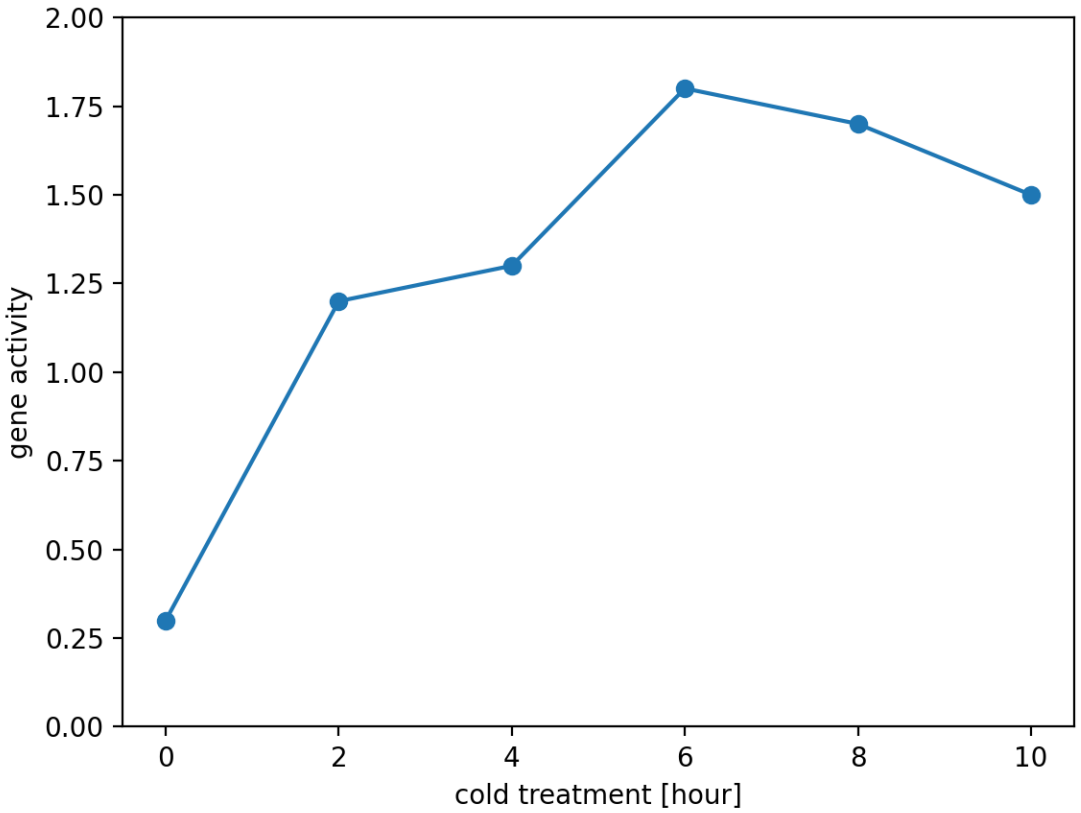
- 線グラフはデータの系列的な変化を見るためのグラフ
- 縦軸および横軸は連続量
- 原点 (0, 0) が省略されることがある
- 観測値を強調するために、観測値に点を明示することもある

線グラフ



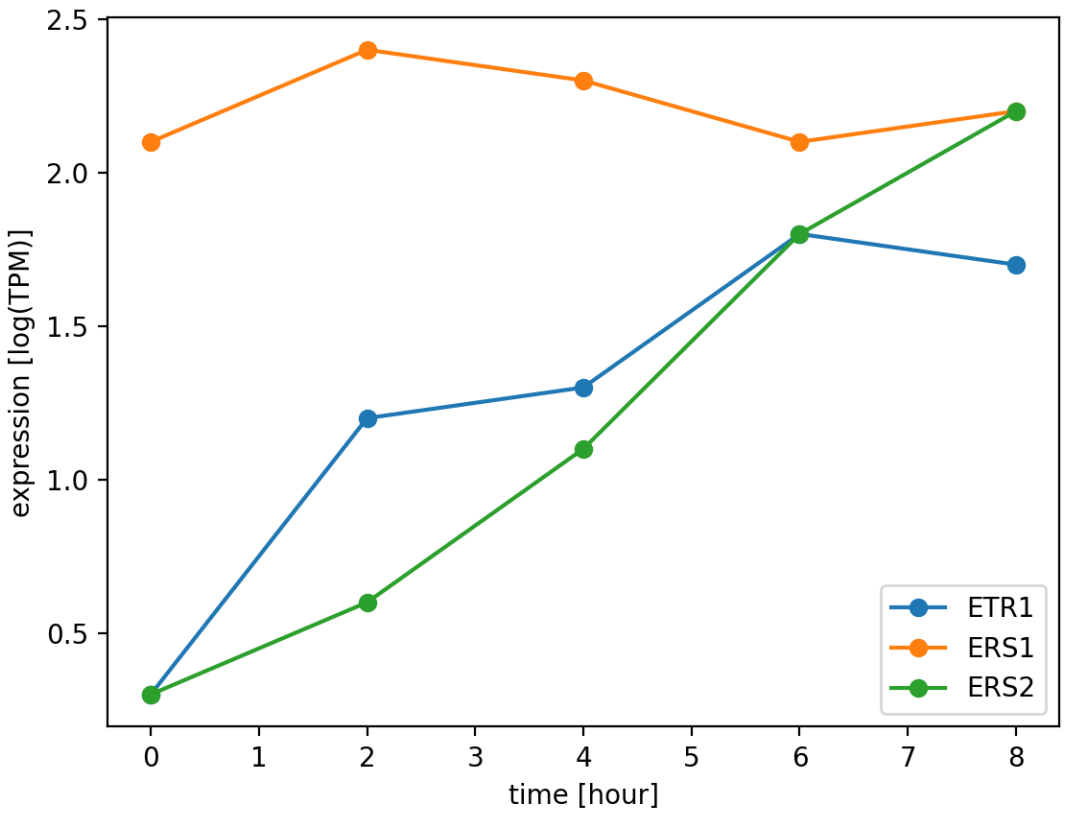

```
import matplotlib.pyplot as plt
import numpy as np

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()
```

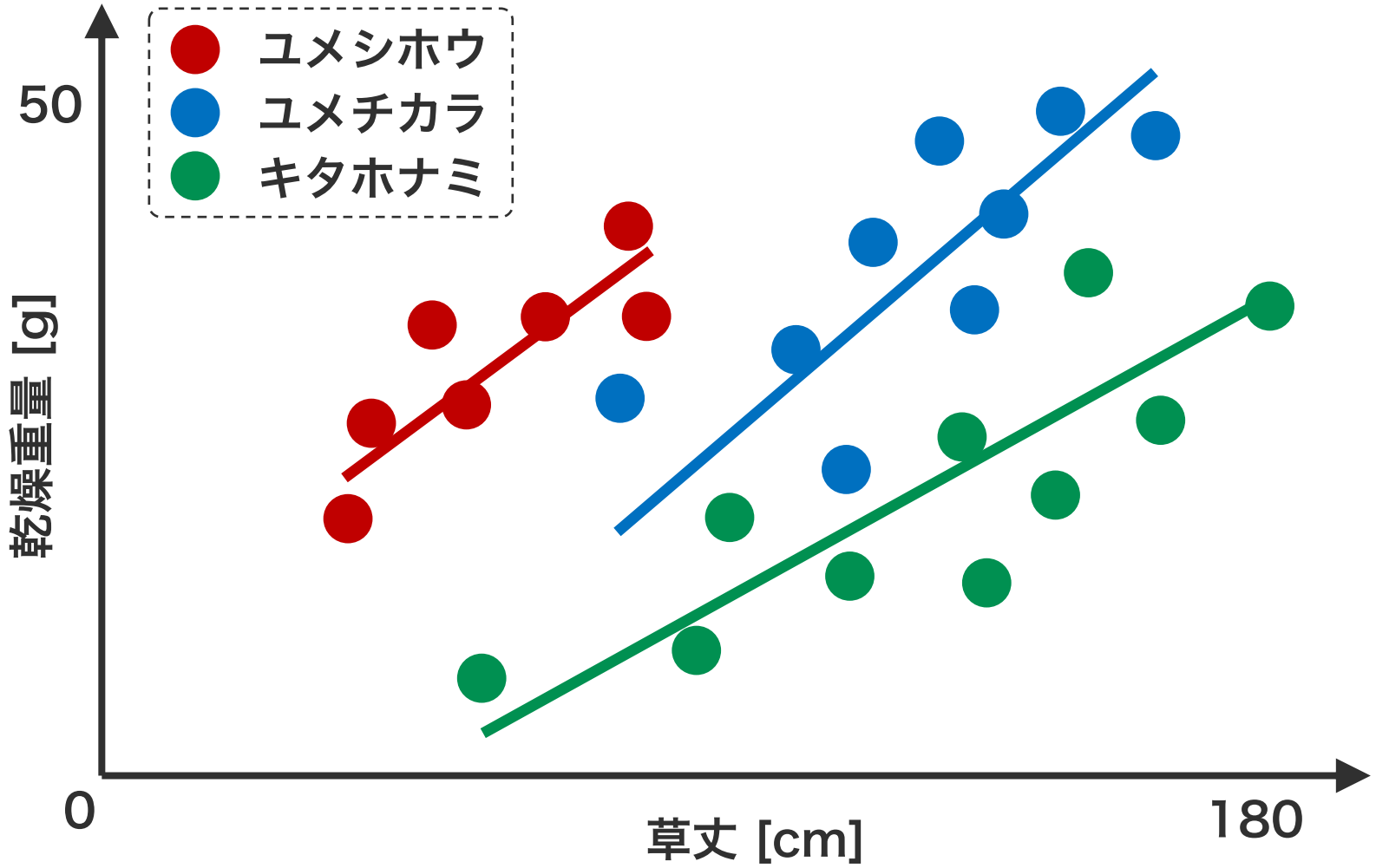
線グラフ




```
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')
```

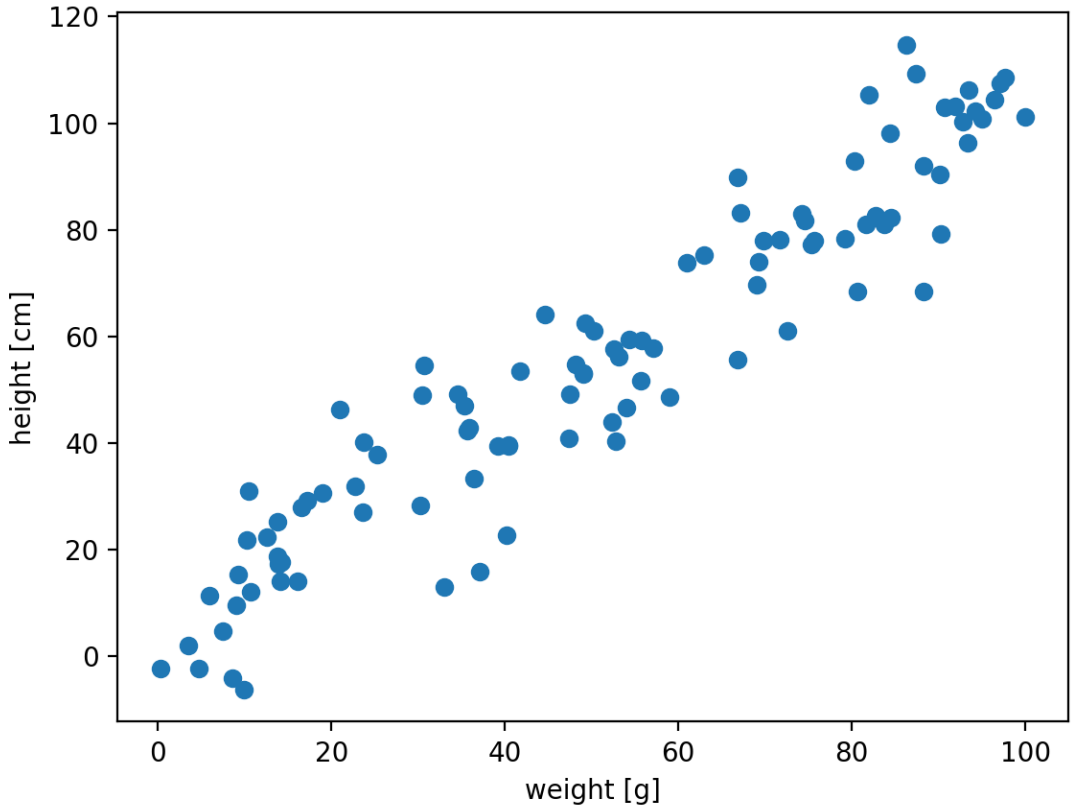
plot メソッドを複数回使うことで、複数の線グラフを描くことができる。グラフを描く際に色を指定しない場合は、自動的に配色される。

散布図



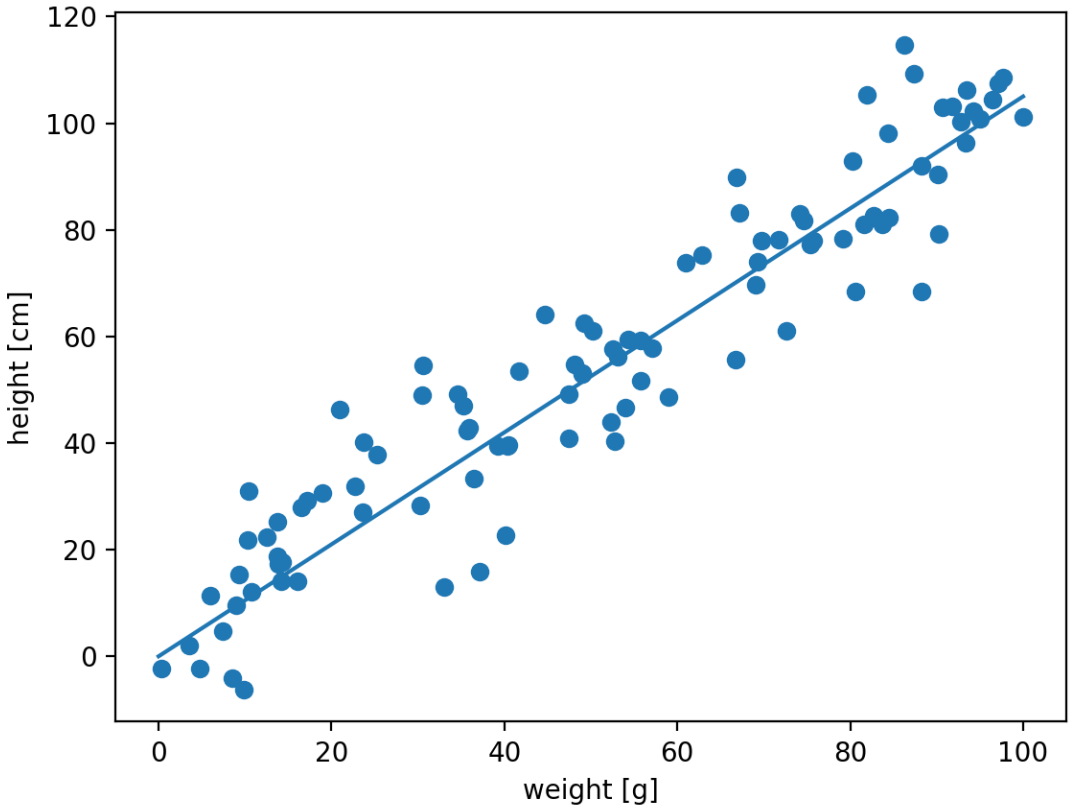
- 多変量データ同士の相関や分布などを見るためのグラフ
- 縦軸および横軸は連続量
- 原点（0, 0）が省略されることがある
- 回帰直線との併用もよく見られる

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

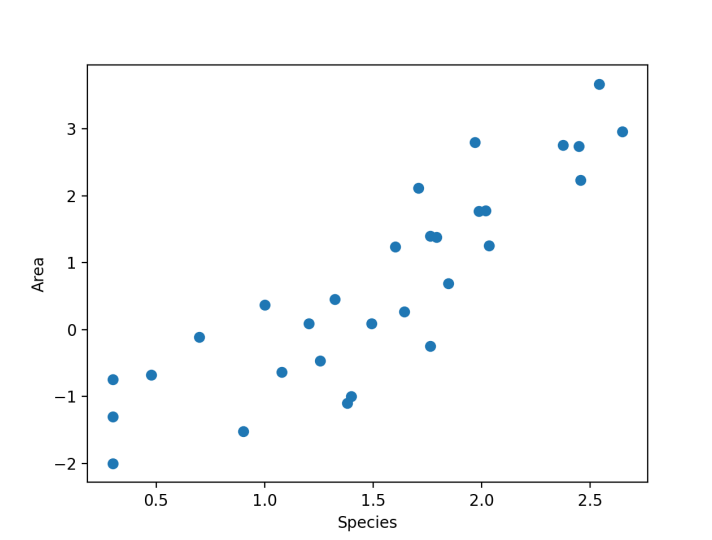



```
import matplotlib.pyplot as plt
import numpy as np

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

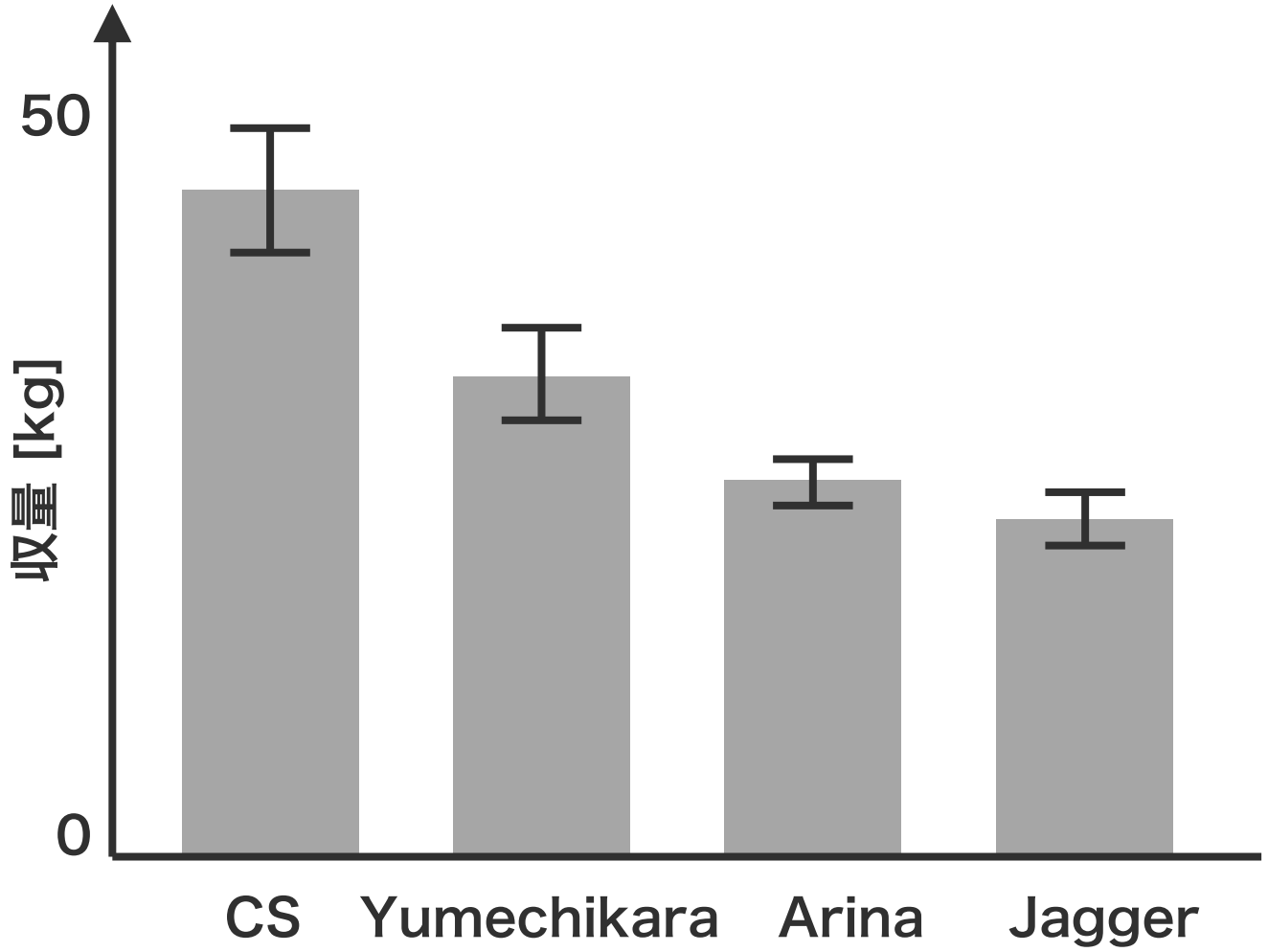
diversity_galapagos.txt には、ガラパゴス島における種の多様性データが記載されている。このデータを読み込み、島の面積（Area）と種数（Species）の関係を散布図で描け。

```
import pandas as pd

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



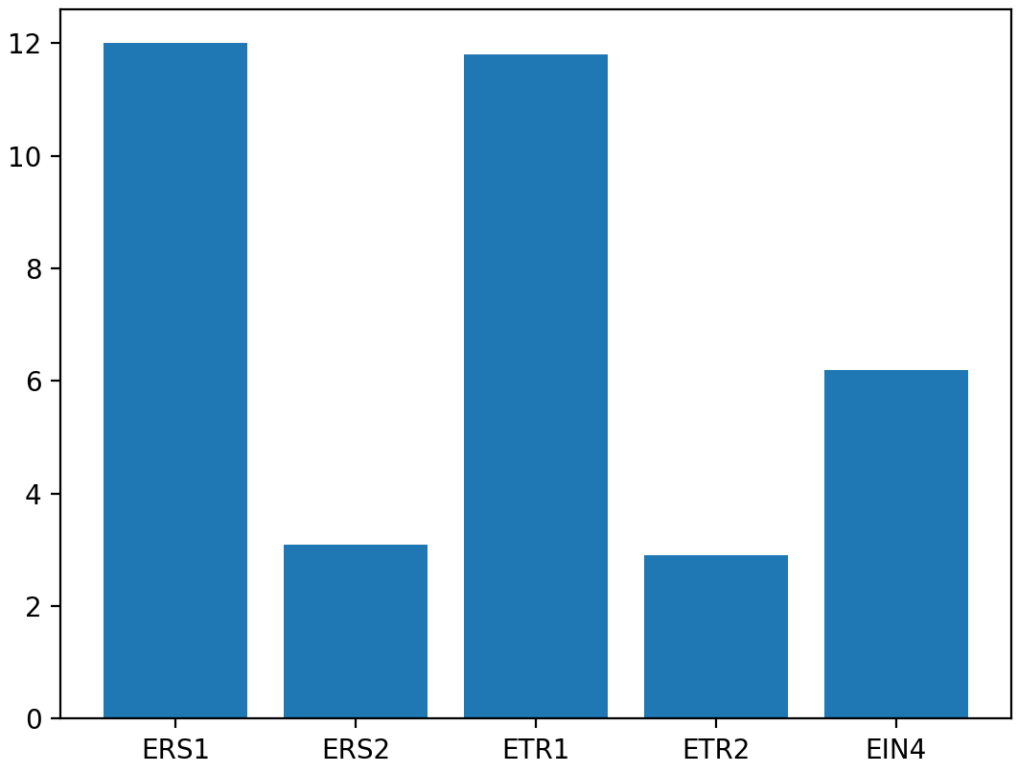
https://aabbdd.jp/notes/data/diversity_galapagos.txt

棒グラフ



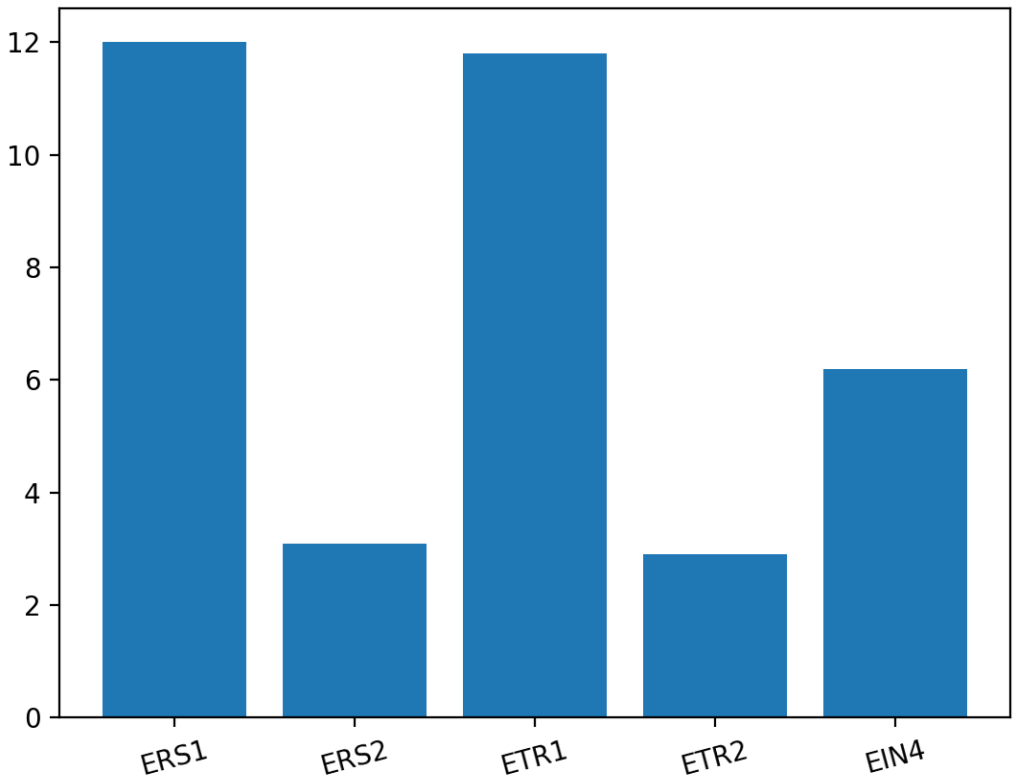
- 複数のカテゴリに属している値同士の大小を可視化するためのグラフ
- 縦軸が連続量、横軸がカテゴリ、あるいはその逆
- 人を騙す目的で使用するとき、原点をゼロ以外の値にしたり、縦軸の目盛り間隔を操作すると効果的である
- エラーバーとともに用いられることがある

棒グラフ



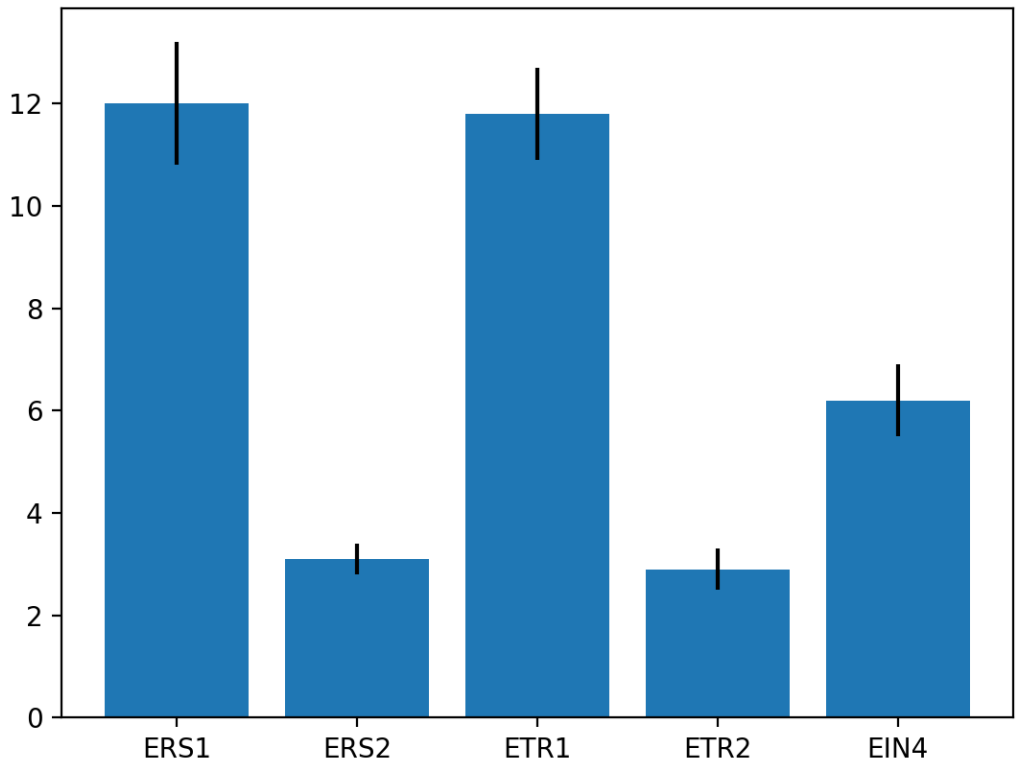

```
import matplotlib.pyplot as plt
import numpy as np

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



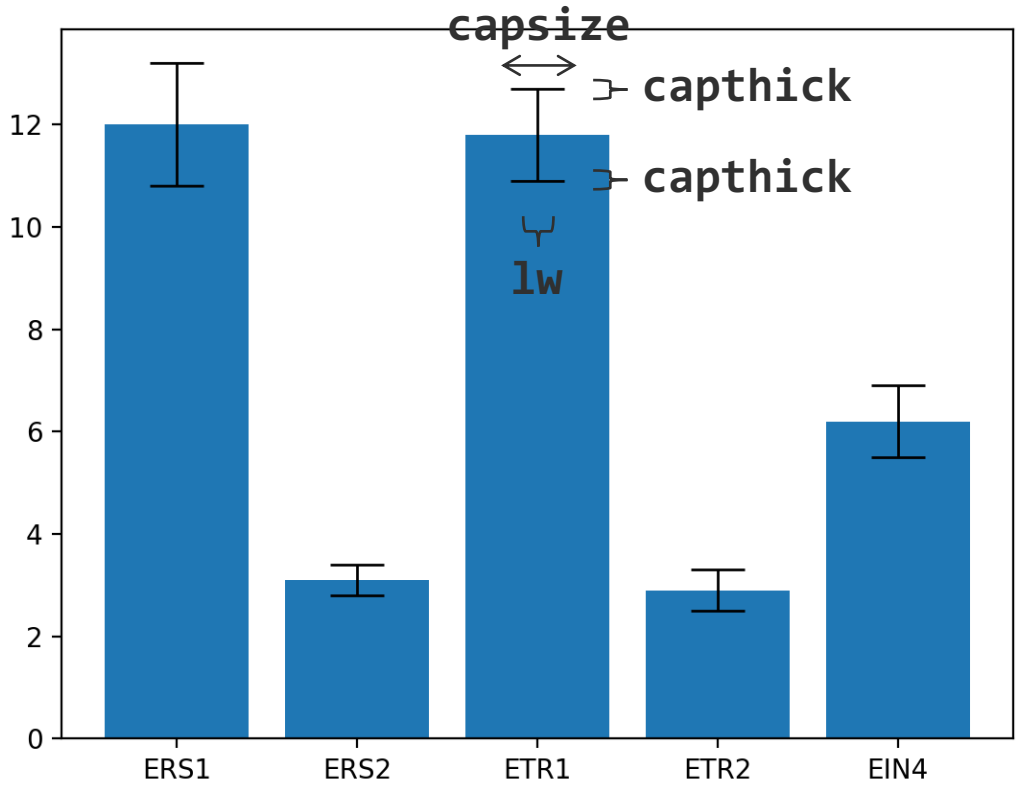

```
import matplotlib.pyplot as plt
import numpy as np

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



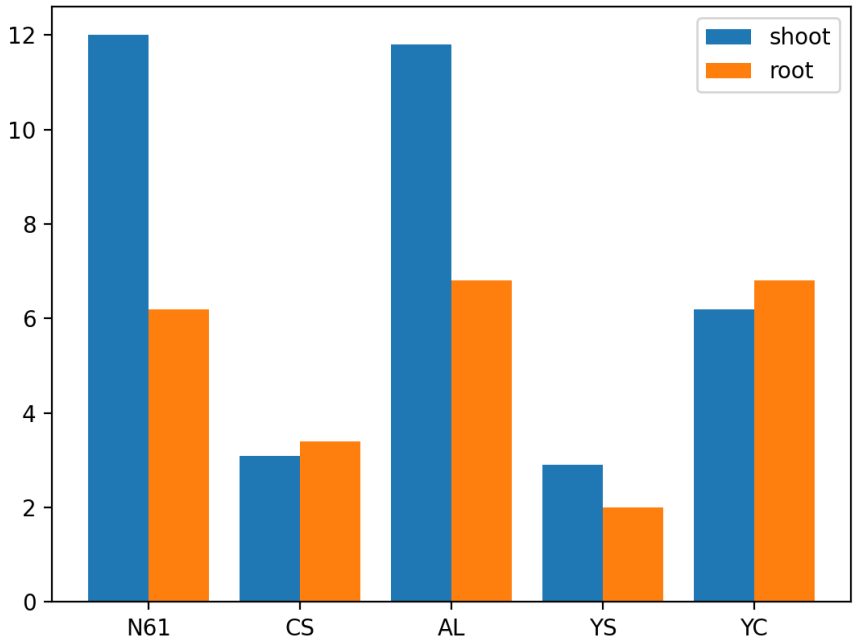

```
import matplotlib.pyplot as plt
import numpy as np

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()
```

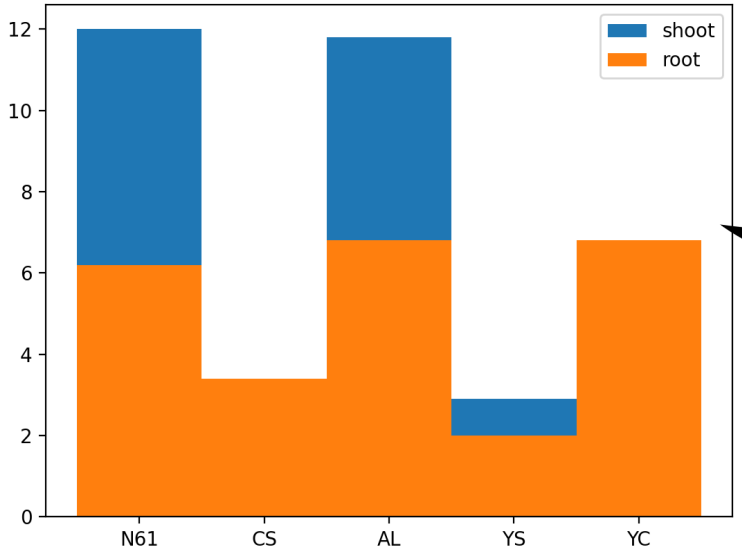
棒グラフ



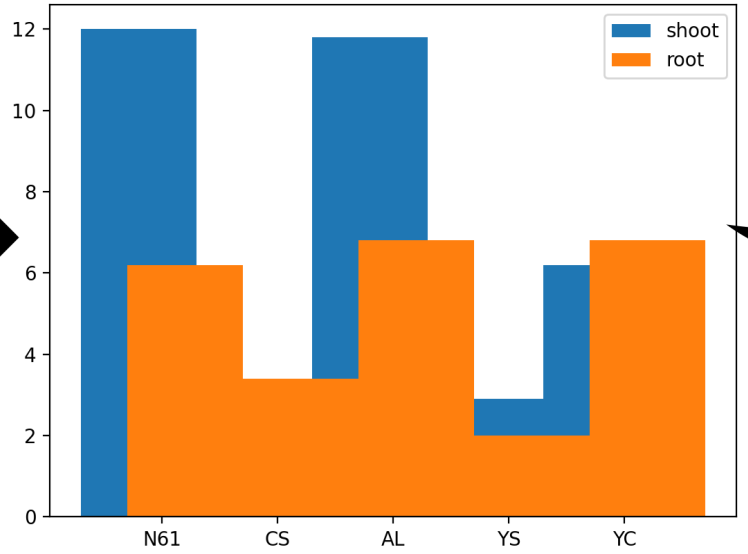

```
import matplotlib.pyplot as plt
import numpy as np

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

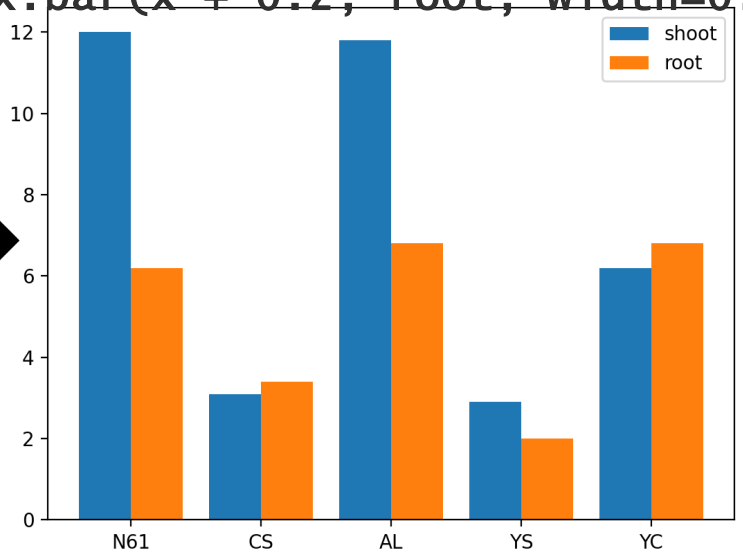
```
ax.bar(x, shoot)
ax.bar(x, root)
```



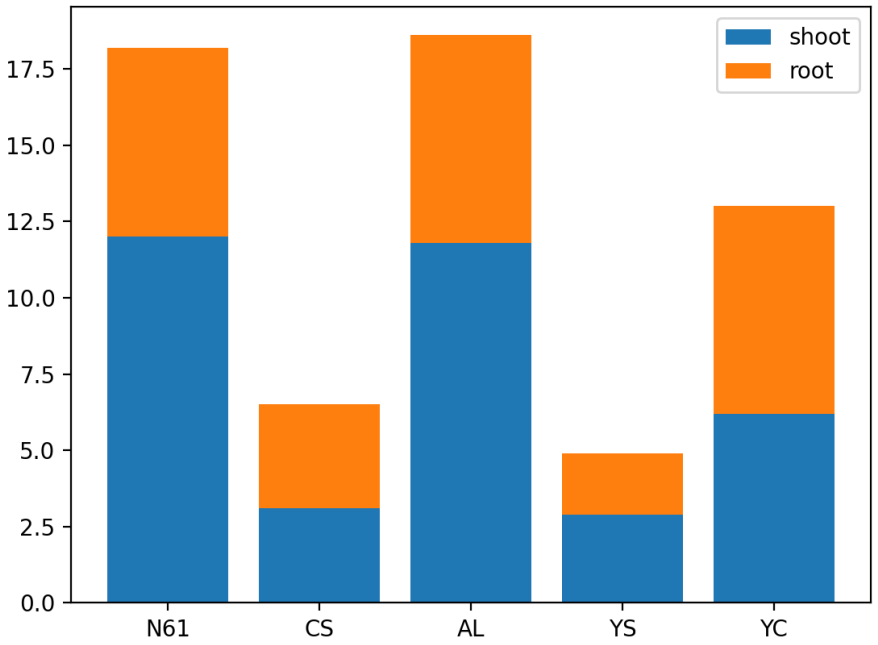

```
ax.bar(x - 0.2, shoot)
ax.bar(x + 0.2, root)
```




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



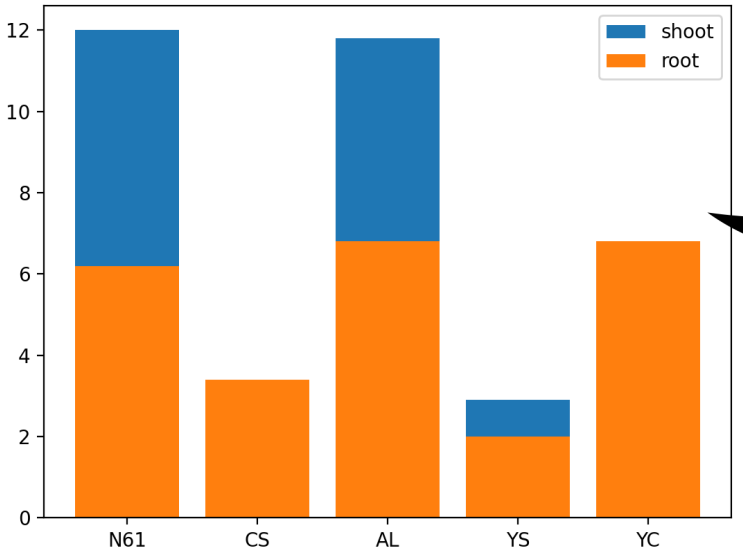
棒グラフ



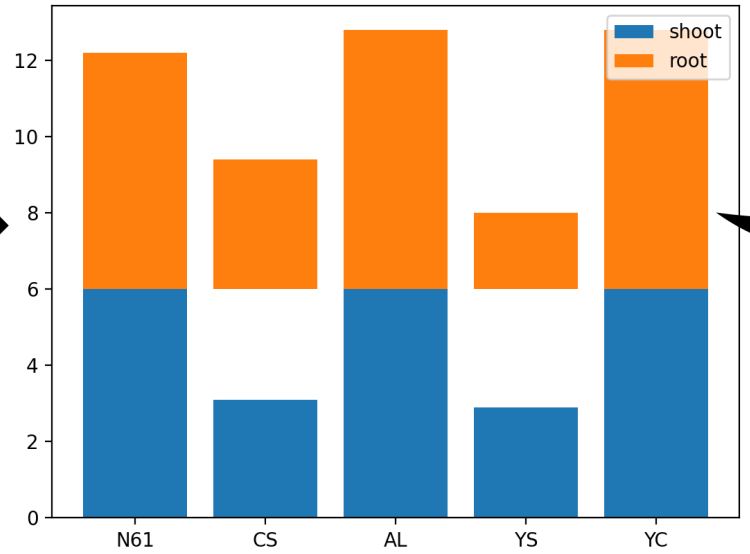

```
import matplotlib.pyplot as plt
import numpy as np

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

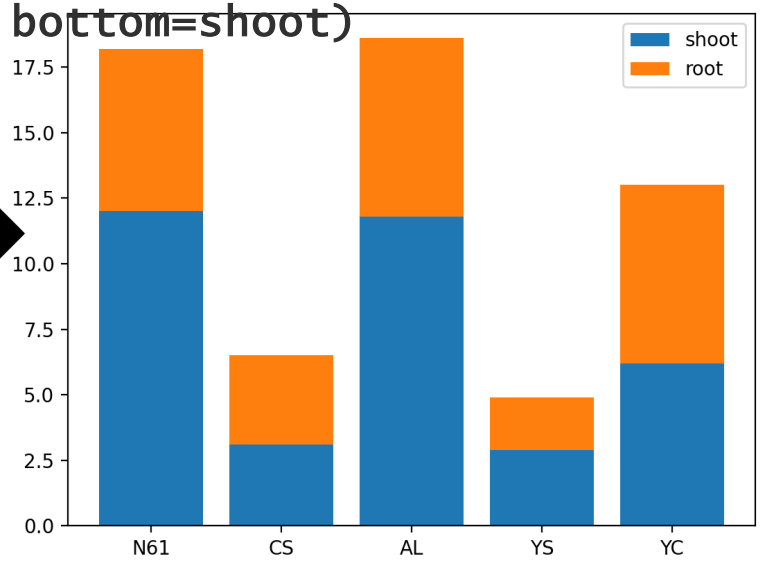
```
ax.bar(x, shoot)
ax.bar(x, root)
```



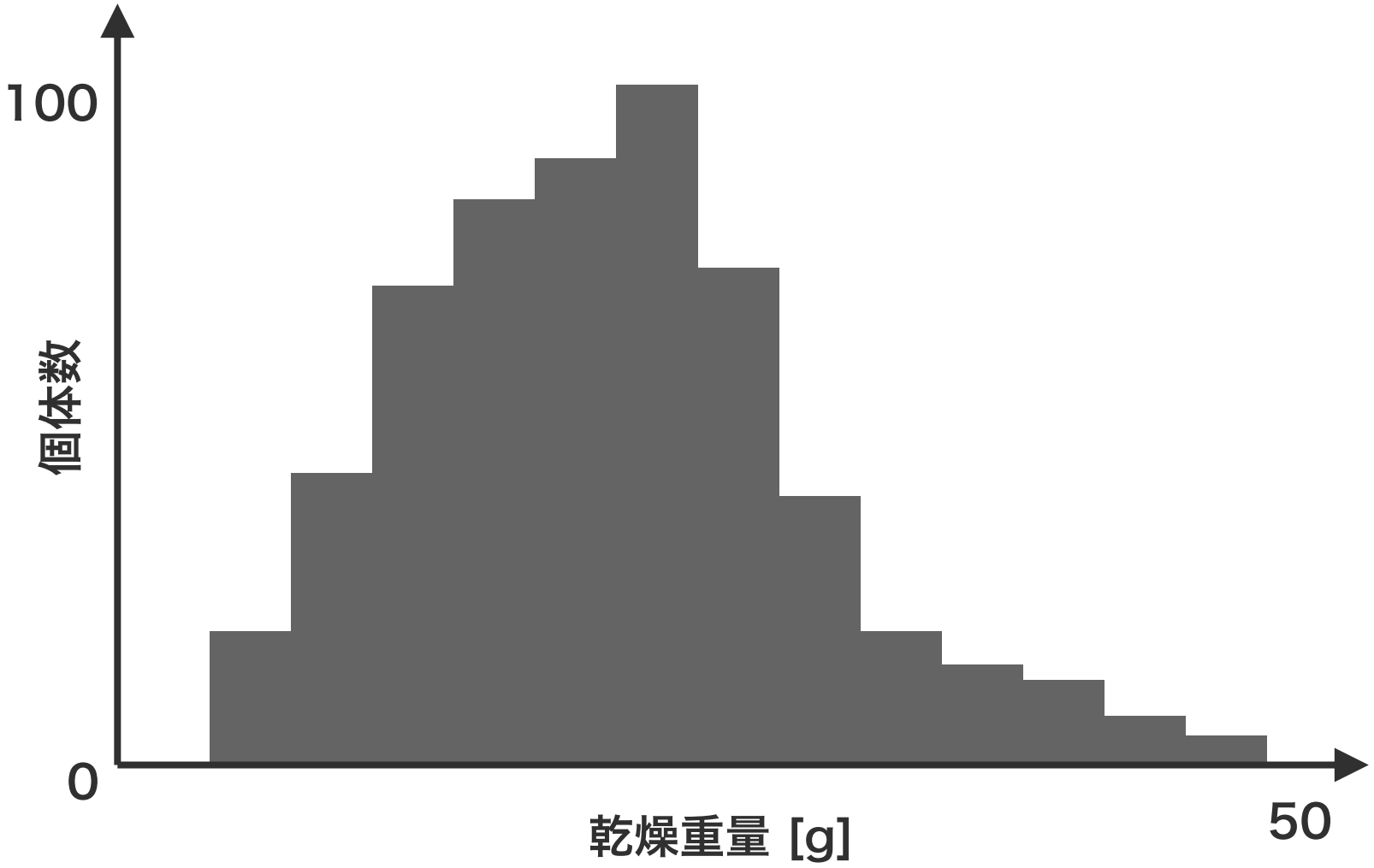

```
ax.bar(x, shoot)
ax.bar(x, root, bottom=6)
```




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

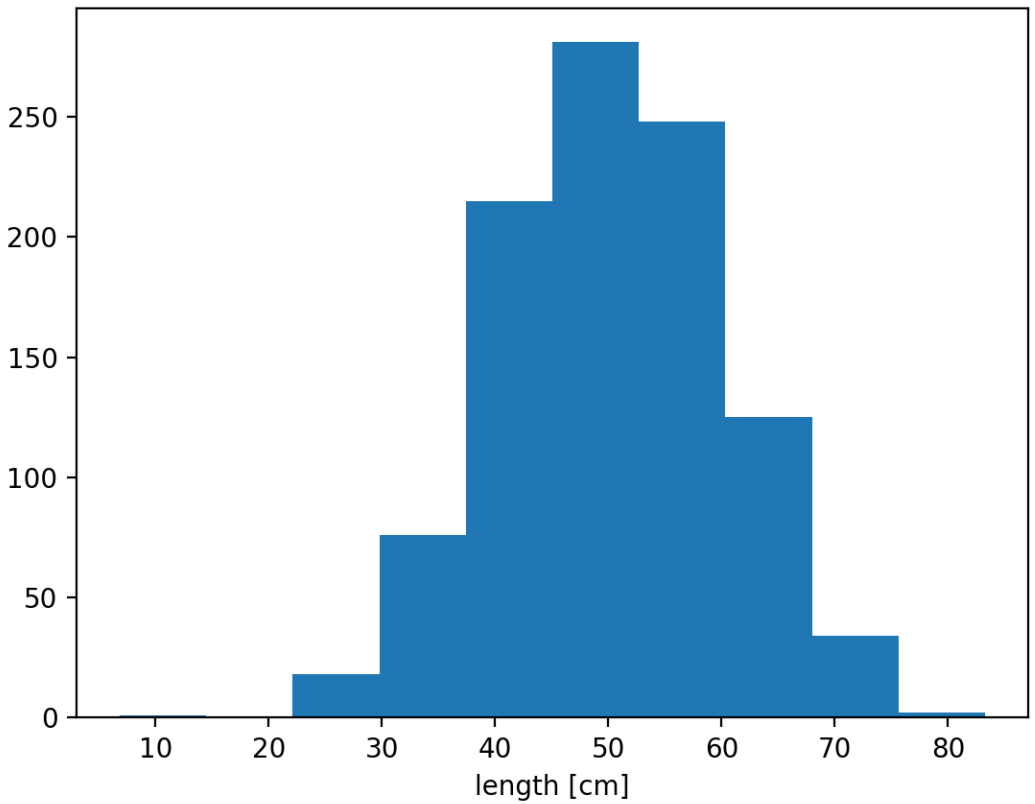


ヒストグラム



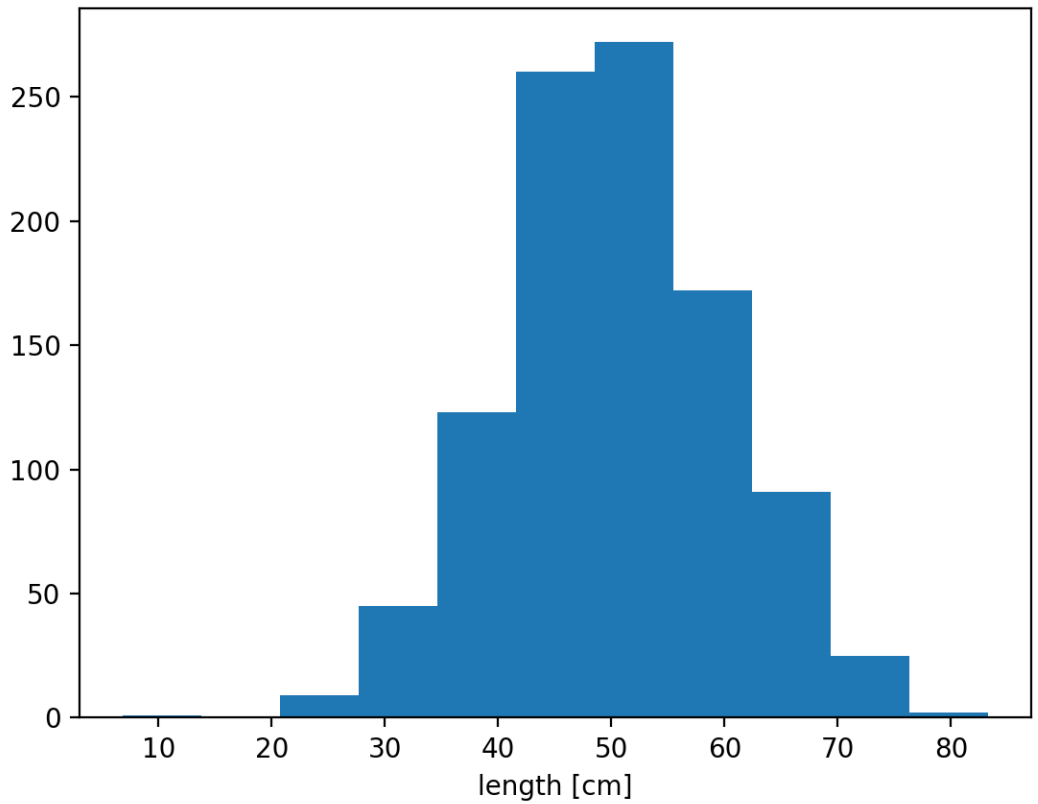
- １変量の連続値データを可視化するためのグラフ
- 横軸が連続量であり、縦軸は頻度・個数または確率である
- 横幅は恣意的（経験的）に決めることもあれば、スタージェスの公式などで決めることもある

ヒストグラム



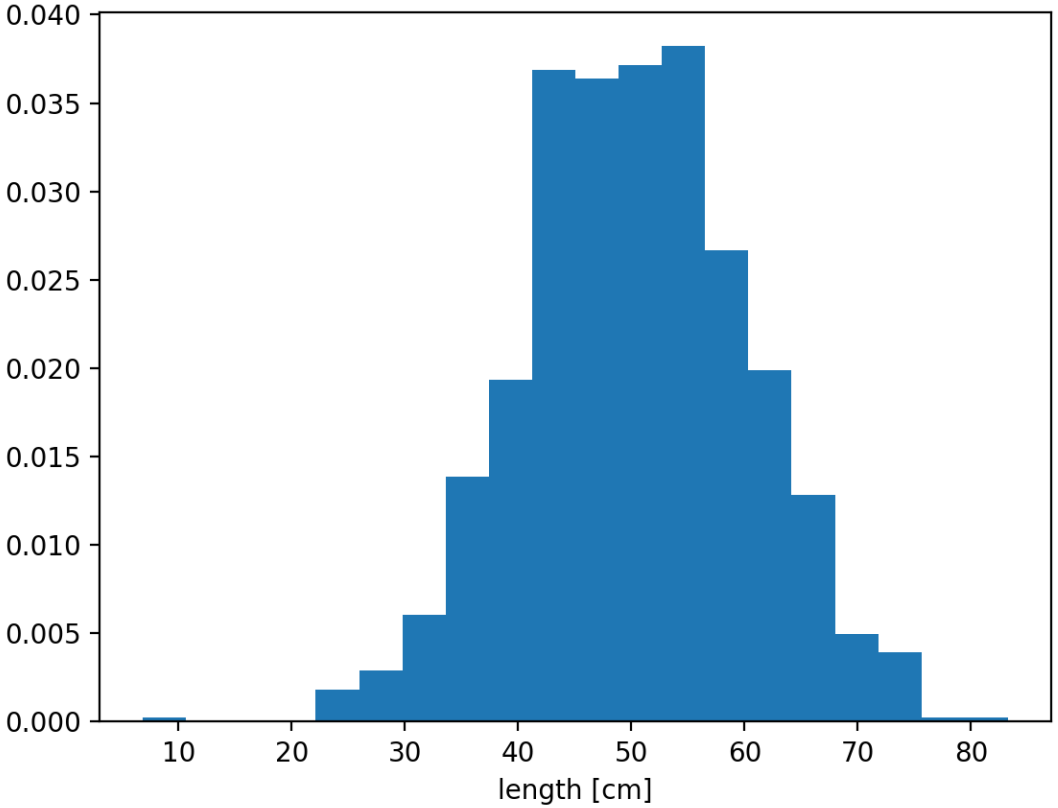

```
import numpy as np
import matplotlib.pyplot as plt

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()
```

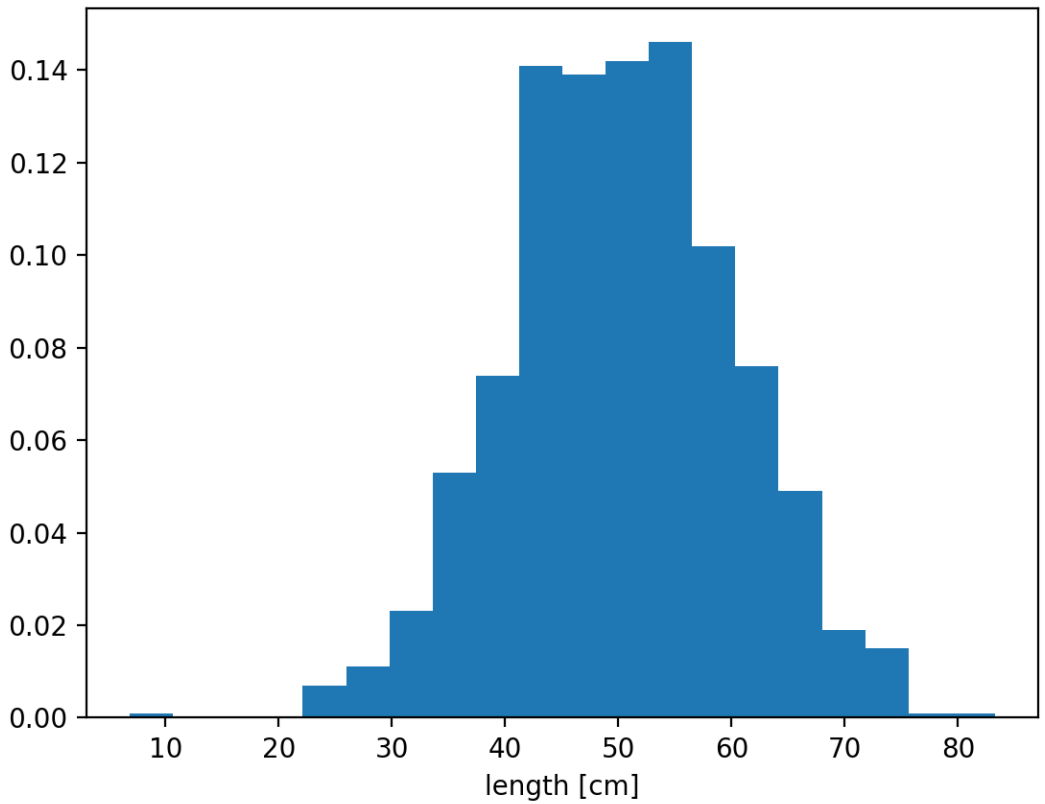
ヒストグラム




```
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



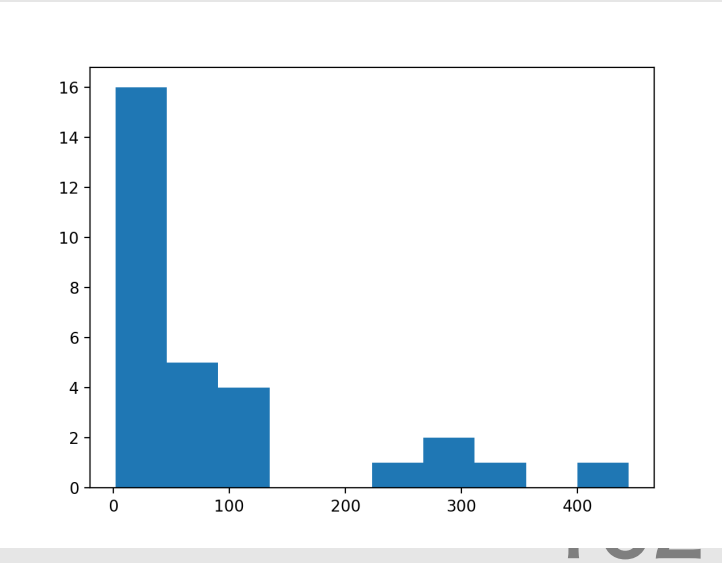

```
import numpy as np
import matplotlib.pyplot as plt

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。

問題 M1-2

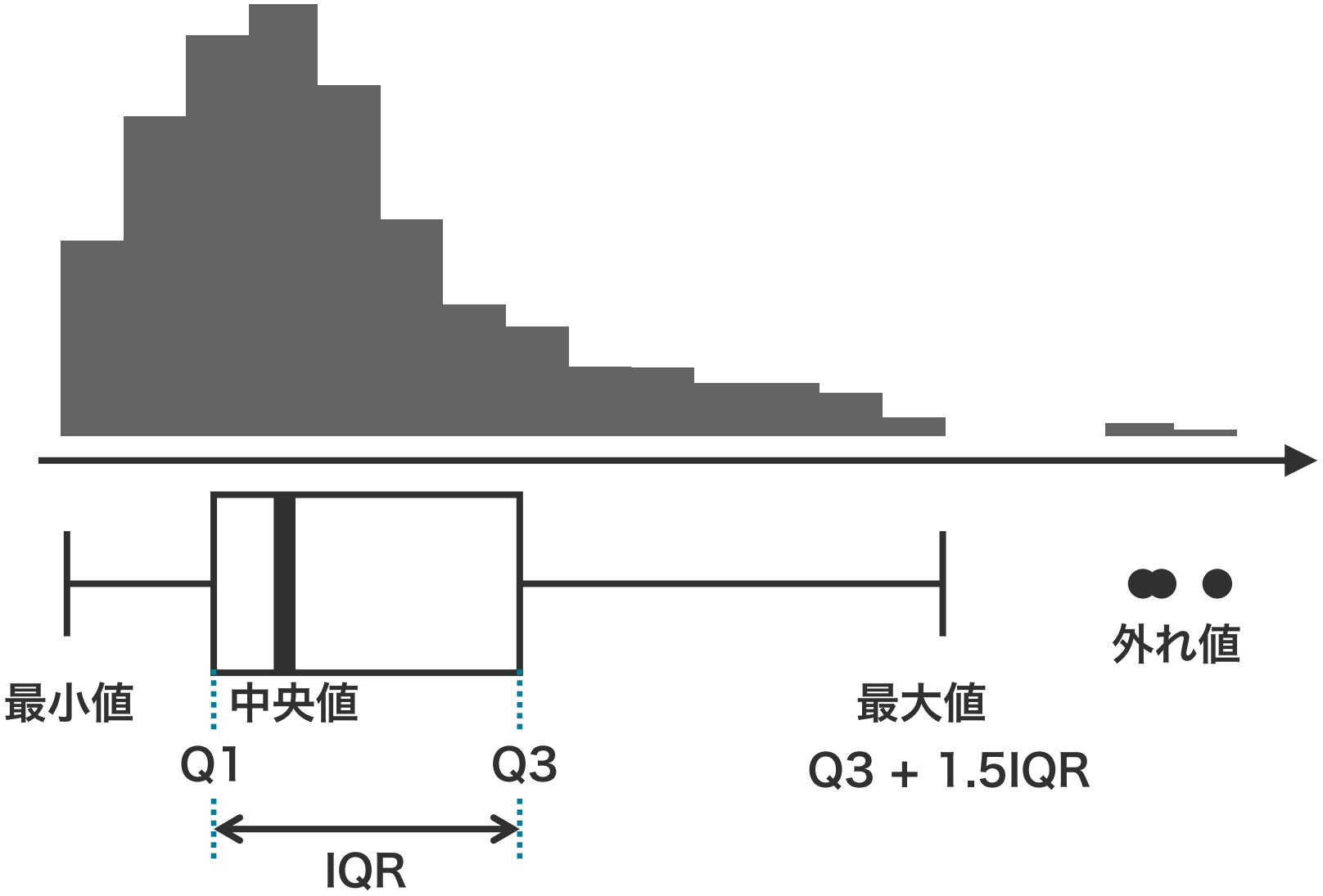
diversity_galapagos.txt には、ガラパゴス島における種の多様性データが記載されている。このデータを読み込み、このデータセットにおける種数（Species）の分布をヒストグラムで描け。

```
import pandas as pd

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



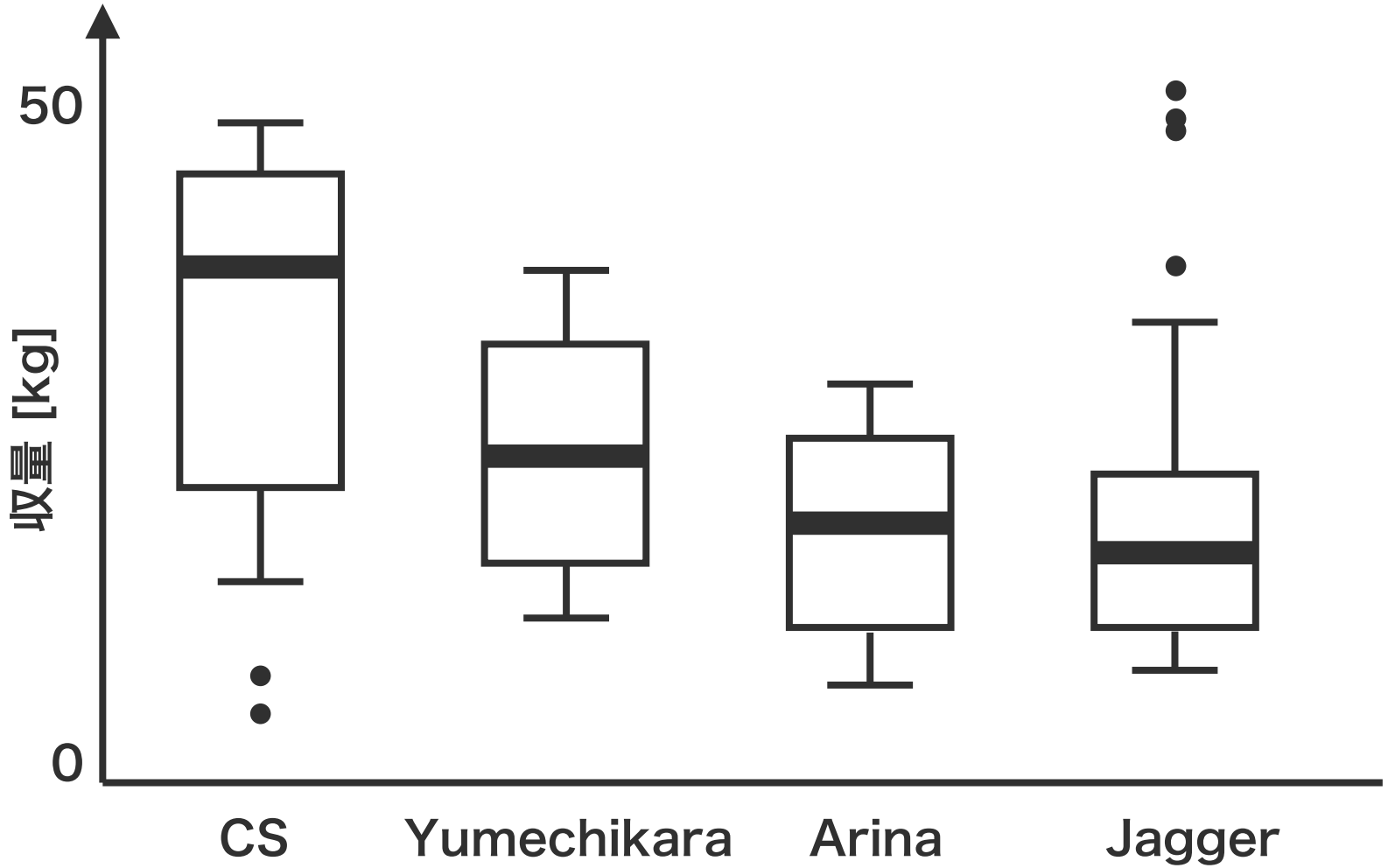
https://aabbdd.jp/notes/data/diversity_galapagos.txt

ボックスプロット



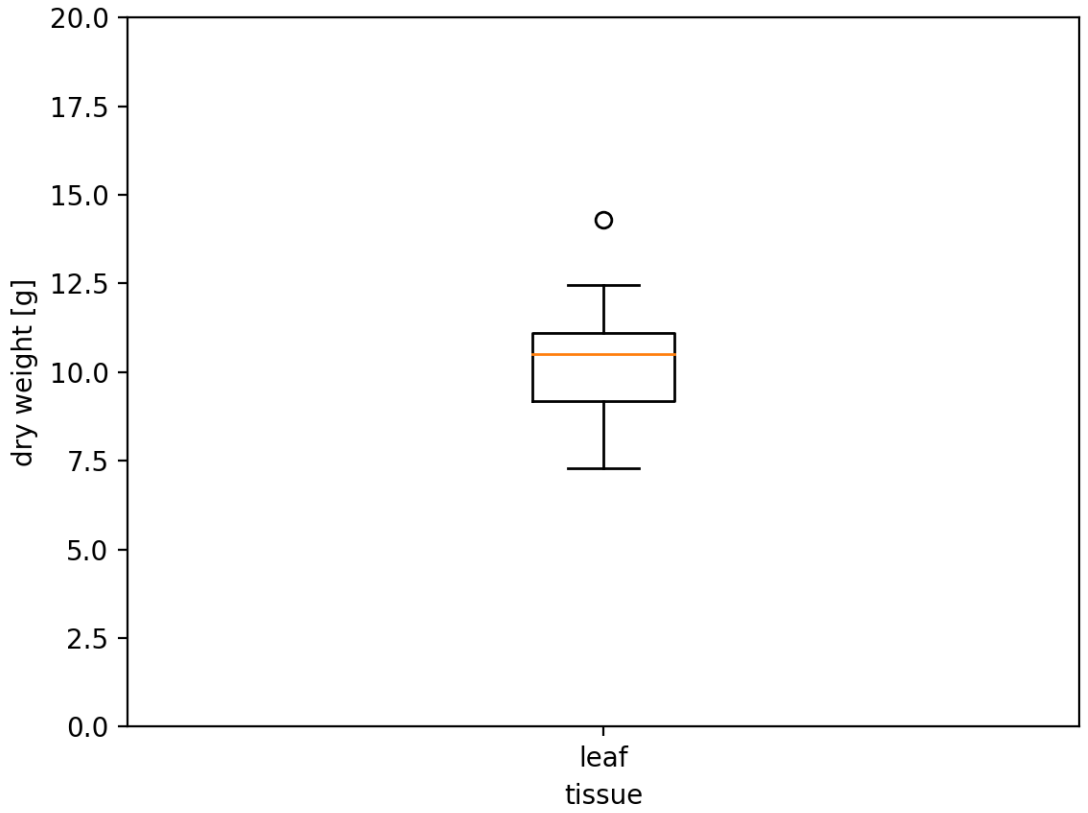
- 複数の 1 変量の連続値データを可視化するためのグラフ
- 最大値、最小値、第 1 四分位数 Q1、中央値、第 3 四分位数 Q3 などを簡単に確認できる
- [Q1 – 1.5IQR, Q3 + 1.5IQR] の外側にあるデータは外れ値

ボックスプロット



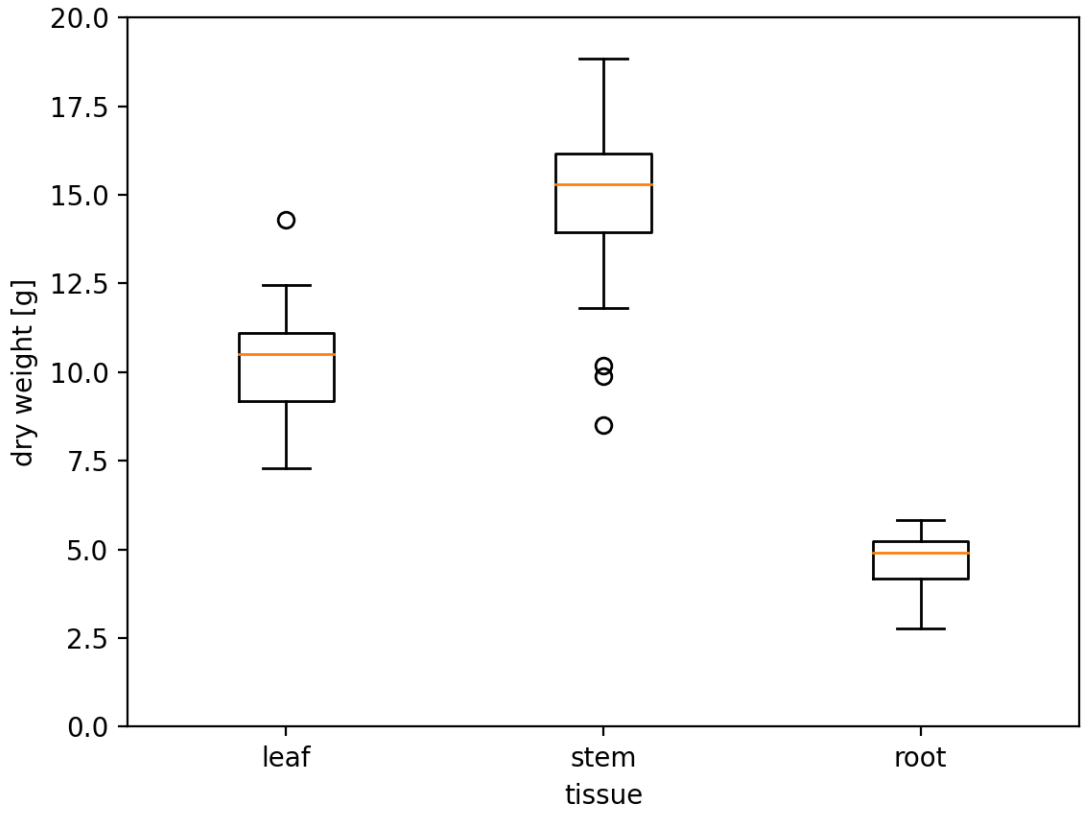
- 複数の 1 変量の連続値データを可視化するためのグラフ
- 最大値、最小値、第 1 四分位数 Q1、中央値、第 3 四分位数 Q3 などを簡単に確認できる
- [Q1 – 1.5IQR, Q3 + 1.5IQR] の外側にあるデータは外れ値
- 複数の変量の分布を同時に比べるときに便利

ボックスプロット



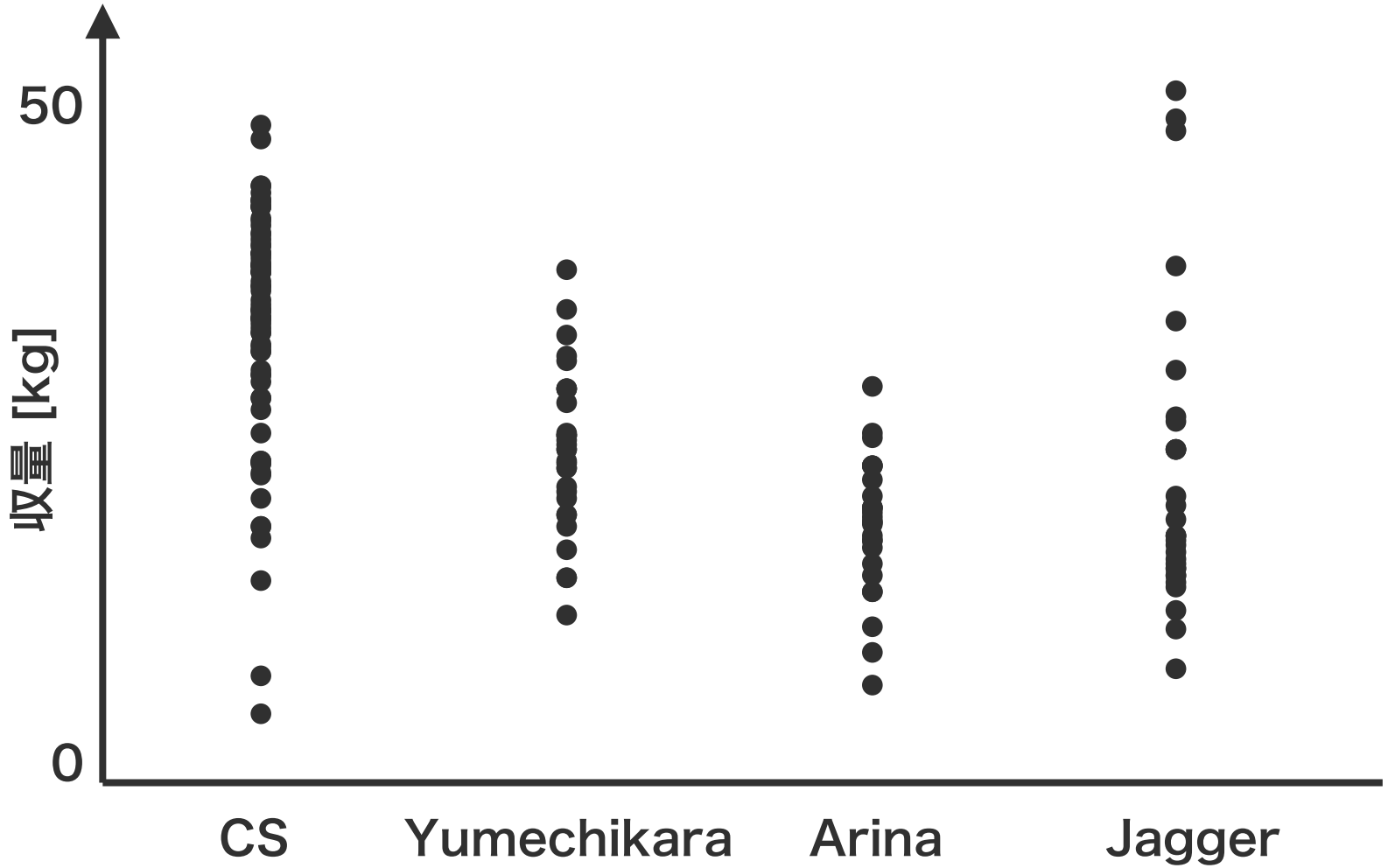

```
import numpy as  np
import matplotlib.pyplot as plt
np.random.seed(2018)

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



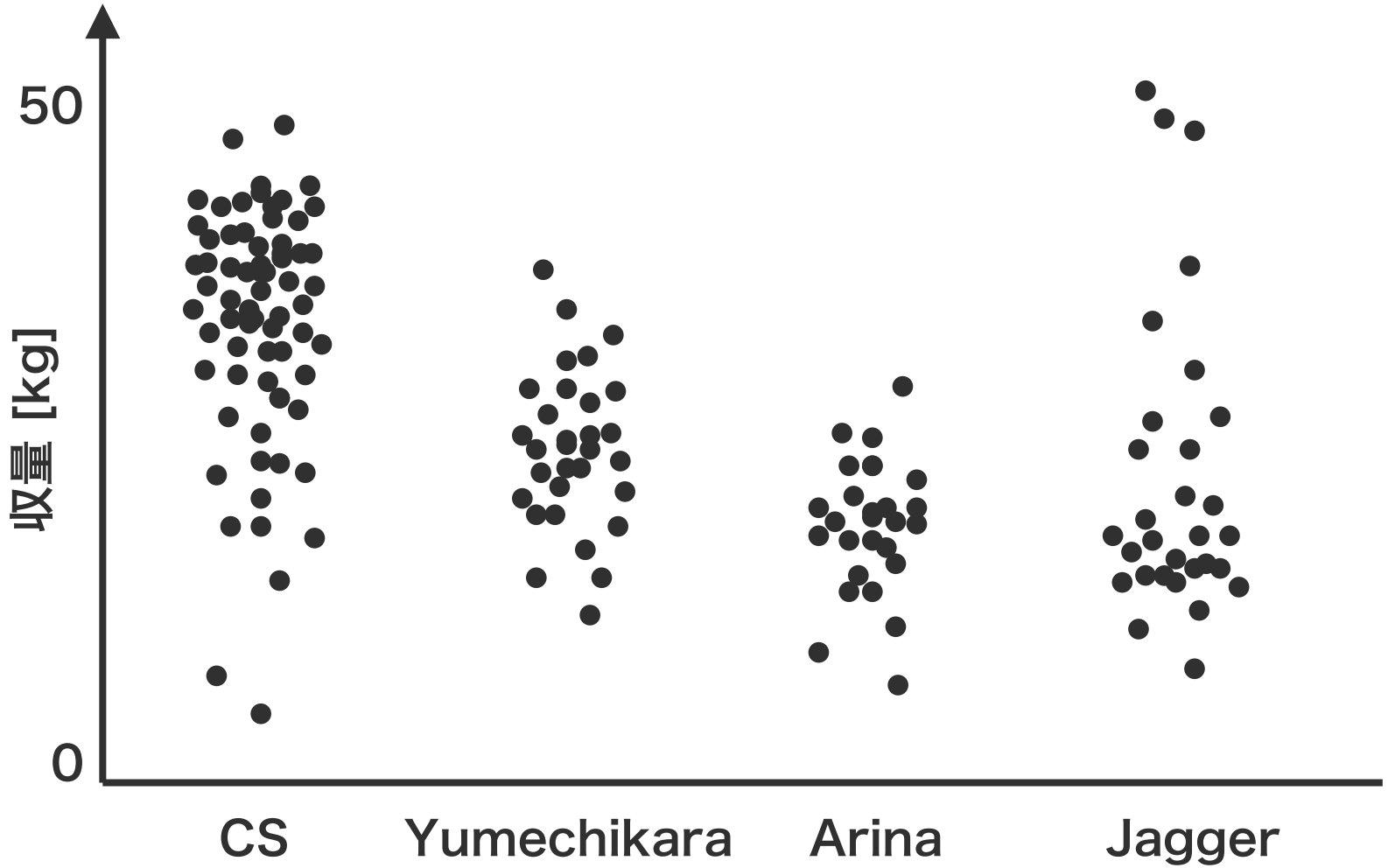

```
import numpy as  np
import matplotlib.pyplot as plt
np.random.seed(2018)

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



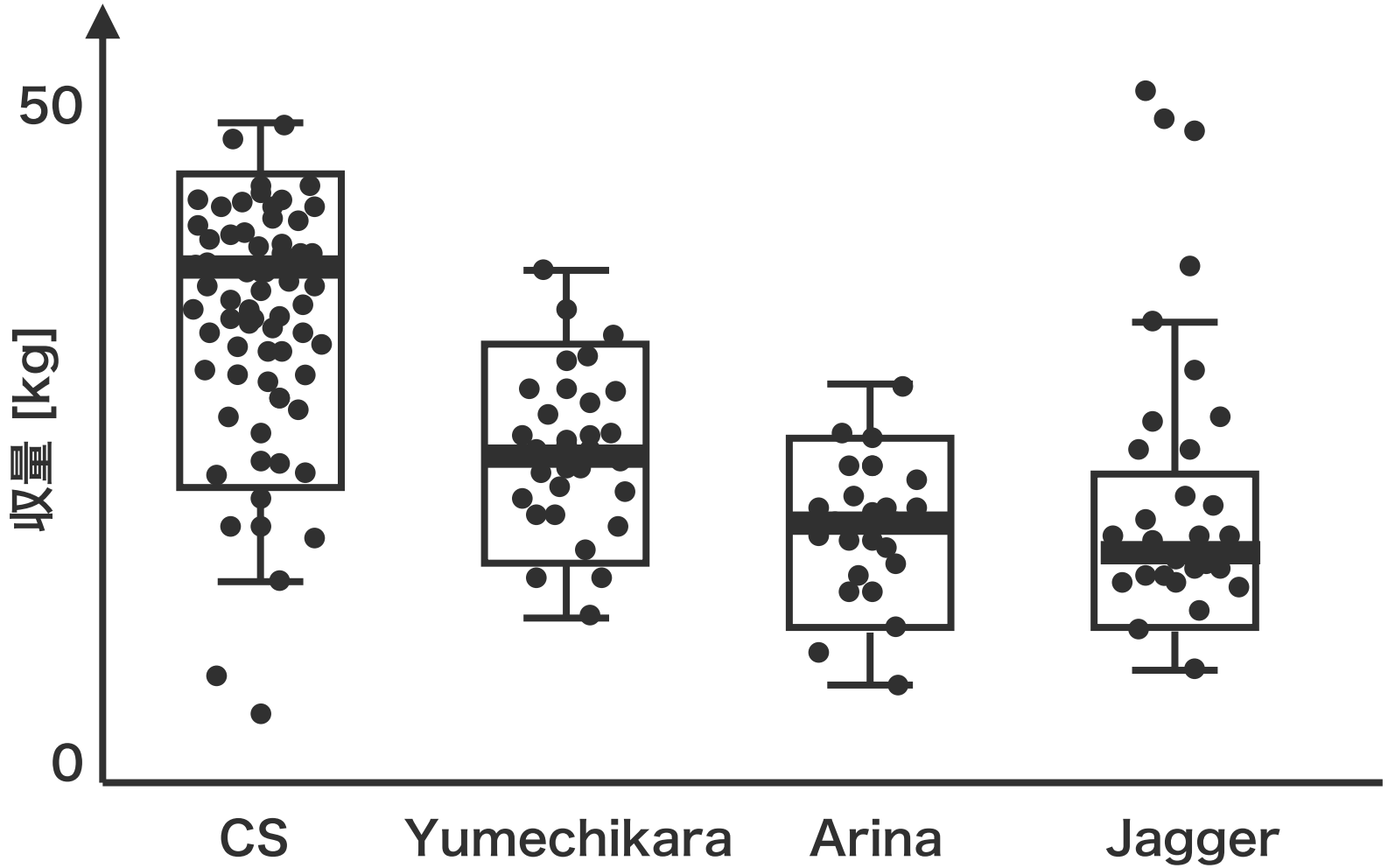
- 連続値データを可視化するためのグラフ、観測値をそのままプロットする
- データの多い所に点が重なるため、点を左右にランダムにずらしてプロットする
- ボックスプロットと合わせて使われる場合もある

jitter



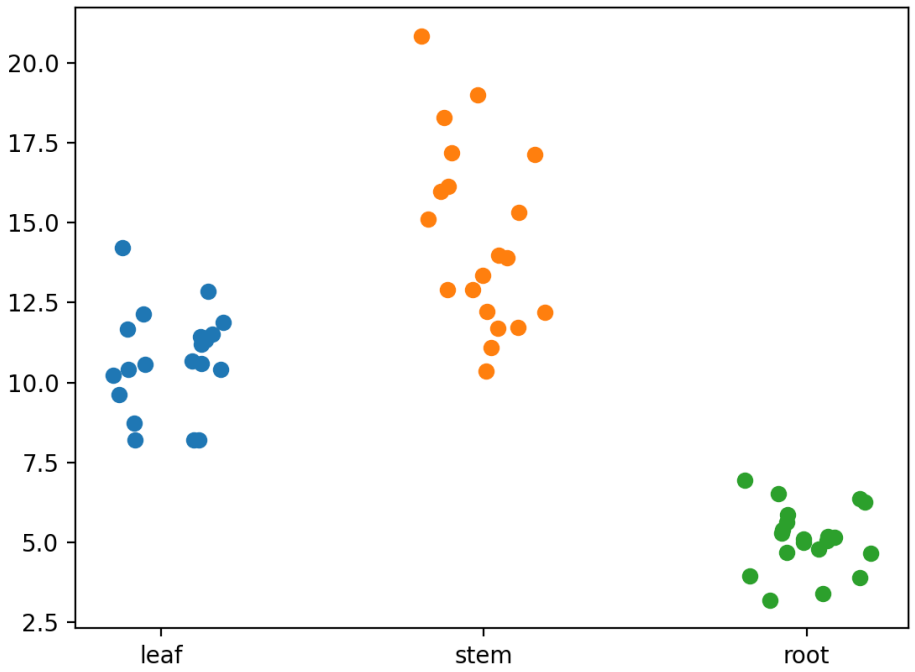
- 連続値データを可視化するためのグラフ、観測値をそのままプロットする
- データの多い所に点が重なるため、点を左右にランダムにずらしてプロットする
- ボックスプロットと合わせて使われる場合もある

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

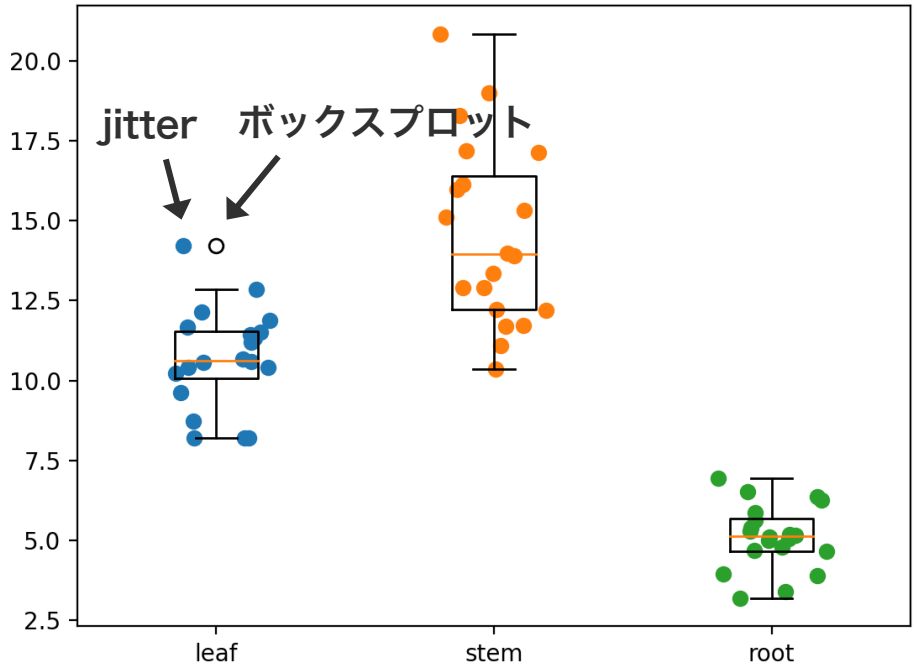
jitter




```
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()
```

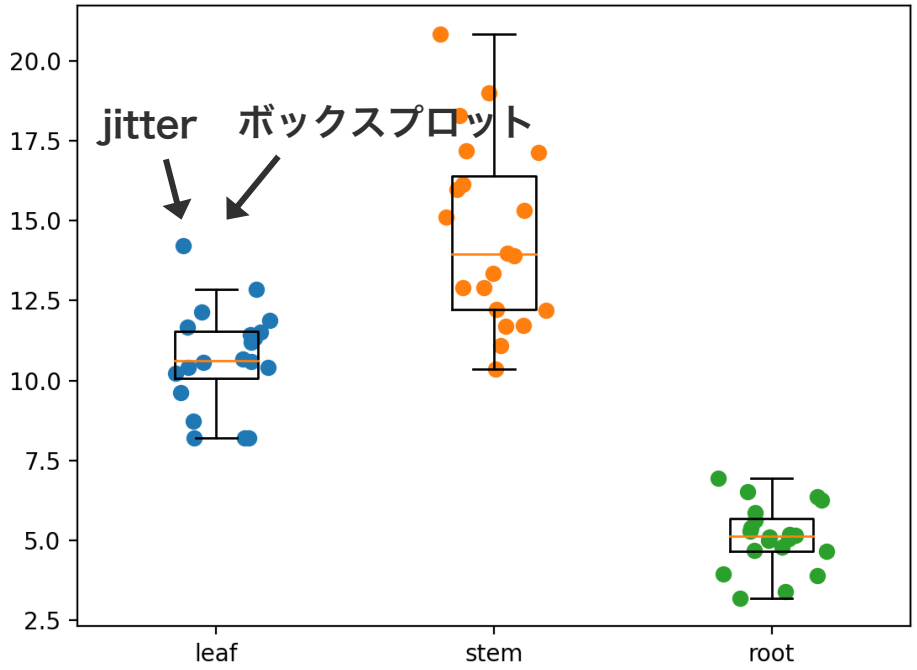
y1、y2、y3 のデータの y 座標をそのままにして、x 座標に乱数を加えて左右にずらす。

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()
```

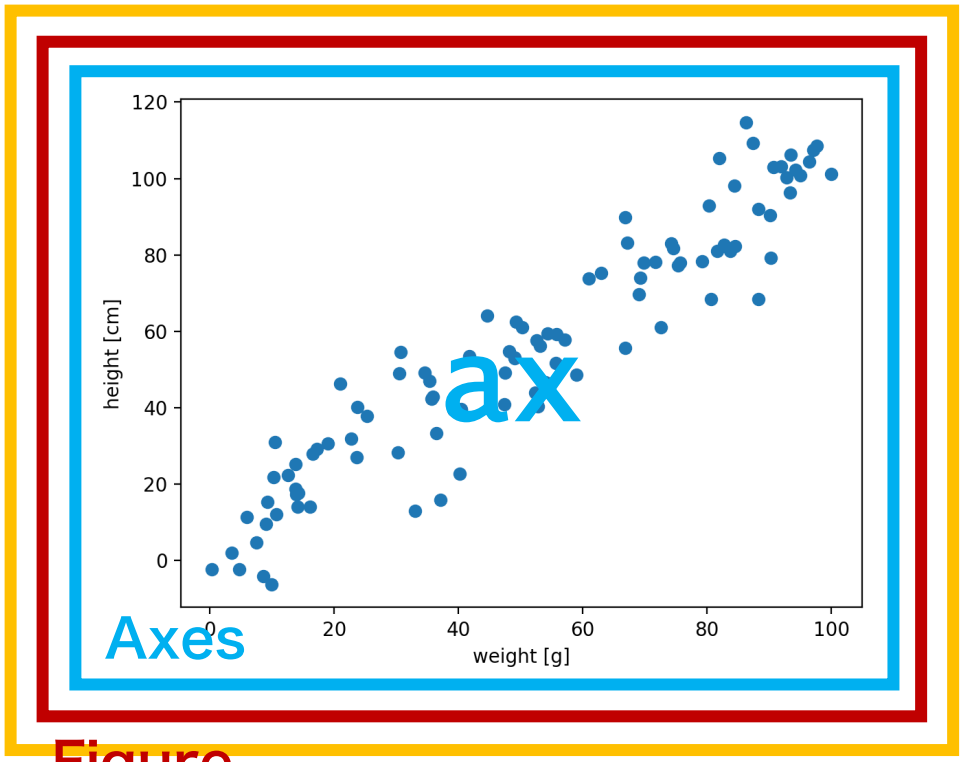
jitter




```
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()
```

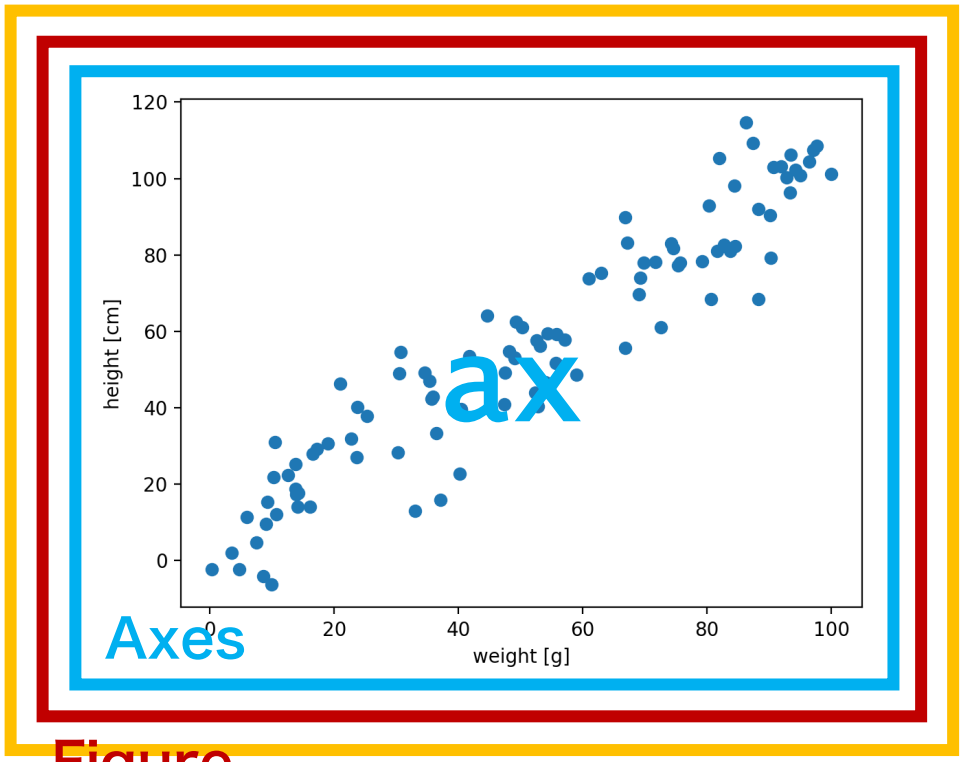
showfliers=False を指定することで、boxplot メソッドは外れ値を描かなくなる。

subplot



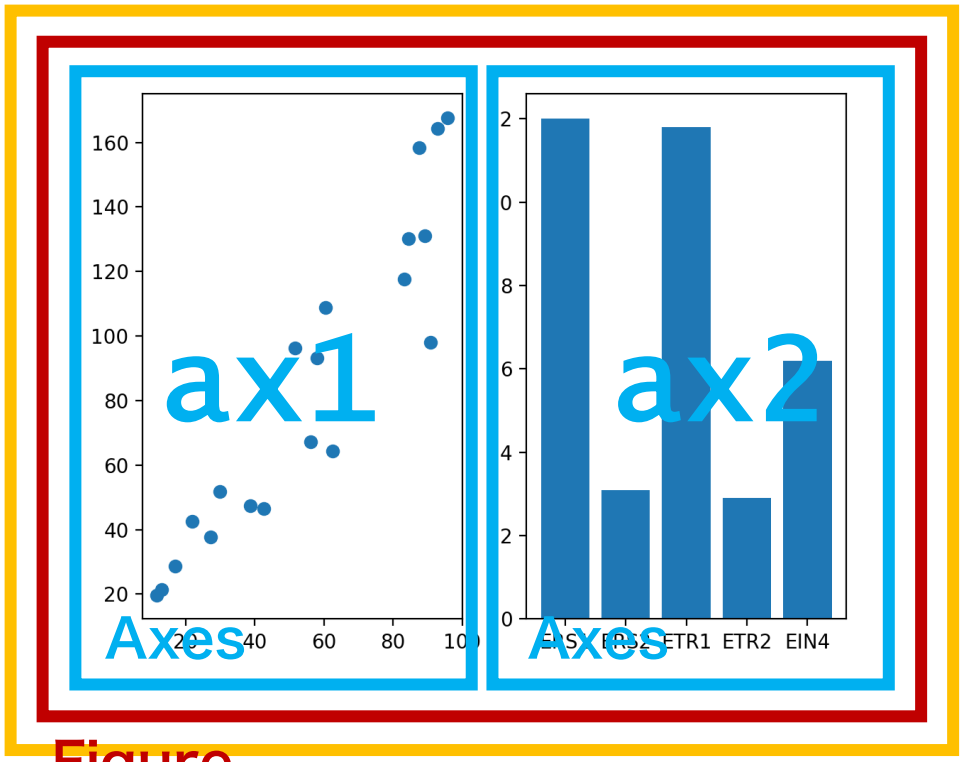

```
import matplotlib.pyplot as plt
import numpy as np

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



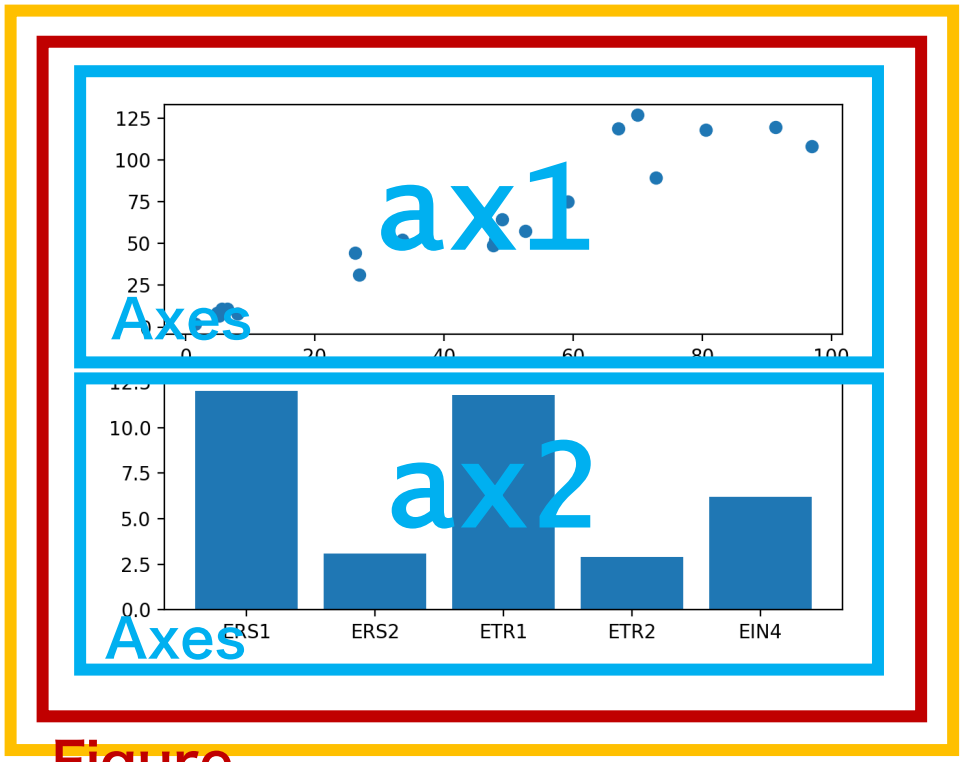

```
import matplotlib.pyplot as plt
import numpy as np

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



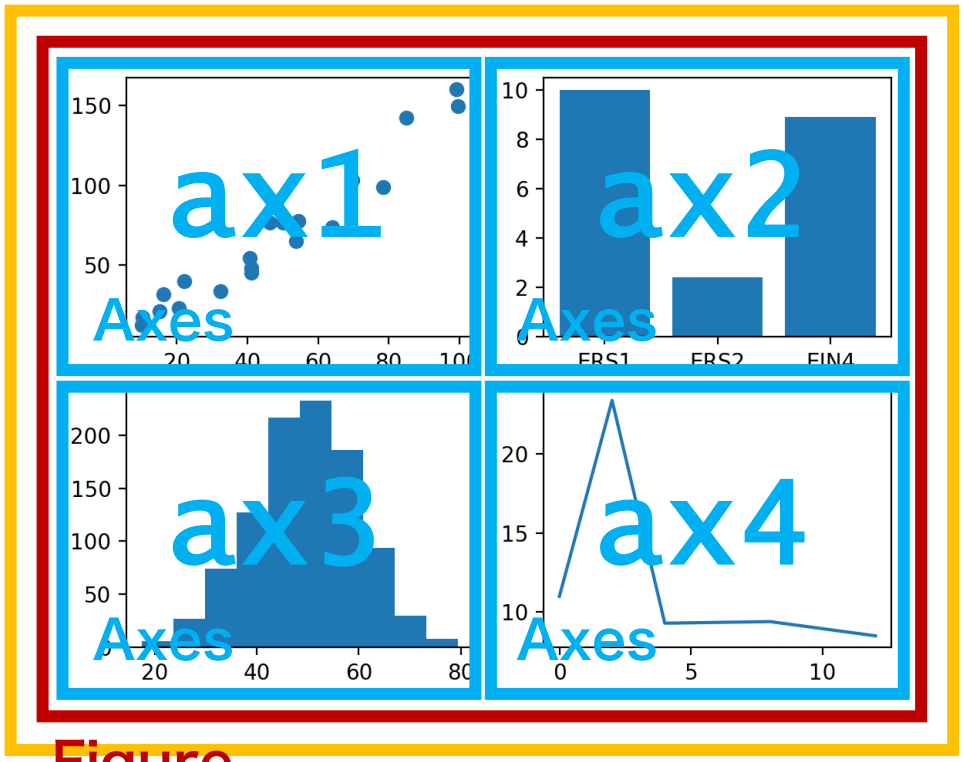

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
fig = plt.figure()
x1 = np.random.uniform(0, 100, 20)
y1 = x1 * np.random.uniform(1, 2, 20)

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



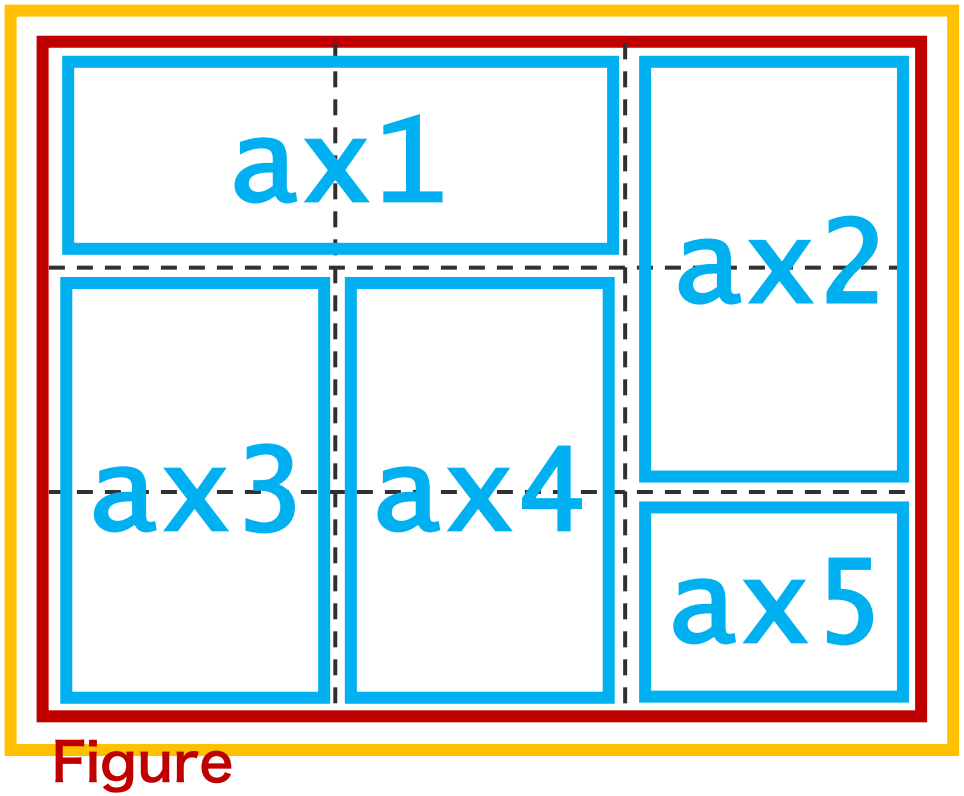

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
fig = plt.figure()
x1 = np.random.uniform(0, 100, 20)
y1 = x1 * np.random.uniform(1, 2, 20)

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)

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.