

# Dataframeの基礎

# 階層型のデータ

データの保存・整理に向けたデータ形式

[都道府県別訪日外国人]

```
[{'year': 2019, 'prefecture': 'Saitama',  
  'result': {'changes':
```

```
    [{'countryCode': '103',  
      'countryName': '大韓民国',  
      'data': {'total': 9884,  
                'sightseeing': 6800,  
                'business': 3084}}],
```

```
    {'countryCode': '105',  
      'countryName': '中華人民共和国',  
      'data': {'total': 11508,  
                'sightseeing': 7508,  
                'business': 4000}}],
```

```
    {'countryCode': '106',  
      'countryName': '台湾',  
      'data': {'total': 9456,  
                'sightseeing': 4456,  
                'business': 5000}}]
```

# 表形式のデータ

データの分析に向けたデータ形式

階層型のままでは、集計・演算等がやりにくい

[都道府県別・国籍別年間訪問者数]

列(column)毎にデータの要素の  
意味を表す見出しがついている

|   | year | prefecture | country | total |
|---|------|------------|---------|-------|
| 1 | 2019 | Saitama    | 韓国      | 9884  |
| 2 | 2019 | Saitama    | 中国      | 11508 |
| 3 | 2019 | Saitama    | 台湾      | 9456  |
| 4 | 2019 | Tokyo      | 韓国      | 30388 |
|   |      |            |         |       |

行毎にデータのid (ラベル)  
がついている

目的に応じてデータを選択  
的に取り出すのに便利

# 世界は階層型データで表現されている！

## 1. Webサイトのデータはすべて階層型

パスタ レシピ (903)

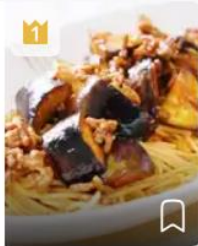
### 極ウマ♡ナスとひき肉のボロネーゼ風パスタ

パスタ・豚ひき肉・ナス・にんにく・ケチャップ・ウスターソース・顆粒コンソメ・塩・ブラックペッパー・パルメザンチーズ

🍳 1人分



ないく



### 超シンプル！トマトと大葉で和風冷製パスタ

パスタ・大葉・トマト・めんつゆ・EVオリーブオイル・すり胡麻・ニンニク・黒胡椒・ツナ、クリチ、アボカド等好きな食材を足して下さい カリカリに炒ったジャコや海苔のトッピングもめんつ...

🍳 2人分



putimiko



### ☆ツナと玉ねぎのにんにく醤油パスタ☆

パスタ・ツナ・にんにく・玉ねぎ・サラダ油・水・しょうゆ・顆粒和風だし・塩こしょう・かつおぶし・万能ねぎ

🍳 2人前



☆栄養士のれしび☆



### 超シンプル！トマトと大葉で和風冷製パスタ



putimiko @putimiko

Yahoo掲載 & 3000レボ大感謝♡食べたいと思ったら10分以内に完成♪  
材料も超シンプル♡簡単過ぎてごめんなさい

もっと読む

### 材料

🍳 2人分

パスタ（細めのもの・好みで） 200g

大葉 10枚～

トマト（大） 1個（230gでした）

### 作り方

- 1 たっぶりの湯を沸し、レシピ分量外の塩を入れ、パスタを表示より1分くらい長めに茹でる
- 2 その間トマトを食べ易い大きさに切り、大葉も細かくきざむ  
◆と一緒にボールに入れザッと混ぜる





## パスタ

ひき肉のボロネーゼ風

トマトと大葉の冷製パスタ

ツナと玉ねぎのニンニク  
醤油パスタ

パスタ 80g

豚ひき肉（合挽きでも） 50g

ナス 1本

にんにく 1片

ケチャップ（市販） 大さじ2

パスタ（細めのもの・好みで） 200g

大葉 10枚〜

トマト（大） 1個（230gでした）

◆めんつゆ(2倍濃縮) 大さじ3

◆EVオリ- ツナ 1缶(70g)

◆すり胡麻 パスタ 180g

◆ニンニク にんにく(みじん切り) 1かけ

玉ねぎ 1/2個

サラダ油 大さじ1

●水 100cc

●しょうゆ 大さじ1と1/2

レシピ相互の素材構成を比較するには表形式データにした方が便利

|                  | パスタ | 豚ひき肉 | ナス | にんにく | 大葉 | トマト | 玉ねぎ |
|------------------|-----|------|----|------|----|-----|-----|
| ひき肉のボロネーゼ風       | 80  | 50   | 1  | 1    | 0  | 0   | 0   |
| トマトと大葉の冷製パスタ     | 200 | 0    | 0  | 1    | 10 | 230 | 0   |
| ツナと玉ねぎのニンニク醤油パスタ | 180 | 0    | 0  | 1    | 0  | 0   | 0.5 |

# データサイエンスでのビッグデータの形式

1. 保管・整理には階層型データ形式
2. 演算・分析には表形式データ

## 階層形式データ

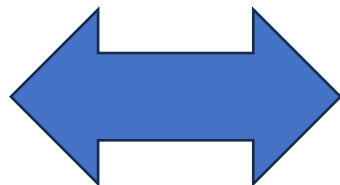
情報の保存・整理



Pythonによる階層  
形式データ

Json

相互に変換



## 表形式データ

データの演算・分析



Pythonによる表  
形式データ

DataFrame

## 2. 文書は階層型

### 第1章 序論

- 1.1 研究の背景 . . . . .
- 1.2 研究の目的 . . . . .
- 1.3 本論文の構成 . . . . .

### 第2章 トポロジカル光と軌道角運動量

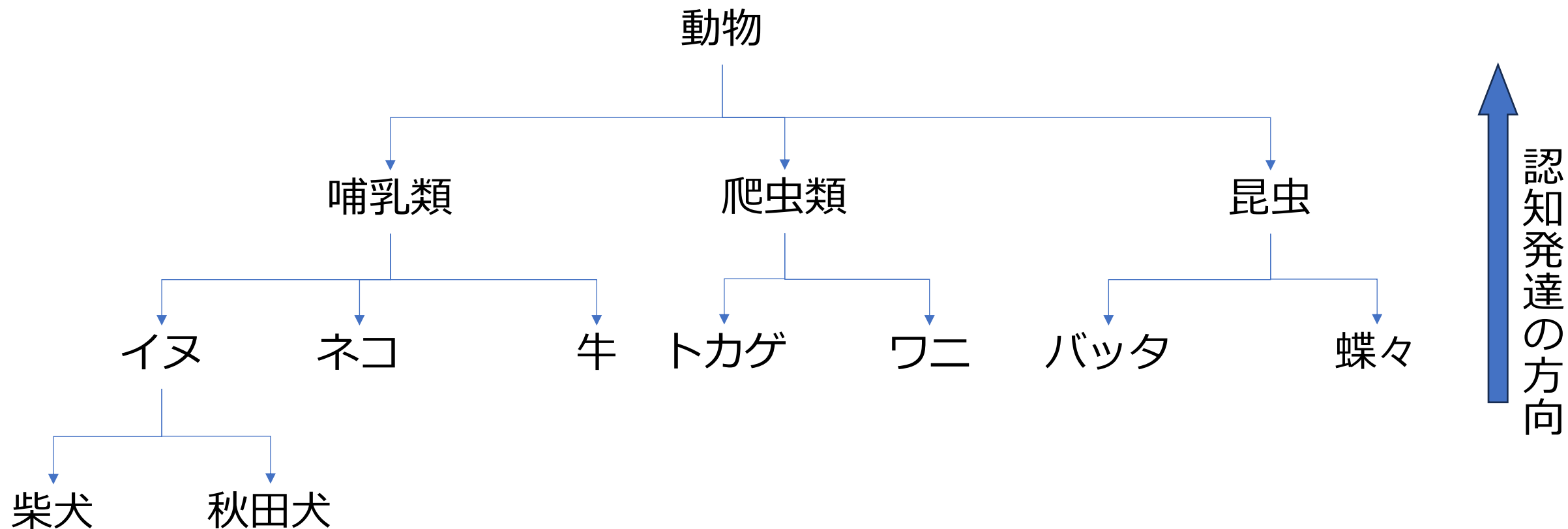
- 2.1 光渦 . . . . .
- 2.2 ドーナツ . . . . .
- 2.3 コーヒーカップ . . . . .

### 第3章 美味しいチャーハンの作り方

- 3.1 これまでのおいしいチャーハンの作り方の問題点 . . . . .
- 3.2 美味しいチャーハン作製法 . . . . .
  - 3.2.1 火力増強法 . . . . .
  - 3.2.2 中華鍋法 . . . . .
- 3.3 新手法 . . . . .
- 3.4 実験結果 . . . . .
- 3.5 考察 . . . . .
  - 3.5.1 焦げる原因 . . . . .
  - 3.5.2 全体的に美味しくない原因 . . . . .
- 3.6 今後の展望 . . . . .

### 第4章 結論

### 3. 人の知識は階層型



人は情報を整理するとき、階層型にする



# DataFrame (データフレーム)

numpy2次元配列データに列ラベル : column, 行ラベル : index が付いたpythonの表データ形式

## Column (カラム)

|    | name   | height | weight | blood type | nation  | age | occupation |
|----|--------|--------|--------|------------|---------|-----|------------|
| U1 | Lucy   | 160    | 52     | A          | USA     | 20  | student    |
| E1 | Bob    | 180    | 52     | O          | England | 30  | dentist    |
| J1 | Shohei | 193    | 80     | AB         | Japan   | 30  | MLB        |
| J2 | Mami   | 180    | 62     | B          | Japan   | 28  | wife       |

## Index (インデックス)

データの値 : Numpy 2次元配列

# DataFrameを作成する構文

Pandasをimport

```
import pandas as pd
```

```
profile_df = pd.DataFrame(profiles, index=['U1', 'E1', 'J1', 'J2'],  
                           columns = ['name', 'height', 'weight',  
                                     'blood type', 'nation', 'age', 'occupation'])
```

DataFrameを作成

pd.DataFrame( 2次元配列, index=行ラベル (リスト型) , columns=列ラベル (リスト型) )

# 列・行の連結

## 列方向の連結

(行数が一致する必要がある)

pd.concat([df1,df2], axis=1)

df1

|    | name   | height | weight | blood type | nation  | age | occupation |
|----|--------|--------|--------|------------|---------|-----|------------|
| U1 | Lucy   | 160    | 52     | A          | USA     | 20  | student    |
| E1 | Bob    | 180    | 52     | O          | England | 30  | dentist    |
| J1 | Shohei | 193    | 80     | AB         | Japan   | 30  | MLB        |
| J2 | Mami   | 180    | 62     | B          | Japan   | 28  | wife       |

df2

|    | hobby    | married |
|----|----------|---------|
| U1 | tennis   | married |
| E1 | game     | single  |
| J1 | baseball | married |
| J2 | baseball | married |



## 行方向の連結

(列数が一致する必要がある)

pd.concat([df1,df3], axis=0)

df3

|    | name  | height | weight | blood type | nation  | age | occupation    |
|----|-------|--------|--------|------------|---------|-----|---------------|
| FR | Frank | 190    | 80     | O          | Germany | 15  | student       |
| NA | Naomi | 185    | 70     | A          | Japan   | 30  | tennis player |



# スライシングの例

1) 行ラベル・列ラベルを指定したスライシング

```
.loc[J1:FR, blood type:occupation]
```

2) 行インデックス・列インデックスを指定したスライシング

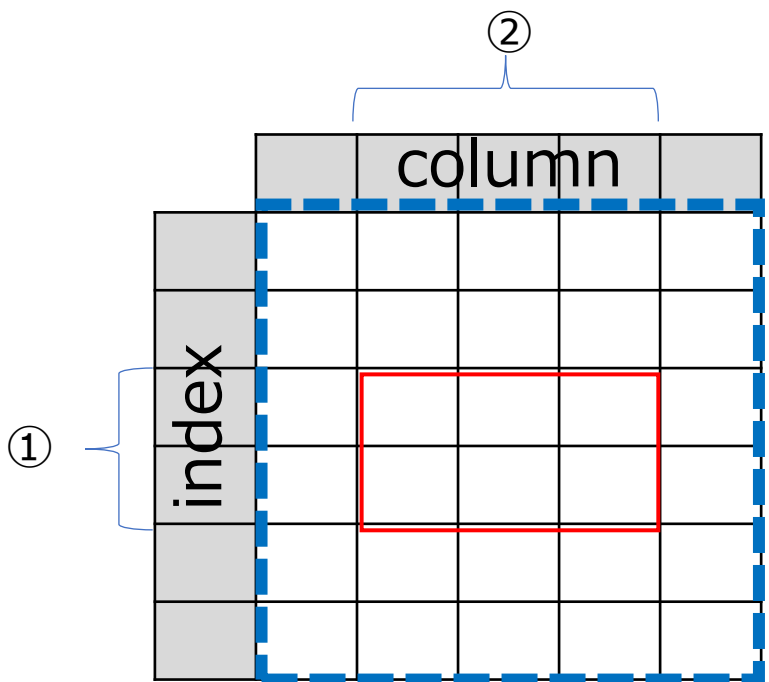
```
.iloc[2:5, 3:7]
```

|           | name   | height | weight | blood type | nation  | age | occupation    | male/female | hobby    | married |
|-----------|--------|--------|--------|------------|---------|-----|---------------|-------------|----------|---------|
| <b>U1</b> | Lucy   | 160    | 52     | A          | USA     | 20  | student       | female      | tennis   | married |
| <b>E1</b> | Bob    | 180    | 52     | O          | England | 30  | dentist       | male        | game     | single  |
| <b>J1</b> | Shohei | 193    | 80     | AB         | Japan   | 30  | MLB           | male        | baseball | married |
| <b>J2</b> | Mami   | 180    | 62     | B          | Japan   | 28  | wife          | female      | baseball | married |
| <b>FR</b> | Frank  | 190    | 80     | O          | Germany | 15  | student       | male        | ski      | single  |
| <b>NA</b> | Naomi  | 185    | 70     | A          | Japan   | 30  | tennis player | female      | game     | married |

## 2種類のスライシング

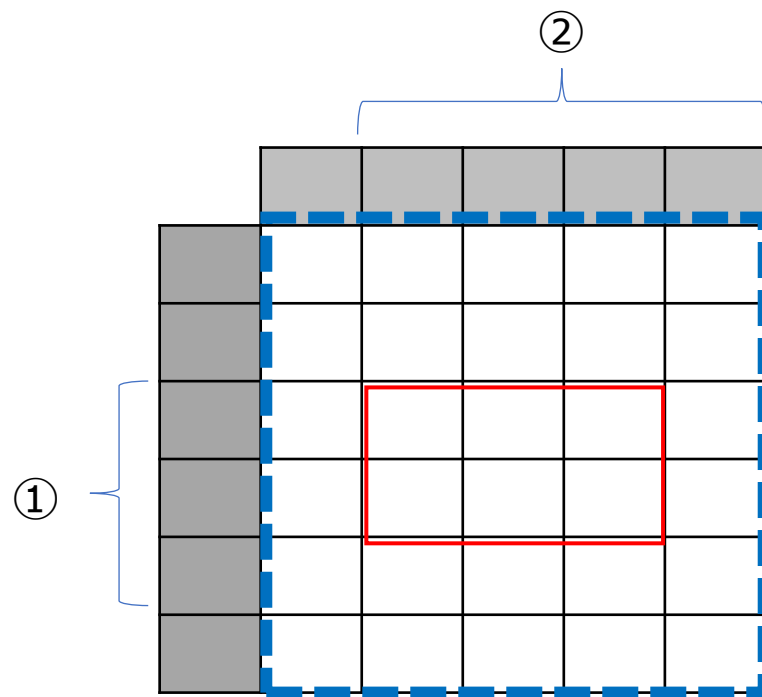
`.loc[ from : to , from : to ]`

行ラベル① 列ラベル②






`df.iloc[ from : to , from : to ]`

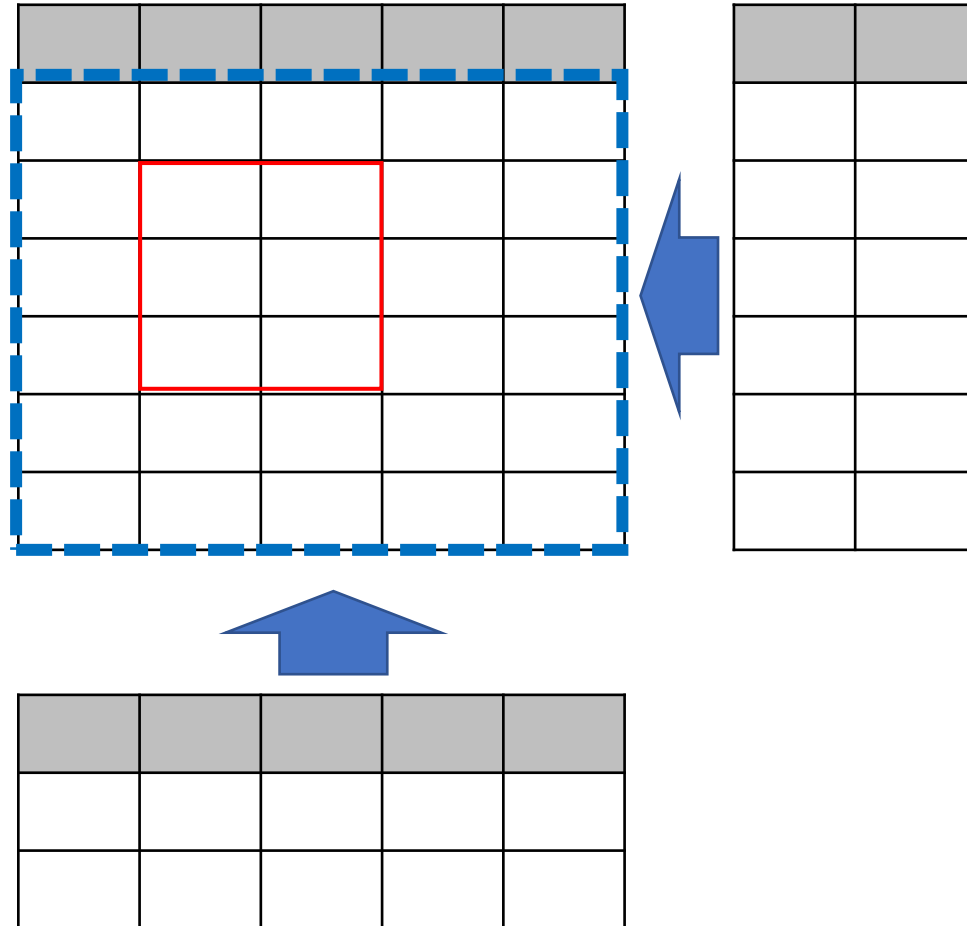
行index① 列index②

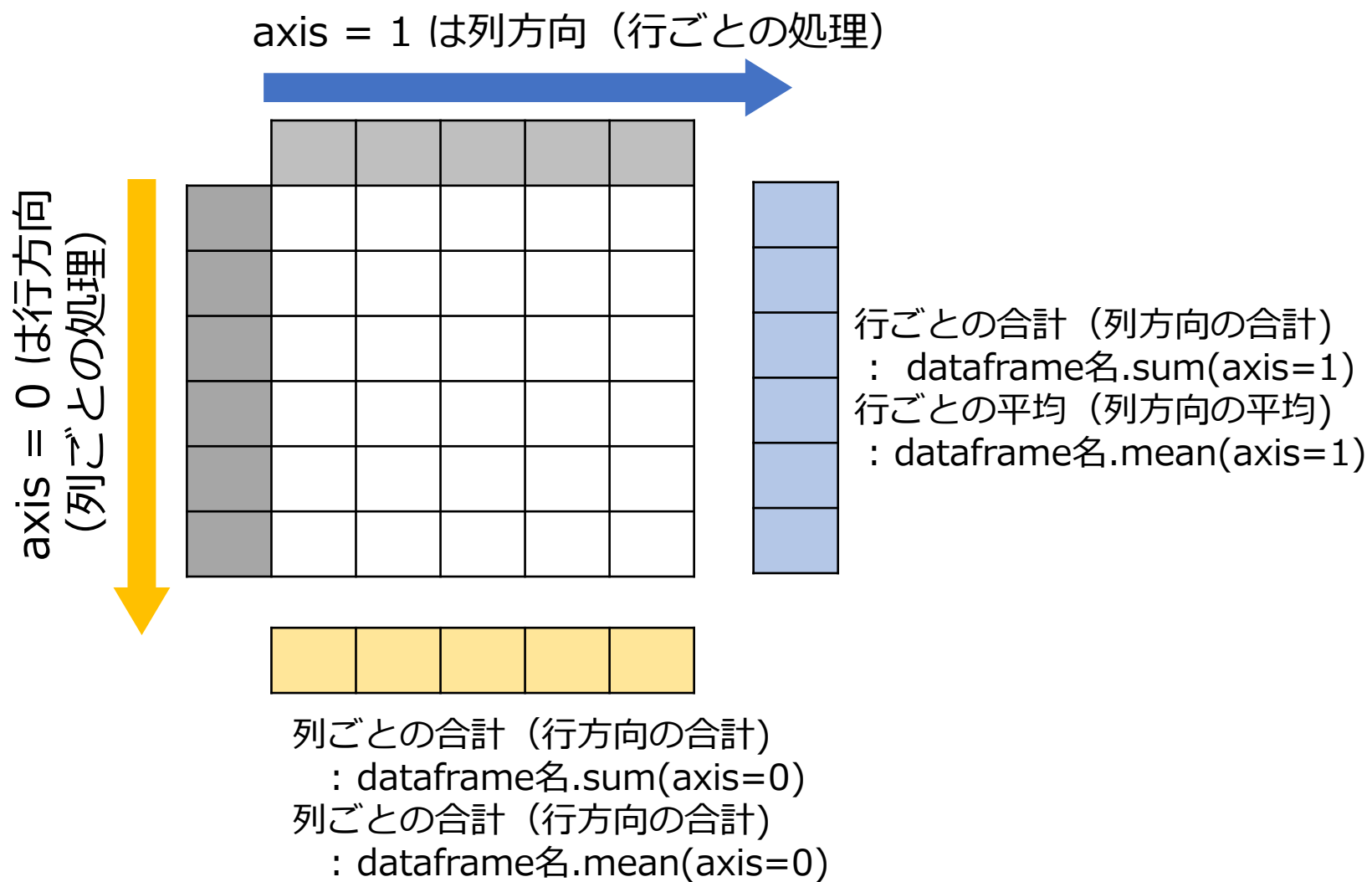


indexの to は取り出したい列+1

# Pandasを用いたDataFrameの重要な処理

1. スライシング 
2. 条件抽出
3. データセット作成 
4. 結合 
5. csvファイルとの読書き





# 演習

1. 以下のループの各回で計算される **b, a** を書き  
のような**DataFrame**にして表示せよ

```
for a in range(100):  
    b = a**a
```

b, a を列見出しとする

|   | b   | a   |
|---|-----|-----|
| 0 | 0   | 0   |
| 1 | 1   | 1   |
| 2 | 2   | 4   |
| 3 | 3   | 9   |
| 4 | 4   | 16  |
| 5 | 5   | 25  |
| 6 | 6   | 36  |
|   | ... | ... |

2. 以下のdataframe を作成せよ

Indexは省略する

| ID  | City     | Birth Year | Name    |
|-----|----------|------------|---------|
| 100 | tokyo    | 1990       | Hiroshi |
| 101 | osaka    | 1989       | Akiko   |
| 102 | kyoto    | 1992       | Yuki    |
| 103 | hokkaido | 1997       | Satoru  |
| 104 | tokyo    | 1982       | Steve   |



実は、DataFrameはcolumns, indexを取り除くとnumpy配列

```
>>> df
```

|   | ID  | City  | Birth Year | Name    |
|---|-----|-------|------------|---------|
| 0 | 100 | tokyo | 1990       | Hiroshi |
| 1 | 101 | osaka | 1989       | Akiko   |
| 2 | 102 | Kyoto | 1997       | Satoru  |
| 3 | 104 | tokyo | 1982       | Steve   |

赤枠内はnumpy `val = df.values` ってやると取り出せる

構文： 変数名 = DataFrame名.values

numpyをlist型に変換したい場合は

構文： 変数名 = DataFrame名.values.tolist()

## 演習 .ilocによるスライシング

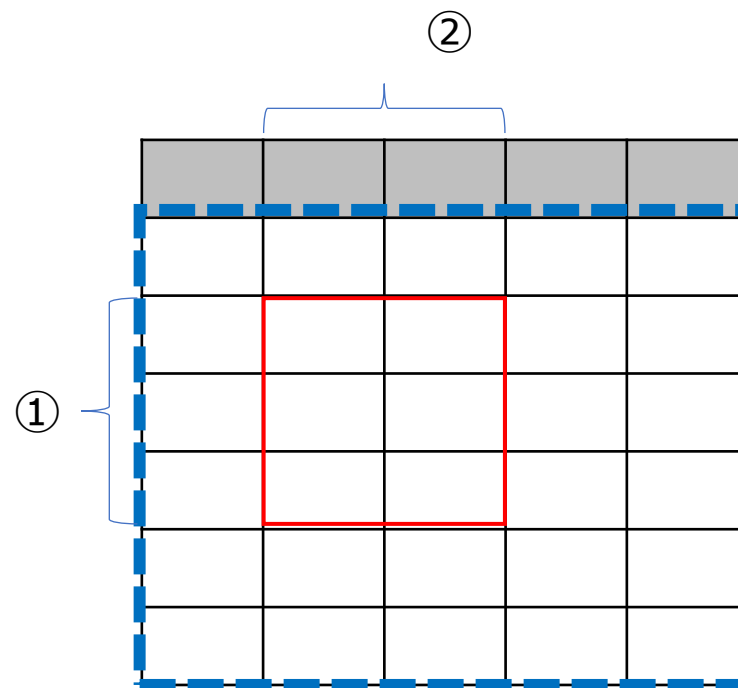
1. ilocを使って2~4行目かつ City Birth Yearの要素を取り出せ
2. スライドの構文でDataFrameのデータ部分をnumpy配列として取り出し、
  1. 同じ要素を取りだせ。

参考

<https://python.atelierkobato.com/slicing/>

# クロス辞書として縦横キーにより同時スライシング(.loc) (DataFrameを縦横辞書型として扱う)

Index、columnsを辞書のキーに使う。 toは取り出したい行（列）の最後のindex



# 演習 .locによるスライシング

まず以下のデータを、IDをDataFrameのindexにして新規でDataFrameとして作成しなおしてください。

| ID  | City     | Birth Year | Name    |
|-----|----------|------------|---------|
| 100 | tokyo    | 1990       | Hiroshi |
| 101 | osaka    | 1989       | Akiko   |
| 102 | kyoto    | 1992       | Yuki    |
| 103 | hokkaido | 1997       | Satoru  |
| 104 | tokyo    | 1982       | Steve   |

## 演習続き

### 1. 列見出しによるスライシング

1) 以下のdataframeからNameだけを取りだせ

2) locを使って2~4行目かつ City, Birth Yearの要素を取り出せ

|     | City     | Birth Year | Name    |
|-----|----------|------------|---------|
| 100 | tokyo    | 1990       | Hiroshi |
| 101 | osaka    | 1989       | Akiko   |
| 102 | kyoto    | 1992       | Yuki    |
| 103 | hokkaido | 1997       | Satoru  |
| 104 | tokyo    | 1982       | Steve   |

各行をfor文で取り出すには？？

|     | City     | Birth Year | Name    |
|-----|----------|------------|---------|
| 100 | tokyo    | 1990       | Hiroshi |
| 101 | osaka    | 1989       | Akiko   |
| 102 | kyoto    | 1992       | Yuki    |
| 103 | hokkaido | 1997       | Satoru  |
| 104 | tokyo    | 1982       | Steve   |



```
for k, row in df.iterrows():  
    for i, item in row.items():  
        print(i.item)
```

.iterrows() を使ってループすると、dataframeの行を  
ごっそり取り出せる（kはインデックス）

## 演習

前頁のdataframeを1行ずつ取り出して表示せよ

ヒント <https://note.nkmk.me/python-pandas-dataframe-for-iteration/>

# 条件抽出

Birth Year > 1990 のデータを抽出する

やってみてください。



# DataFrameの連結

df

|   | ID   | City  | Birth Year | Name     |
|---|------|-------|------------|----------|
| 0 | [100 | tokyo | 1990       | Hiroshi] |
| 1 | [101 | osaka | 1989       | Akiko]   |
| 2 | [102 | Kyoto | 1997       | Satoru]  |
| 3 | [104 | tokyo | 1982       | Steve]]  |

df2

| Age   |
|-------|
| [[32] |
| [33]  |
| [25]  |
| [50]] |



axis=1

df3

|   | ID   | City    | Birth Year | Name     |
|---|------|---------|------------|----------|
| 4 | [110 | saitama | 2000       | Sakiko]] |

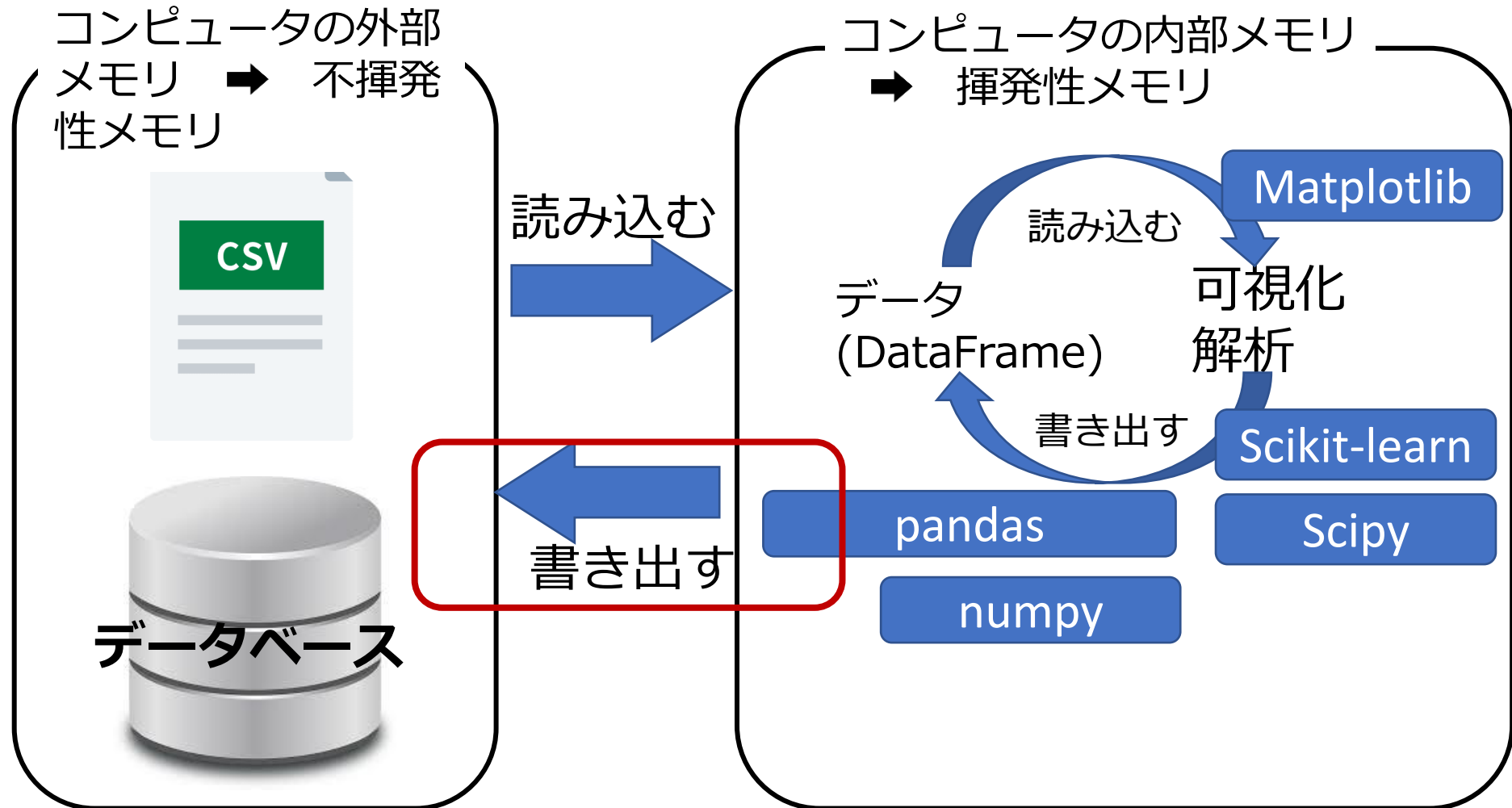


axis=0

```
col_df = pd.concat([df,df2],axis=1)
```

```
row_df = pd.concat([df,df3],axis=0)
```

# Dataframeをcsvに書き出す



# Dataframeはキーが縦横についた辞書構造

```
for i,t in titanic.iterrows():  
    for k,v in t.items():
```

|    | survived | pclass | sex    | age  | sibsp | parch | fare     |
|----|----------|--------|--------|------|-------|-------|----------|
| 1  | 1        | 1      | female | 38.0 | 1     | 0     | 71.2833  |
| 3  | 1        | 1      | female | 15.0 | 1     | 0     | 53.1000  |
| 6  | 0        | 1      | male   | 54.0 | 0     | 0     | 51.8625  |
| 27 | 0        | 1      | male   | 19.0 | 3     | 2     | 263.0000 |
| 31 | 1        | 1      | female | NaN  | 1     | 0     | 146.5208 |

## 2重構造の辞書

- titanic(dataframe)は、インデックスをkey, 1行をvalueとする辞書型
- tは、1行の値に見出しがついたseries辞書型（見出しがkey : 値がvalue）

{ i : t }

1回目

|   | survived | pclass | sex    | age  | sibsp | parch | fare    |
|---|----------|--------|--------|------|-------|-------|---------|
| 1 | 1        | 1      | female | 38.0 | 1     | 0     | 71.2833 |

{k:v}

{ i : t }

2回目

|   | survived | pclass | sex    | age  | sibsp | parch | fare    |
|---|----------|--------|--------|------|-------|-------|---------|
| 3 | 1        | 1      | female | 15.0 | 1     | 0     | 53.1000 |

{k:v}