Python for bioinformatics

どんぐり研究所　孫 建強

コンテンツ

Introduction

1. プログラミング言語概要
2. Python 環境構築

Basic

3. データ型
4. 基本文法

Packages

5. テキスト処理
6. 数値計算 NumPy
7. データ処理 Pandas
8. データ可視化 matplotlib
9. バイオインフォマティクス

Advanced

10. オブジェクト指向

データ処理 Pandas

- シリーズ
- データフレーム
- 表データ処理

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

pandas

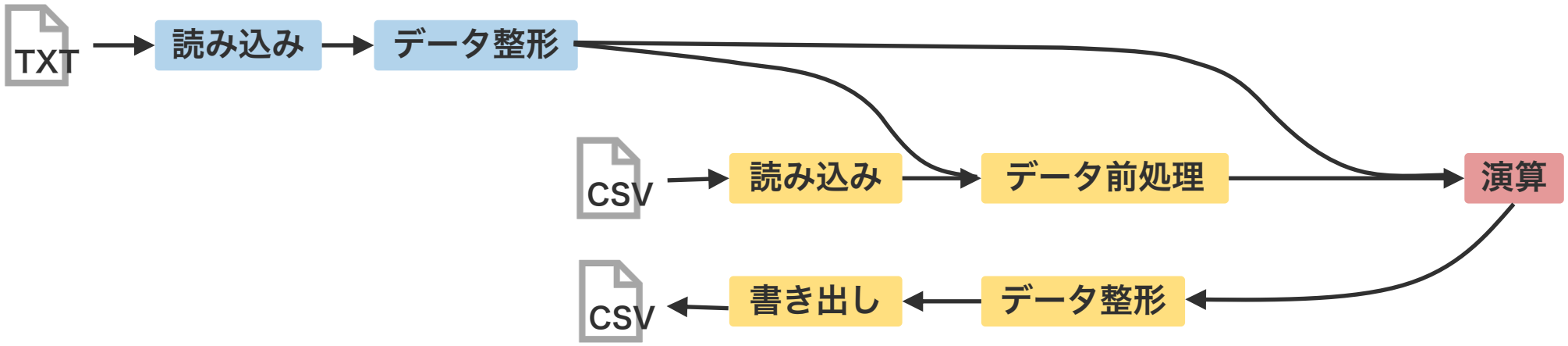
CSV ファイルの読み書きや表データの操作や整形などに特化した機能を提供する。

- シリーズ
- データフレーム

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

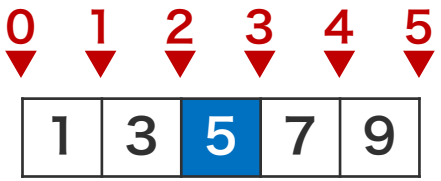


データ処理

- シリーズ
- データフレーム
- 表データ処理

シリーズ

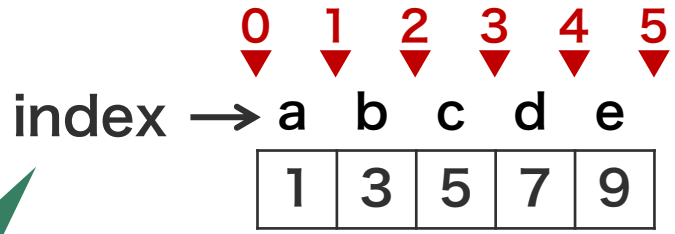
Pandas のシリーズは、1 次元の配列データを扱うときに使用する。pd.Series 関数にリストを代入して作成する。シリーズから要素を取り出すときは、位置番号を指定して取り出す。




```
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。

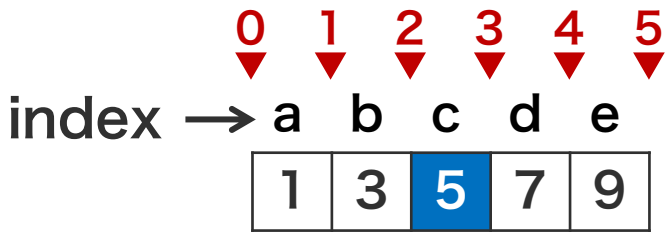


シリーズの各要素に位置番号の他に index と呼ばれる索引が付けられる。

```
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。

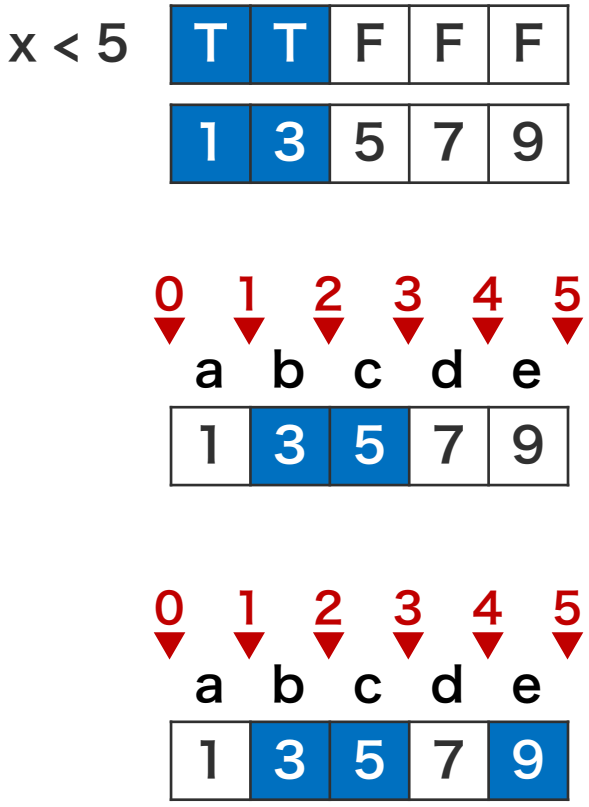
```
import pandas as pd

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

5 min

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性があることに注意。

```
import numpy as np
import pandas as pd

x = pd.Series([1, 3, np.nan, 7, np.nan])
x
# a     3.0
# b     7.0
# c     NaN
# d     NaN
# e    15.0
# 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
```

欠損値 / notnull

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。

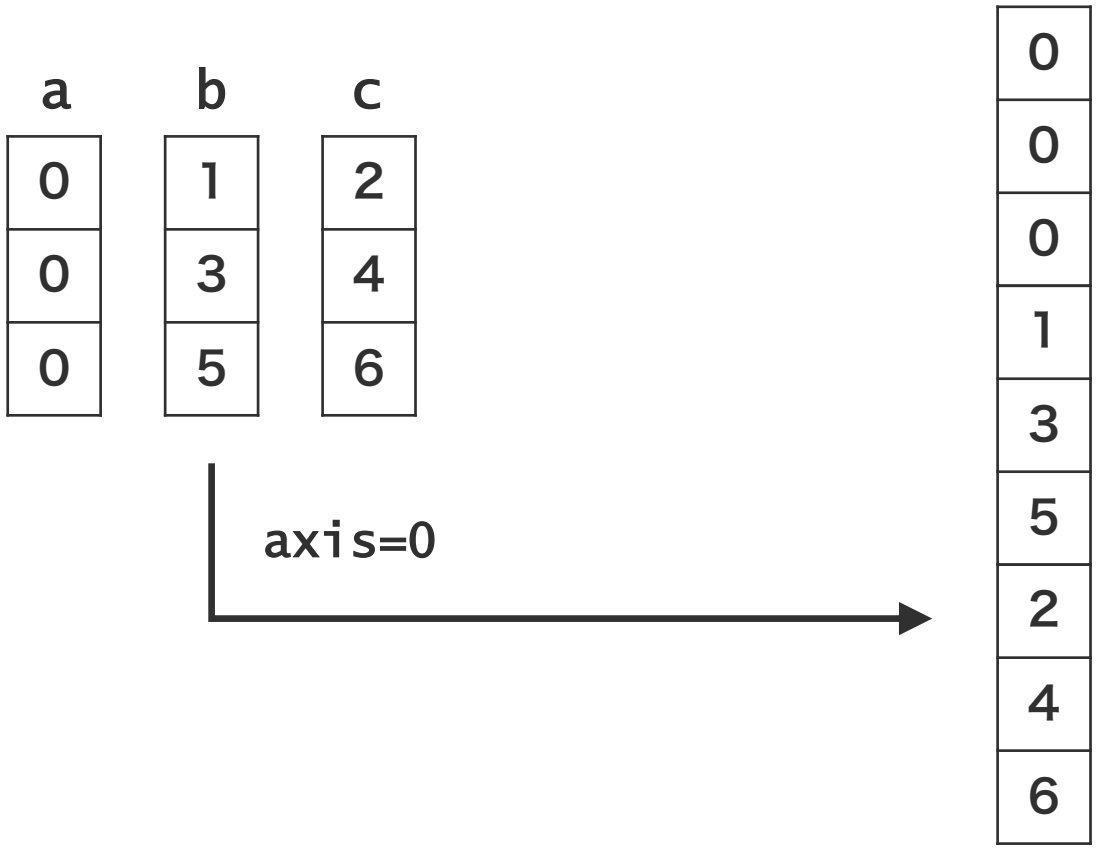
```
import numpy as np
import pandas as pd

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


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

データ処理

- 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

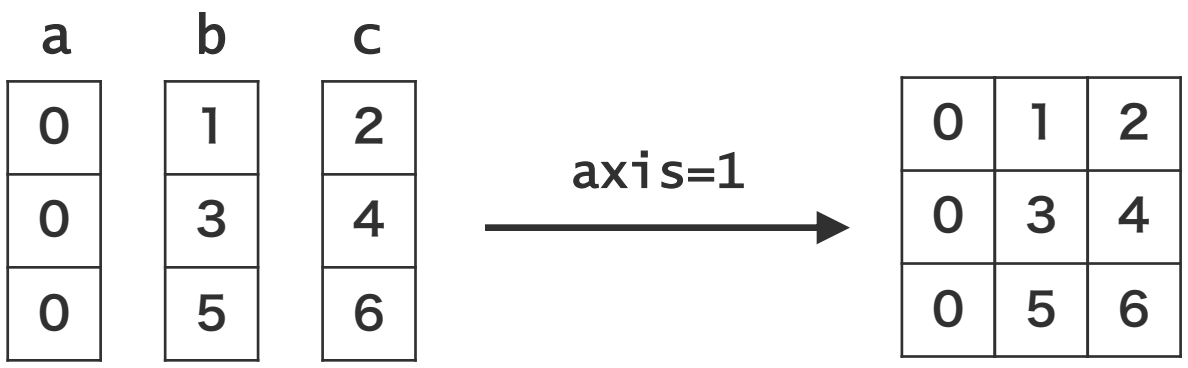
Pandas では、行列型のデータをデータフレームと呼ぶ。データフレームは、シリーズを行方向あるいは列方向に束ねることで作成される。シリーズ同士を束ねるとき pd.concat 関数を使用する。




```
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
# e   NaN    9.0   8.0
# f   1.0    7.0   NaN
```

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

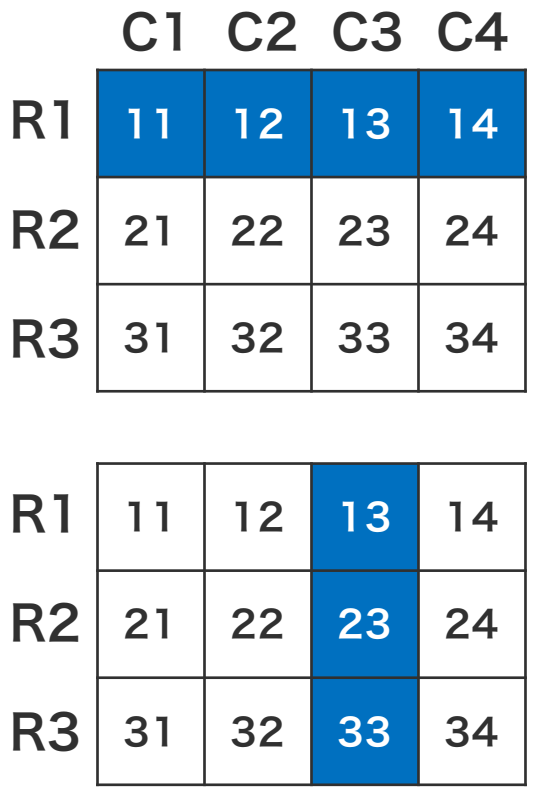
```
import pandas as pd

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。




```
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

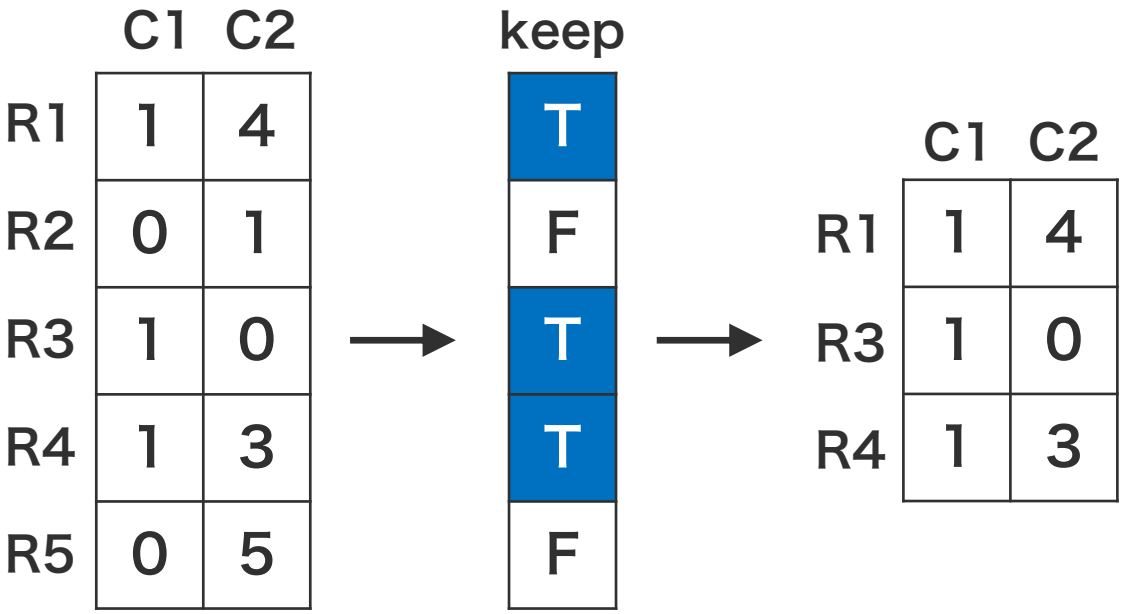
```
import pandas as pd

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
```

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

```
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

axis=0

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

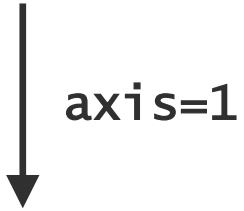
```
import pandas as pd

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
```

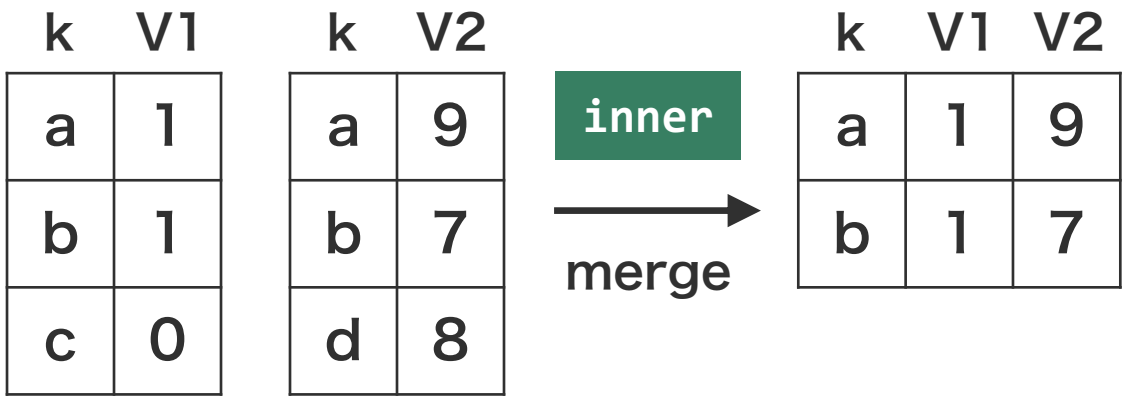
データフレーム / pd.concat

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。




```
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。

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

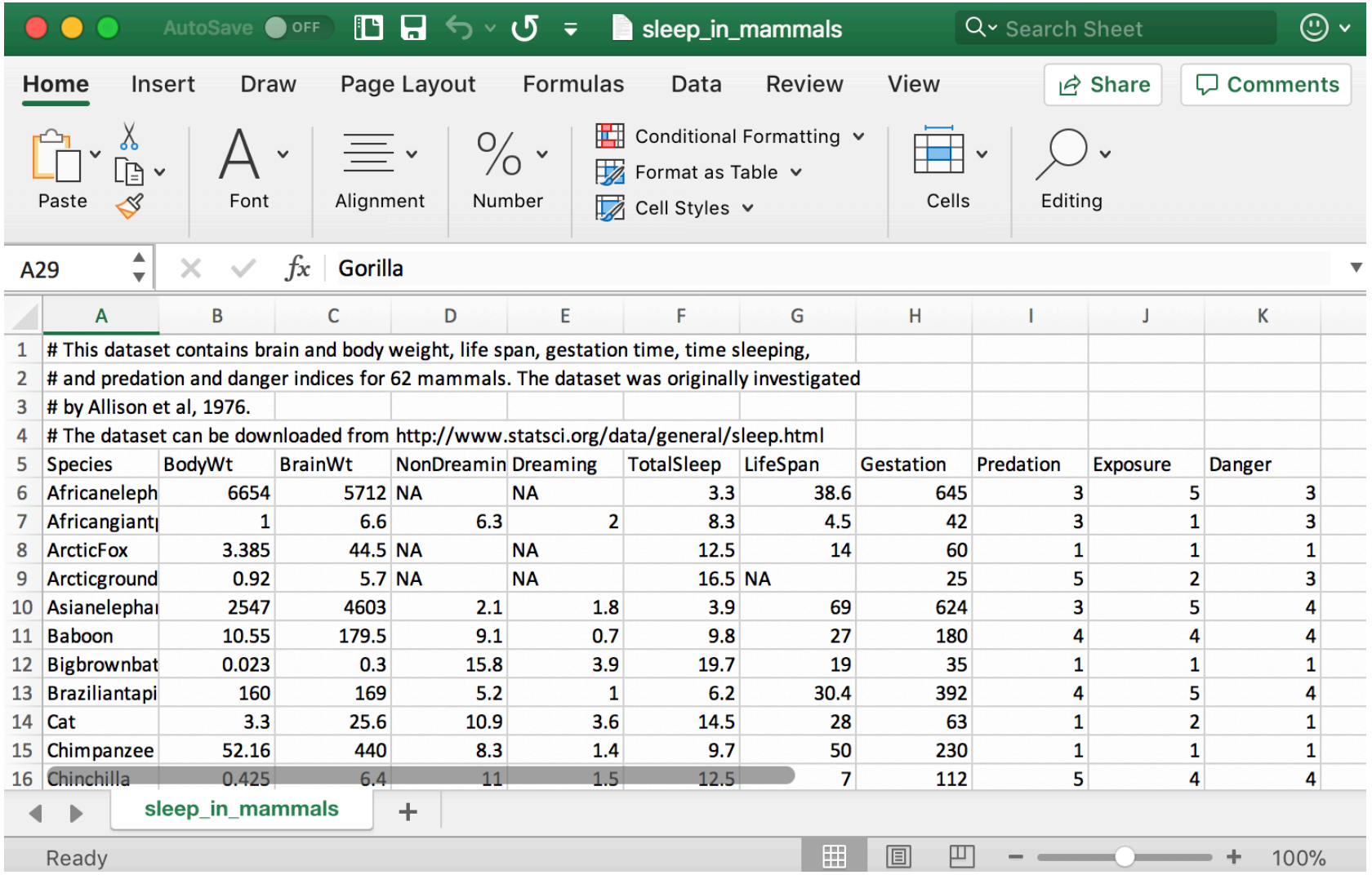
```
import pandas as pd

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
```

データ処理

- 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, time sleeping,
and predation and danger indices for 62 mammals. The dataset was originally investigated
by Allison et al, 1976.
The dataset can be downloaded from http://www.statsci.org/data/general/sleep.html

Species	BodyWt	BrainWt	NonDreamin	Dreaming	TotalSleep	LifeSpan	Gestation
Africaneleph	6654	5712	NA	NA	3.3	38.6	645
Africangiant	1	6.6	6.3	2	8.3	4.5	42
ArcticFox	3.385	44.5	NA	NA	12.5	14	60
Arcticground	0.92	5.7	NA	NA	16.5	NA	25
Asianelepha	2547	4603	2.1	1.8	3.9	69	624
Baboon	10.55	179.5	9.1	0.7	9.8	27	180
Bigbrownbat	0.023	0.3	15.8	3.9	19.7	19	35
Braziliantapi	160	169	5.2	1	6.2	30.4	392
Cat	3.3	25.6	10.9	3.6	14.5	28	63

表データ

属性（特徴量）

ヘッダー →

サンプル

Species	BodyWt	BrainWt	NonDreamin	Dreaming	TotalSleep	LifeSpan	Gestation
Africaneleph	6654	5712	NA	NA	3.3	38.6	645
Africangiant	1	6.6	6.3	2	8.3	4.5	42
ArcticFox	3.385	44.5	NA	NA	12.5	14	60
Arcticground	0.92	5.7	NA	NA	16.5	NA	25
Asianelepha	2547	4603	2.1	1.8	3.9	69	624
Baboon	10.55	179.5	9.1	0.7	9.8	27	180
Bigbrownbat	0.023	0.3	15.8	3.9	19.7	19	35
Braziliantapi	160	169	5.2	1	6.2	30.4	392
Cat	3.3	25.6	10.9	3.6	14.5	28	63

欠損値

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)
```

```
pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.read()

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._read_low_memory()

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._read_rows()

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._tokenize_rows()

pandas/_libs/parsers.pyx in pandas._libs.parsers.raise_parser_error()

ParserError: Error tokenizing data. C error: Expected 1 fields in line 5, saw 11
```

sleep_in_mammals.txt ファイルは、read_table 関数が想定しているフォーマットに従っていないため、エラーになった。

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

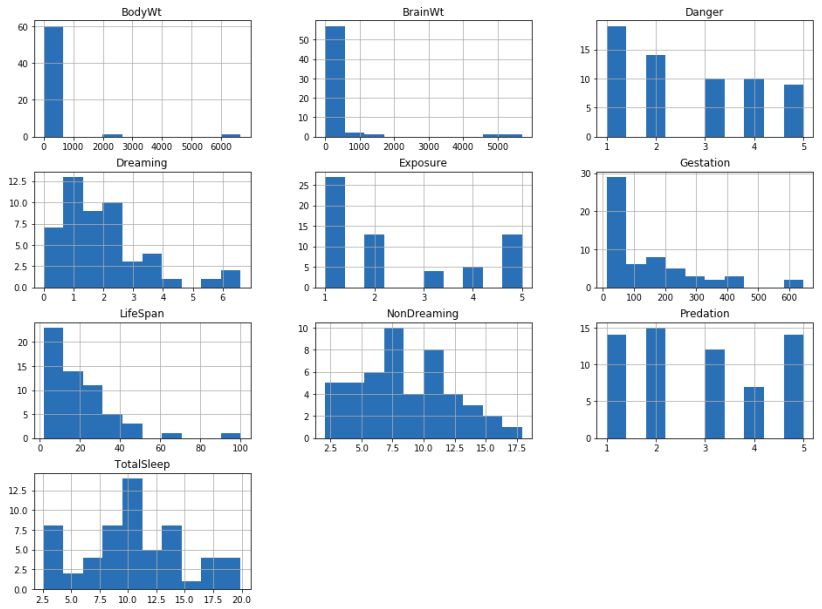
表データ読み込み / データ要約

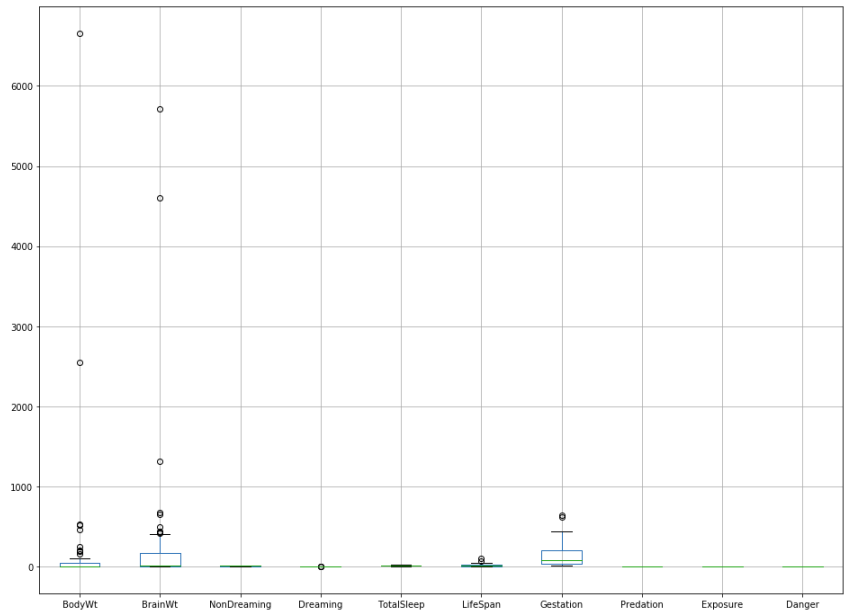
```
import pandas as pd

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/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/data/sleep_in_mammals.txt

問題 P2-1

10 min

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)
```

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

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。

問題 P2-2

10 min

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/data/diversity_galapagos.txt

問題 P2-3

15 min

trees.txt を Pandas で読み込み、高さ（Height）80インチ以上の木の外周長（Girth）の平均値と分散、高さ80インチ未満の木の外周長の平均値と分散を求めよ。なお、平均値と分散を求める関数はそれぞれ mean および var である。

```
import pandas as pd

f = 'trees.txt'
```

https://aabbdd.jp/data/trees.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

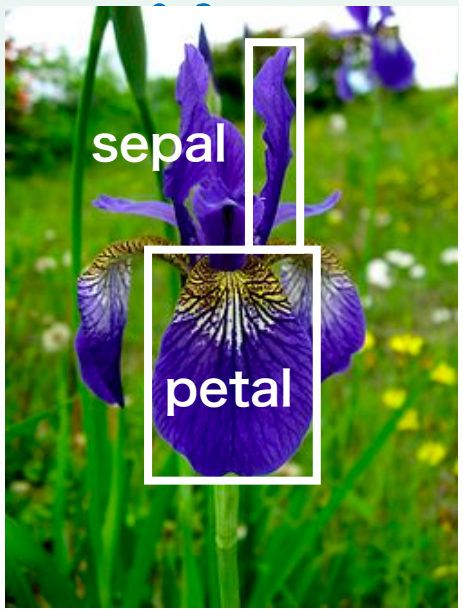
表データ処理

- 表データの読み書き
- 集合演算

グループ演算

Pandas のデータフレームでは、ある列の値に基づいて、全データをいくつかのサブセット分け、それぞれのサブセットに対して平均や分散などを計算する機能が提供されている。

```
import pandas as pd
f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')

d.head()
#    ID  Sepal.Length  Sepal.Width  Petal.Length  Petal.Width Species
# 0   1           5.1          3.5           1.4          0.2  setosa
# 1   2           4.9          3.0           1.4          0.2  setosa
# 2   3           4.7          3.2           1.3          0.2  setosa
# 3   4           4.6          3.1           1.5
# 4   5           5.0          3.6           1.4
```

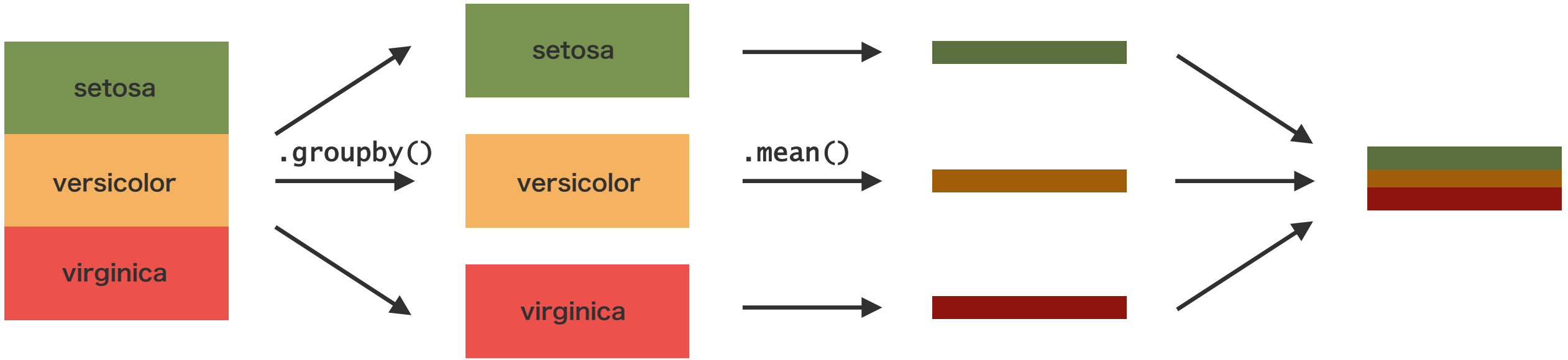


iris データセットは、3種のアヤメ(setosa・versicolor・virginica)に対して、萼 sepal の長さ length と幅 width、花弁 petal の長さ length と幅 width を測定したデータである。各種には 50 個体のデータ含まれ、3種で計150個体のデータが含まれている。

https://aabbdd.jp/data/iris.txt

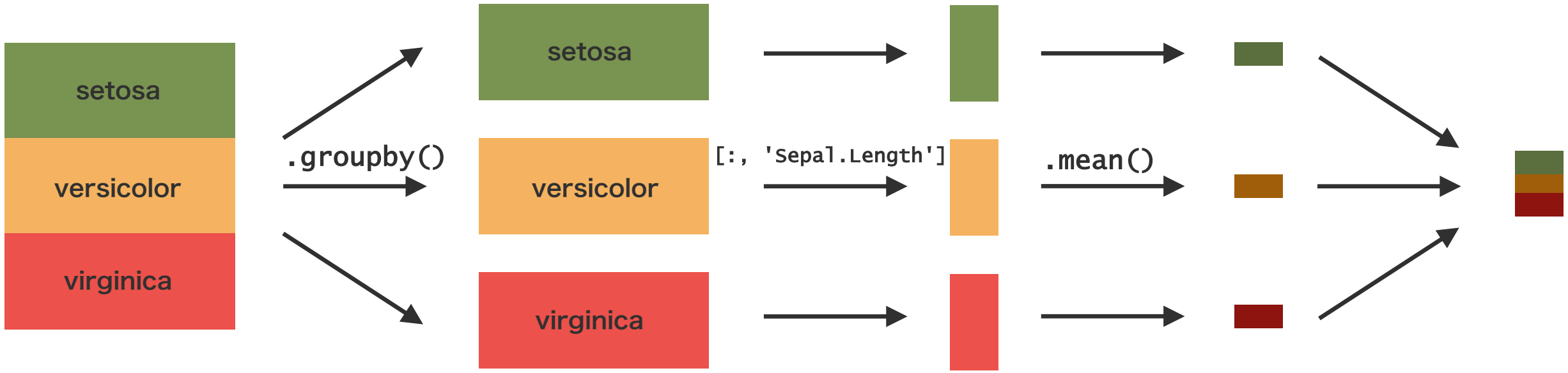
グループ演算 / groupby

```
import pandas as pd
f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')

d.groupby('Species').mean()
#                ID  Sepal.Length  Sepal.Width  Petal.Length  Petal.Width
# Species
# setosa       25.5         5.006        3.428         1.462        0.246
# versicolor   75.5         5.936        2.770         4.260        1.326
# virginica   125.5         6.588        2.974         5.552        2.026
```



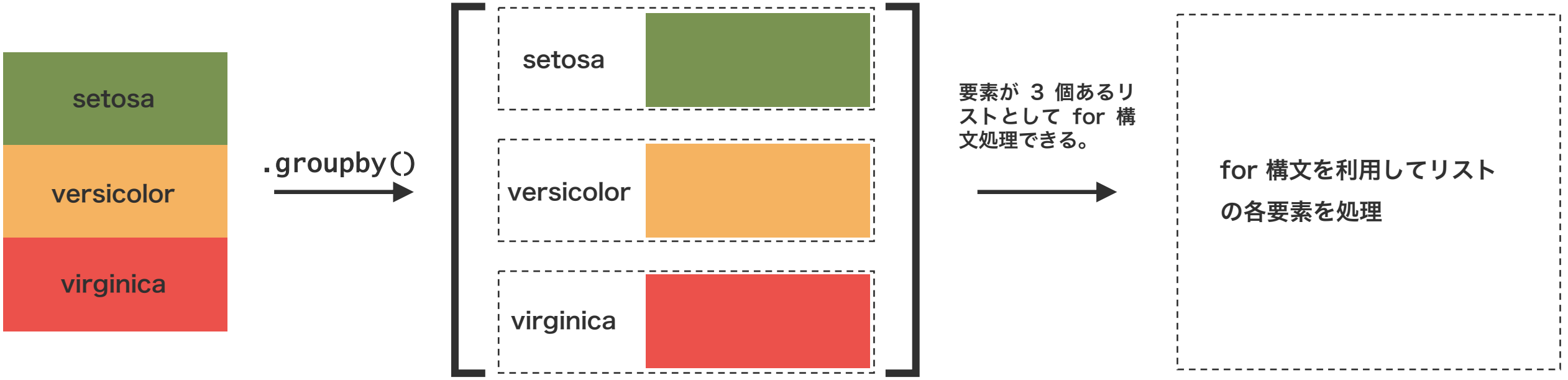
グループ演算 / groupby

```
import pandas as pd
f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')

d.groupby('Species').loc[:, 'Sepal.Length'].mean()
# Species
# setosa         5.006
# versicolor     5.936
# virginica      6.588
# Name: Sepal.Length, dtype: float64
```



グループ演算 / groupby

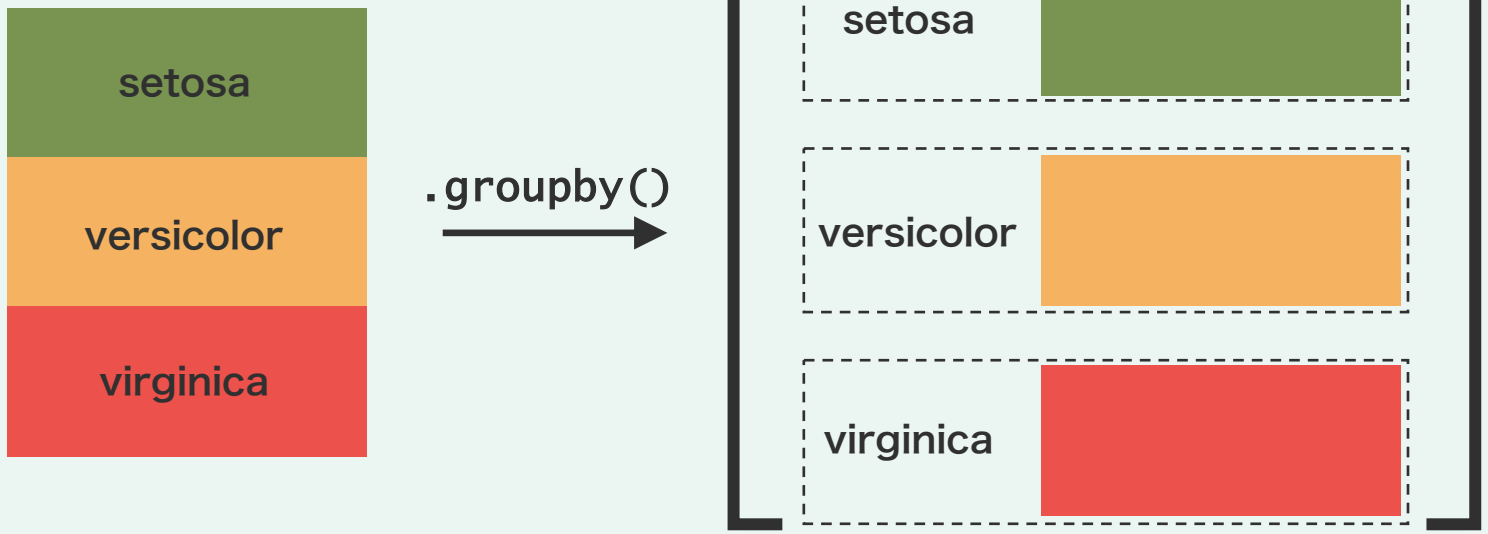
```
import pandas as pd
f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')

for gname, subset in d.groupby('Species'):
    print(gname)
    print(subset.head())
```



グループ演算 / groupby

```
import pandas as pd
f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')

for gname, subset in d.groupby('Species'):
    M = subset.iloc[:, 1:5].max(axis=0)
    m = subset.iloc[:, 1:5].min(axis=0)
    print(gname)
    print(M - m)

# setosa
# Sepal.Length 1.5
# Sepal.Width 2.1
# Petal.Length 0.9
# Petal.Width 0.5
# versicolor
# Sepal.Length 2.1
# Sepal.Width 1.4
# ...
# ...
```

2 列目から 6 列目の部分列を抽出し、各列の最大値を計算する。



グループ演算 / apply

```
import pandas as pd
f = 'iris.txt'
d = pd.read_csv(f, header=0, sep='\t')


def calc_diff(df):
    M = df.iloc[:, 1:5].max(axis=0)
    m = df.iloc[:, 1:5].min(axis=0)
    return M - m


d.groupby('Species').apply(calc_diff)
#             Sepal.Length  Sepal.Width  Petal.Length  Petal.Width
# Species
# setosa               1.5          2.1           0.9          0.5
# versicolor           2.1          1.4           2.1          0.8
# virginica            3.0          1.6           2.4          1.1
```

for 構文を利用したグループ演算を関数化することで、高速な apply 関数を利用できるようにある。

問題 P3-1

10 min

iris.txt を読み込み、各種の花弁 petal の長さ length と幅 width の比率の平均値を求めよ。

```
import pandas as pd

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

https://aabbdd.jp/data/iris.txt

問題 P3-2

15 min

rice.txt には、野生型イネ wt と実験処理後のイネ ANU842 を 3 つの環境（F10, NH4Cl, NH4NO3）で栽培し、その収穫後のバイオマスの乾燥重量（shoot_dry_mass, root_dry_mass）を測定したデータが記録されている。F10 栽培環境で栽培したイネ wt の shoot_dry_mass と root_dry_mass の平均値をそれぞれ求めよ。

```
import pandas as pd

f = 'rice.txt'
d = pd.read_csv(f, header=0, sep='\t')
d.head()
#    replicate  block  root_dry_mass  shoot_dry_mass  trt fert variety
# 0          1      1             56             132  F10  F10      wt
# 1          2      1             66             120  F10  F10      wt
# 2          3      1             40             108  F10  F10      wt
# 3          4      1             43             134  F10  F10      wt
# 4          5      1             55             119  F10  F10      wt
```

https://aabbdd.jp/data/rice.txt

問題 P3-3

15 min

rice.txt ファイルを読み込み、F10, NH4Cl, NH4NO3 栽培環境で栽培したイネ wt の shoot_dry_mass と root_dry_mass の平均値をすべて求めよ。

```
import pandas as pd

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

https://aabbdd.jp/data/rice.txt

問題 P3-4

15 min

rice.txt を読み込み、イネ wt およびイネ AUN842 を F10, NH4Cl, NH4NO3 の環境で栽培した時の shoot_dry_mass と root_dry_mass の平均値をすべて求めよ。なお、必要に応じて groupby('fert', 'variety') を使ってよい。

```
import pandas as pd

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

https://aabbdd.jp/data/rice.txt