# Структури даних

Однією з найбільш зручних структур даних є масиви ndarray бібліотеки NumPy.
Створити масив ndarray, а по суті, перетворити будь-який масивоподібний об’єкт в масив, можна за допомогою функції

**`np.array()`**

Масиви мають наступні атрибути: **`ndim`** (розмірність), **`shape`** (розмір кожного виміру), **`size`** (загальний розмір масиву), **`dtype`** (тип елементів), **`nbytes`** (розмір масиву в байтах), **`itemsize`** (розмір елементу в байтах)

# Структури даних

Звернення до елементів масиву відбувається аналогічно до звичайних списків мови Python. Але виділені підмасиви ndarray є представлення даних масиву, а не їх копією. Тобто при зміні даних підмасиву відбувається зміна даних і в самому масиві, що зручно при роботі з великими об’ємами даних.

Для створення копії підмасиву можна скористатись методом **`copy()`**

Можна перетворити одновимірний масив на багатовимірний за допомогою методу reshape і провести зворотне перетворення методом ravel:

```
a=np.arange(0,10).reshape(5,2)
```

```
a.ravel()
```

# Структури даних

Існує три методи об’єднання масивів: додавання, конкатенація та укладання.

1.Додавання включає приєднання одного масиву в кінець іншого масиву, відбувається за допомогою функції np.append. Якщо не вказувати вісь, то результатом буде одновимірний масив.

```
x=np.array([[1,2],[3,4]])
y=np.array([[6,7,8],[9,10,11]])
np.append(x,y)
```

```
array([ 1, 2, 3, 4, 6, 7, 8, 9, 10, 11])
```

```
np.append(x,y,axis=1)
```

```
array([[ 1,  2,  6,  7,  8],
       [ 3,  4,  9, 10, 11]])
```

# Структури даних

2.Конкатенація передбачає об’єднання масивів уздовж осі (вертикальної або горизонтальної). Виконується функцією np.concatenate.

```
a=np.array([0,1,2])
b=np.array([3,4,5])
c=np.array([6,7,8])
np.concatenate([a,b,c])
```

array([0, 1, 2, 3, 4, 5, 6, 7, 8])

```
x=np.array([[1,2],[3,4]])
y=np.array([[6,7],[9,10]])
np.concatenate((x,y))
```

array([[ 1, 2],
[ 3, 4],
[ 6, 7],
[ 9, 10]])

# Структури даних

3. Укладання: воно може бути двох типів: вертикальне або горизонтальне. Вертикальне укладання складає масиви один під одним. Кількість елементів у кожному підмасиві масивів має бути однаковою. Використовується функція np.vstack

```
a=np.array([0,1,2])
b=np.array([[3,4,5],[6,7,8]])
np.vstack([a,b])
```

```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

# Структури даних

Горизонтальне укладання «вкладає» масиви поруч. Кількість підмасивів має бути однаковою для кожного з масивів, які розміщені горизонтально. Використовується функція np.hstack

```
a=np.array([[0],[1]])
b=np.array([[3,4,5],[6,7,8]])
np.hstack([b,a])
```

```
array([[3, 4, 5, 0],
       [6, 7, 8, 1]])
```

# Структури даних

Аналогічно масиви можна роз’єднувати наступними функціями, що приймають індекси, які задають точки роз’єднання:

**`np.split`**

```
a,b,c=np.split(d,[2,6])
d,a,b,c
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8]),
 array([0, 1]),
 array([2, 3, 4, 5]),
 array([6, 7, 8]))
```

# Структури даних

Вертикальне роз’єднання **np.vsplit**

```
a, b = np.vsplit(g,[1])
g,a,b
```

```
(array([[3, 4, 5, 0],
        [6, 7, 8, 1]]),
 array([[3, 4, 5, 0]]),
 array([[6, 7, 8, 1]]))
```

Горизонтальне роз’єднання **np.hsplit**

```
a,b = np.hsplit(g,[2])
g,a,b
```

```
(array([[3, 4, 5, 0],
        [6, 7, 8, 1]]),
 array([[3, 4],
        [6, 7]]),
 array([[5, 0],
        [8, 1]]))
```

# Структури даних

Над масивами, а саме над кожним елементом масива, можливо здійснювати звичайні арифметичні операції: додавання (+, **`np.add`**), віднімання (-, **`np.subtract`**), множення (*, **`np.multiply`**), ділення (/, **`np.divide`**), піднесення до ступеню (**, **`np.power`**), ділення за модулем (%, **`np.mod`**), зміна знаку на протилежний (-, **`np.negative`**), модуль (**`np.abs`**).

# Структури даних

При роботі з об’ємними даними виникає потреба задати масив, в якому буде збережений результат обчислення. Зробити це можна за допомогою аргументу **`out`**:

```
a=np.arange(10)
b=np.empty(10)
np.divide(a,2,out=b)
b
```

```
array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])
```

# Структури даних

Суму елементів масиву можна отримати, використавши функцію **`np.sum`**(), добуток елементів **`np.prod`**(), максимальне та мінімальне значення **`np.max`**() та **`np.min`**() відповідно, **`np.argmax`**() та **`np.argmin`**() – їх індекси. Всі ці функції дозволяють обирати вісь, за якою виконується операція

```
g,g.max(axis=0),g.max(axis=1),g.sum(axis=1)
(array([[3, 4, 5, 0],
        [6, 7, 8, 1]]),
 array([6, 7, 8, 1]),
 array([5, 8]),
 array([12, 22]))
```

# Структури даних

У бібліотеці NumPy також реалізовані оператори порівняння, такі як <(«менше») і> («більше») у вигляді поелементних універсальних функцій. Результат цих операторів порівняння завжди являє собою масив з булевим типом даних. Доступні для використання все шість стандартних операторів порівняння:

| | | |
|---|---|---|
| == | або | **np.equal** |
| != | або | **np.not_equal** |
| < | або | **np.less** |
| <= | або | **np.less_equal** |
| > | або | **np.greater** |
| >= | або | **np.greater_equal** |

# Структури даних

```
a=np.random.randint(10,size=(10))
a, a<5
```

(array([0, 5, 2, 4, 7, 7, 4, 3, 4, 5]),

array([ True, False,  True,  True, False, False,  True,  True,  True, False]))

```
a>=3
```

array([False,  True, False,  True,  True,  True,  True,  True,  True,True])

# Структури даних

```
b=np.random.randint(10,size=(10))
b,a==b
```

(array([5, 8, 3, 1, 7, 1, 4, 3, 6, 2]),
 array([False, False, False, False,  True, False,  True,  True, False, False]))

```
a=np.random.randint(10,size=(2,3))
a,a!=5
```

(array([[3, 7, 5],
        [4, 7, 5]]),
 array([[ True,  True, False],
        [ True,  True, False]]))

# Структури даних

Оскільки результатами порівняння є булеві масиви, то цікаві функції для роботи з ними.
Скільки в масиві елементів зі значенням True?
**`np.count_nonzero()`** або **`np.sum()`**

```
a,np.count_nonzero(a>4),np.sum(a>3,axis=1)
```

```
(array([[3, 7, 5],
        [4, 7, 5]]),
 4,
 array([2, 3]))
```

# Структури даних

Чи є в масиві хоч одне значення True або False?

**`np.any()`** та **`np.all()`**

```
a,np.any(a<3),np.all(a!=6)
```

```
(array([[3, 7, 5],
        [4, 7, 5]]),
 False,
 True)
```

# Структури даних

Можна використовувати і побітові логічні операції

```
&    np.bitwise_and
|    np.bitwise_or
^    np.bitwise_xor
~    np.bitwise_not
```

```
a,(a>3)&(a<9)
```

```
(array([[3, 2, 9, 7, 9],
        [4, 5, 4, 0, 4]]),
 array([[False, False, False,  True, False],
        [ True,  True,  True, False,  True]]))
```

```
a,np.sum((a<3)|(a>5))
```

```
(array([[3, 2, 9, 7, 9],
        [4, 5, 4, 0, 4]]),
 5)
```

# Структури даних

Булеві масиви можна використовувати в якості масок для вибору потрібних підмножин самих даних.

```
a
```

```
array([[3, 2, 9, 7, 9], [4, 5, 4, 0, 4]])
```

```
a>5
```

```
array([[False, False,  True,  True,  True],
       [False, False, False, False, False]])
```

Щоб вибрати потрібні значення з масиву, досить просто проіндексувати вихідний масив a з цього булеву масиву. Така дія носить назву операції накладення маски або маскування:

```
a[a>5]
```

```
array([9, 7, 9])
```

# Структури даних

Для сортування масивів існує функція **`np.sort()`**. За замовчанням вона використовує метод швидкого сортування, що можна змінити за допомогою параметра **`kind`**.

```
b=np.random.randint(100,size=(10))
b,np.sort(b)
```

(array([27, 99, 74,  7, 48,  7, 12, 35, 64, 28]),
 array([ 7,  7, 12, 27, 28, 35, 48, 64, 74, 99]))

```
b.sort()
b
```

array([ 7,  7, 12, 27, 28, 35, 48, 64, 74, 99])

# Структури даних

Функція **`argsort`** повертає індекси відсортованих елементів.

```
b=np.random.randint(100,size=(10))
b,np.sort(b),np.argsort(b)
```

```
(array([51,  9, 75, 47, 19, 93, 57, 42, 94, 73]),
 array([ 9, 19, 42, 47, 51, 57, 73, 75, 93, 94]),
 array([1, 4, 7, 3, 0, 6, 9, 2, 5, 8], dtype=int64))
```

Перший елемент цього результату відповідає індексу мінімального елемента, другий - індексу другого за величиною і т. д.

# Структури даних

За допомогою аргументу axis можна сортувати двовимірні масиви за рядками та стовпцями.

```
a=np.random.randint(100,size=(3,6))
a
```

```
array([[65, 70, 67,  3, 47, 33],
       [25, 27, 76, 14,  5, 41],
       [31, 76, 56, 82, 13, 70]])
```

```
np.sort(a,axis=0)
```

```
array([[25, 27, 56,  3,  5, 33],
       [31, 70, 67, 14, 13, 41],
       [65, 76, 76, 82, 47, 70]])
```

```
np.sort(a,axis=1)
```

```
array([[ 3, 33, 47, 65, 67, 70],
       [ 5, 14, 25, 27, 41, 76],
       [13, 31, 56, 70, 76, 82]])
```

# Структури даних

Іноді не потрібно сортувати весь масив, а просто потрібно знайти K найменших значень в ньому. Бібліотека NumPy надає для цієї мети функцію **`np.partition`**. Функція **`np.partition`** приймає на вході масив і число K. Результат являє собою новий масив з K найменшими значеннями зліва від точки розбиття і інші значення праворуч від неї в довільному порядку:

```
b,np.partition(b,4)
```

```
(array([51,  9, 75, 47, 19, 93, 57, 42, 94, 73]),
 array([ 9, 19, 42, 47, 51, 93, 57, 75, 94, 73]))
```

```
np.argpartition(b,4)
```

```
array([1, 4, 7, 3, 0, 5, 6, 2, 8, 9], dtype=int64)
```

# Структури даних

Можна отримувати доступ до елементів масиву не лише за окремими індексами, а й за масивами індексів (тобто доступ до декількох елементів масиву одночасно).

```
a=np.random.randint(50,size=(10))
a
```

array([ 4, 23, 41, 25, 18, 23, 15, 19, 18, 23]

```
a[[0,3,8]]
```

array([ 4, 25, 18])

# Структури даних

Бібліотека NumPy має також і структуровані масиви і масиви записів для зберігання неоднорідних даних.

Для створення структурованого масиву потрібно вказати типи даних

```
a = np.zeros(97, dtype={'names':('sex', 'body', 'heart'),
'formats':('U1', 'f4', 'f4')})
```

‘b’ байт
‘i’ ціле число
‘u’ ціле без знаку
‘f’ число з плаваючою крапкою
‘c’ комплексне число
'S', ‘a’ рядок
‘U’ рядок Unicode
‘V’ void

# Структури даних

Далі можна заповнити порожній масив значеннями

```
data = pd.read_csv('catsM.csv')
a['sex'] = data['Sex']
a['body'] = data['Bwt']
a['heart'] = data['Hwt']
print(a)
```

```
[('M', 2. ,  6.5) ('M', 2. ,  6.5) ('M', 2.1, 10.1) ('M', 2.2,  7.2)
 ('M', 2.2,  7.6) ('M', 2.2,  7.9) ('M', 2.2,  8.5) ('M', 2.2,  9.1)
 ('M', 2.2,  9.6) ('M', 2.2,  9.6) ('M', 2.2, 10.7) ('M', 2.3,  9.6)
 ('M', 2.4,  7.3) ('M', 2.4,  7.9) ('M', 2.4,  7.9) ('M', 2.4,  9.1)
 ('M', 2.4,  9.3) ('M', 2.5,  7.9) ('M', 2.5,  8.6) ('M', 2.5,  8.8)
 ('M', 2.5,  8.8) ('M', 2.5,  9.3) ('M', 2.5, 11. ) ('M', 2.5, 12.7)
 ('M', 2.5, 12.7) ('M', 2.6,  7.7) ('M', 2.6,  8.3) ('M', 2.6,  9.4)
```

# Структури даних

```
a[11]
```

('M', 2.3, 9.6)

```
a[a['body']>3.5]
```

```
array([('M', 3.6, 11.8), ('M', 3.6, 13.3), ('M', 3.6, 14.8),
       ('M', 3.6, 15. ), ('M', 3.7, 11. ), ('M', 3.8, 14.8),
       ('M', 3.8, 16.8), ('M', 3.9, 14.4), ('M', 3.9, 20.5)],
      dtype=[('sex', '<U1'), ('body', '<f4'), ('heart', '<f4')])
```

```
a[(a['body']>2.5)&(a['body']<3.0)]
```

```
array([('M', 2.6,  7.7), ('M', 2.6,  8.3), ('M', 2.6,  9.4),
       ('M', 2.6,  9.4), ('M', 2.6, 10.5), ('M', 2.6, 11.5),
       ('M', 2.7,  8. ), ('M', 2.7,  9. ), ('M', 2.7,  9.6),
       ('M', 2.7,  9.6), ('M', 2.7,  9.8), ('M', 2.7, 10.4),
       ('M', 2.7, 11.1), ('M', 2.7, 12. ), ('M', 2.7, 12.5),
       ('M', 2.8,  9.1), ('M', 2.8, 10. ), ('M', 2.8, 10.2),
       ('M', 2.8, 11.4), ('M', 2.8, 12. ), ('M', 2.8, 13.3),
       ('M', 2.8, 13.5), ('M', 2.9,  9.4), ('M', 2.9, 10.1),
       ('M', 2.9, 10.6), ('M', 2.9, 11.3), ('M', 2.9, 11.8)],
```

# Структури даних

Бібліотека NumPy має клас **`np.recarray`**, практично ідентичний структурованим масивів, але з однією додатковою можливістю: доступ до полів можна здійснювати як до атрибутів, а не тільки як до ключів словника.

```
b = a.view(np.recarray)
b.heart
```

```
array([ 6.5,  6.5, 10.1,  7.2,  7.6,  7.9,  8.5,  9.1,  9.6,  9.6, 10.7,
        9.6,  7.3,  7.9,  7.9,  9.1,  9.3,  7.9,  8.6,  8.8,  8.8,  9.3,
       11. , 12.7, 12.7,  7.7,  8.3,  9.4,  9.4, 10.5, 11.5,  8. ,  9. ,
        9.6,  9.6,  9.8, 10.4, 11.1, 12. , 12.5,  9.1, 10. , 10.2, 11.4,
       12. , 13.3, 13.5,  9.4, 10.1, 10.6, 11.3, 11.8, 10. , 10.4, 10.6,
       11.6, 12.2, 12.4, 12.7, 13.3, 13.8,  9.9, 11.5, 12.1, 12.5, 13. ,
       14.3, 11.6, 11.9, 12.3, 13. , 13.5, 13.6, 11.5, 12. , 14.1, 14.9,
       15.4, 11.2, 12.2, 12.4, 12.8, 14.4, 11.7, 12.9, 15.6, 15.7, 17.2,
       11.8, 13.3, 14.8, 15. , 11. , 14.8, 16.8, 14.4, 20.5],
      dtype=float32)
```

# Структури даних Pandas

Бібліотека Pandas має багато особливотей, які добре підходять для обробки та аналізу даних:

1. Pandas надає засоби для позначення та індексації, що значно пришвидшують доступ до даних.

2. Pandas підтримує читання  та запис популярних форматів файлів, наприклад, JSON , CSV, Excel та інші.

3. Часто дані надходять з декількох джерел і Pandas надає функції для поєднання та комбінування наборів даних.

4.Також Pandas має вбудовані засоби для візуалізації даних на основі Matplotlib.

5. Бібліотека Pandas інтегрується з іншими бібліотеками, такими як Numpy, Matplotlib, Scipy та Scikit-learn.

6. Pandas має функції для групування даних за категоріями, застосування до цих груп окремих операцій та комбінування результату.

7. Pandas також має засоби для обробки відсутніх, дубльованих та помилкових даних.

# Структури даних Pandas

Об'єкт Series бібліотеки Pandas - одновимірний масив індексованих даних. Його можна створити зі списку або масиву наступним чином:

```
a = pd.Series([1,2,3,4])
```

```
0   1
1   2
2   3
3   4
dtype: int64
```

```
pd.Series(2)
```

```
0    2
```

```
pd.Series(list('hello'))
```

```
0    h
1    e
2    l
3    l
4    o
```

```
pd.Series([2]*5)
```

```
0    2
1    2
2    2
3    2
4    2
```

# Структури даних Pandas

pd.Series({1:'AB',2:'CD',3:'EF’})

```
1    AB
2    CD
3    EF
```

pd.Series(np.arange(1,5))

```
0    1
1    2
2    3
3    4
```

pd.Series(np.random.normal(size=4))

```
0     0.470160
1    -1.019613
2     0.066487
3     0.151582
```

pd.Series([2,0,1,6],index=['a','b','c','d'])

```
a    2
b    0
c    1
d    6
```

# Структури даних Pandas

Основна відмінність між об’єктом Series та одновимірним масивом бібліотеки NumPy - індекс. У той час як індекс масиву NumPy, який використовується для доступу до значень, - цілочисельний і описується неявно, індекс об'єкта Series бібліотеки Pandas описується явно і зв'язується зі значеннями. Цей індекс не обов’язково має бути цілим числом, а може складатися з значень будь-якого типу.

```
a = pd.Series([1, 2, 3, 4],index=['I', 'II', 'III', 'IV'])
```

```
a
```

```
I    1
II   2
III  3
IV   4
dtype: int64
```

```
a['IV’]
```

```
4
```

# Структури даних Pandas

Об’єктом Series також має багато спільного зі словниками, тобто наборами пар ключ-значення.

```
a.keys()
```

Index(['I', 'II', 'III', 'IV'], dtype='object’)

```
list(a.items())
```

[('I', 1), ('II', 2), ('III', 3), ('IV', 4)]

```
a['V']=5
a
```

I 1

II 2

III 3

IV 4

V 5

dtype: int64

# Структури даних Pandas

Кількість елементів у об’єкті Series можна знайти трьома способами: за допомогою атрибуту size, функції len або параметру shape. Атрибут size та функція len повертають одне значення, тобто кількість елементів.

```
x=pd.Series(np.arange(1,10))
x.size
```

9

```
len(x)
```

**9**

# Структури даних Pandas

Атрибут shape повертає кортеж з кількістю рядків та стовпців:

```
x.shape
```

(9,)

Атрибут values повертає масив NumPy, що містить значення всіх елементів об’єкта Series.

```
x.values
```

array([1, 2, 3, 4, 5, 6, 7, 8, 9])

# Структури даних Pandas

До індексів об’єкта Series можна отримати доступ через атрибут index. Індекс являє собою об’єкт з типом даних за замовчуванням RangeIndex і множиною значень.

```
x.index
```

RangeIndex(start=0, stop=9, step=1)

# Структури даних Pandas

Метод value_counts() об’єкта Series показує унікальні значення, що містяться в цьому об’єкті, та кількість кожного з цих унікальних значень. Часто цей метод використовують для категоріальних ознак, щоб дізнатись, які значення вони можуть приймати.

```
y=pd.Series(['a','b','a','c','d','b’])
y.value_counts()
```

```
a    2
b    2
d    1
c    1
```

# Структури даних Pandas

До об’єкту Series можна послідовно застосовувати декілька методів. Нижче використовується метод для підрахунку унікальних значень та метод, який показує перші два результати.

```
y=pd.Series(['a','b','a','c','d','b’])
y.value_counts().head(2)
```

```
a    2
b    2
```

# Структури даних Pandas

Наступна базова структура бібліотеки Pandas - об'єкт DataFrame.
Якщо об'єкт Series - аналог одновимірного масиву з гнучкими індексами, об'єкт DataFrame - аналог двовимірного масиву з гнучкими індексами рядків і гнучкими іменами стовпців.

```
a=np.random.randint(50,size=(4))
b=np.random.randint(50,size=(4))
frame = pd.DataFrame({'a': a, 'b': b })
frame
```

| | a | b |
|---|---|---|
| **0** | 36 | 18 |
| **1** | 49 | 42 |
| **2** | 1 | 37 |
| **3** | 36 | 7 |

# Структури даних Pandas

Об’єкт DataFrame містить три складові: об’єкт стовпців column, об’єкт індексів index та масив NumPy, в якому зберігаються значення.

Індекси та стовпці разом називаються осями. Індекси формують вісь 0, а стовпці – вісь 1.

```
odd_numbr=pd.Series([1,3,5])
even_numbr=pd.Series([2,4,6])
df=pd.DataFrame([odd_numbr,even_numbr])
df.columns=['first','second','third']
df
```

| | first | second | third |
|---|---|---|---|
| **0** | 1 | 3 | 5 |
| **1** | 2 | 4 | 6 |

# Структури даних Pandas

```
df=pd.DataFrame(np.arange(1,9).reshape(2,4))
df
```

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| **0** | 1 | 2 | 3 | 4 |
| **1** | 5 | 6 | 7 | 8 |

```
df=pd.DataFrame([(1,3,5),(2,4,6)],columns=['firs
t','second','third'])
df
```

| | first | second | third |
|---|---|---|---|
| **0** | 1 | 3 | 5 |
| **1** | 2 | 4 | 6 |

# Структури даних Pandas

```
data = pd.read_csv('Fish.csv' )
data
```

| | Species | Weight | Length1 | Length2 | Length3 | Height | Width |
|---|---|---|---|---|---|---|---|
| **0** | Bream | 242.0 | 23.2 | 25.4 | 30.0 | 11.5200 | 4.0200 |
| **1** | Bream | 290.0 | 24.0 | 26.3 | 31.2 | 12.4800 | 4.3056 |
| **2** | Bream | 340.0 | 23.9 | 26.5 | 31.1 | 12.3778 | 4.6961 |
| **3** | Bream | 363.0 | 26.3 | 29.0 | 33.5 | 12.7300 | 4.4555 |
| **4** | Bream | 430.0 | 26.5 | 29.0 | 34.0 | 12.4440 | 5.1340 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **154** | Smelt | 12.2 | 11.5 | 12.2 | 13.4 | 2.0904 | 1.3936 |
| **155** | Smelt | 13.4 | 11.7 | 12.4 | 13.5 | 2.4300 | 1.2690 |
| **156** | Smelt | 12.2 | 12.1 | 13.0 | 13.8 | 2.2770 | 1.2558 |
| **157** | Smelt | 19.7 | 13.2 | 14.3 | 15.2 | 2.8728 | 2.0672 |
| **158** | Smelt | 19.9 | 13.8 | 15.0 | 16.2 | 2.9322 | 1.8792 |

159 rows × 7 columns

# Структури даних Pandas

До окремих об’єктів Series, якими є стовпці об'єкта DataFrame, можна звертатися за допомогою такої ж індексації, як і для словників, за іменем стовпця:

```
data['Height']
```

```
0    11.5200
1    12.4800
2    12.3778
3    12.7300
4    12.4440
     ...
154   2.0904
155   2.4300
156   2.2770
157   2.8728
158   2.9322
Name: Height, Length: 159, dtype: float64
```

або **`data.Height`**

# Структури даних Pandas

Також до елементів об'єкта DataFrame можна звертатись як до елементів масиву, за індексами.

```
data.values[1]
```

array(['Bream', 290.0, 24.0, 26.3, 31.2, 12.48, 4.3056], dtype=object)

```
data.values[1,4]
```

31.2

# Структури даних Pandas

З об’єктів DataFrame також можна виділяти підмасиви:

**`data[1:5]`**

| | Species | Weight | Length1 | Length2 | Length3 | Height | Width | ratio |
|---|---|---|---|---|---|---|---|---|
| **1** | Bream | 290.0 | 24.0 | 26.3 | 31.2 | 12.4800 | 4.3056 | 0.345000 |
| **2** | Bream | 340.0 | 23.9 | 26.5 | 31.1 | 12.3778 | 4.6961 | 0.379397 |
| **3** | Bream | 363.0 | 26.3 | 29.0 | 33.5 | 12.7300 | 4.4555 | 0.350000 |
| **4** | Bream | 430.0 | 26.5 | 29.0 | 34.0 | 12.4440 | 5.1340 | 0.412568 |

**`data[:10:2]`**

| | Species | Weight | Length1 | Length2 | Length3 | Height | Width | ratio |
|---|---|---|---|---|---|---|---|---|
| **0** | Bream | 242.0 | 23.2 | 25.4 | 30.0 | 11.5200 | 4.0200 | 0.348958 |
| **2** | Bream | 340.0 | 23.9 | 26.5 | 31.1 | 12.3778 | 4.6961 | 0.379397 |
| **4** | Bream | 430.0 | 26.5 | 29.0 | 34.0 | 12.4440 | 5.1340 | 0.412568 |
| **6** | Bream | 500.0 | 26.8 | 29.7 | 34.5 | 14.1795 | 5.2785 | 0.372263 |
| **8** | Bream | 450.0 | 27.6 | 30.0 | 35.1 | 14.0049 | 4.8438 | 0.345865 |

# Структури даних Pandas

З об’єктів DataFrame також можна виділяти підмасиви:

```
data[(data.Weight>200)&(data.Weight<300)]
```

| | Species | Weight | Length1 | Length2 | Length3 | Height | Width | ratio |
|---|---|---|---|---|---|---|---|---|
| **0** | Bream | 242.0 | 23.2 | 25.4 | 30.0 | 11.5200 | 4.0200 | 0.348958 |
| **1** | Bream | 290.0 | 24.0 | 26.3 | 31.2 | 12.4800 | 4.3056 | 0.345000 |
| **52** | Roach | 290.0 | 24.0 | 26.0 | 29.2 | 8.8768 | 4.4968 | 0.506579 |
| **53** | Roach | 272.0 | 25.0 | 27.0 | 30.6 | 8.5680 | 4.7736 | 0.557143 |
| **55** | Whitefish | 270.0 | 23.6 | 26.0 | 28.7 | 8.3804 | 4.2476 | 0.506849 |
| **56** | Whitefish | 270.0 | 24.1 | 26.5 | 29.3 | 8.1454 | 4.2485 | 0.521583 |
| **70** | Parkki | 273.0 | 23.0 | 25.0 | 28.0 | 11.0880 | 4.1440 | 0.373737 |
| **96** | Perch | 225.0 | 22.0 | 24.0 | 25.5 | 7.2930 | 3.7230 | 0.510490 |
| **101** | Perch | 218.0 | 25.0 | 26.5 | 28.0 | 7.1680 | 4.1440 | 0.578125 |
| **103** | Perch | 260.0 | 25.4 | 27.5 | 28.9 | 7.1672 | 4.3350 | 0.604839 |
| **104** | Perch | 265.0 | 25.4 | 27.5 | 28.9 | 7.0516 | 4.3350 | 0.614754 |

# Структури даних Pandas

Атрибут loc дозволяє виконати індексацію і зрізи з використанням прямого індексу. За допомогою індексатора iloc можна індексувати вихідний масив, як ніби це простий масив NumPy, але зі збереженням в результуючих даних міток об'єкта DataFrame для індексу і стовпців:

```
data.iloc[:4, :3]
```

| | Species | Weight | Length1 |
|---|---|---|---|
| 0 | Bream | 242.0 | 23.2 |
| 1 | Bream | 290.0 | 24.0 |
| 2 | Bream | 340.0 | 23.9 |
| 3 | Bream | 363.0 | 26.3 |

```
data.loc[:'1','Length2']
```

| | Species | Weight | Length1 | Length2 |
|---|---|---|---|---|
| 0 | Bream | 242.0 | 23.2 | 25.4 |
| 1 | Bream | 290.0 | 24.0 | 26.3 |

# Структури даних Pandas

Об’єкт DataFrame має кілька атрибутів.

Атрибут index надає тип об’єкту індексів та його значення.

```
df=pd.DataFrame([(1,3,5),(2,4,6)],columns=['first','seco
nd','third’])
```

| | first | second | third |
|---|---|---|---|
| **0** | 1 | 3 | 5 |
| **1** | 2 | 4 | 6 |

```
df.index
```

RangeIndex(start=0, stop=2, step=1)

```
df.columns
```

Index(['first', 'second', 'third'], dtype='object')

# Структури даних Pandas

**df.values**

```
array([[1, 3, 5],
       [2, 4, 6]], dtype=int64)
```

Імена стовпців можна змінити за допомогою методу rename. В якості аргументу цьому методу передається словник, ключі якого – старі назви стовпців, а значення – нові назви. Параметр inplace використовується для того, щоб зміни відбувались в самому об'єкті DataFrame.

df.rename(columns={'first':'I','second':'II','third':'III'},inplace=True)

df

| | I | II | III |
|---|---|---|---|
| **0** | 1 | 3 | 5 |
| **1** | 2 | 4 | 6 |

# Структури даних Pandas

Перейменувати стовпці можна і через відповідний атрибут columns, записавши нові назви в масиві.

```
df.columns=['I','II','III’]
```

Є декілька способів додати новий стовпець до DataFrame.

1) За допомогою оператору індексації [ ]

```
df['fourth']=[7,8]
df
```

| | first | second | third | fourth |
|---|---|---|---|---|
| **0** | 1 | 3 | 5 | 7 |
| **1** | 2 | 4 | 6 | 8 |

# Структури даних Pandas

2) За допомогою методу insert

```
df.insert(2,'two and half',[9,10])
```

```
df
```

| | first | second | two and half | third | fourth |
|---|---|---|---|---|---|
| **0** | 1 | 3 | 9 | 5 | 7 |
| **1** | 2 | 4 | 10 | 6 | 8 |

Перший аргумент – це індекс, за яким потрібно вставити новий стовпець, другий аргумент – назва нового стовпця, а третій – список значень нового стовпця.

# Структури даних Pandas

3) За допомогою loc

```
df.loc[:,'fourth']=[77,88]
df
```

| | first | second | two and half | third | fourth |
|---|---|---|---|---|---|
| **0** | 1 | 3 | 9 | 5 | 77 |
| **1** | 2 | 4 | 10 | 6 | 88 |

4) За допомогою функції concat

```
a=pd.Series([11,22])
df=pd.concat([df,a],axis=1)
df
```

| | first | second | third | fourth | 0 |
|---|---|---|---|---|---|
| **0** | 1 | 3 | 5 | 7 | 11 |
| **1** | 2 | 4 | 6 | 8 | 22 |

# Структури даних Pandas

До об’єкту DataFrame можна додавати рядки за допомогою функції pd.concat:

```
new_row=pd.DataFrame([{'first':1,      'second':2,
'third':3}])

pd.concat([df,new_row])
```

# Структури даних Pandas

Стовпець з DataFrame можна видалити наступними методами:

1) За допомогою функції del.

```
del df['third']
```

2) За допомогою методу pop

```
df.pop('second')
```

Метод pop видаляє стовпець в оригінальному DataFrame і повертає видалений стовпець як об’єкт Series.

3) За допомогою методу drop

```
df.drop(['first'],axis=1,inplace=True)
```

За замовчуванням цей метод видаляє рядки (axis=0), тому для видалення стовпця потрібно вказати значення параметру axis 1 .

# Структури даних Pandas

Також можна видаляти рядки з DataFrame.

1) Використовуючи вибір за допомогою умов та булевих виразів.

Наприклад, видалення всіх значень, що менші п’яти.

```
df[~(df.values<5)]
```

2) Використовуючи метод drop:

```
df.drop(1)
```

# Структури даних Pandas

Як об'єкт Series, так і об'єкт DataFrame містять явний індекс, що забезпечує можливість посилатися на дані і модифікувати їх. Об'єкт Index можна розглядати або як незмінний масив або як впорядкований набір елементів, що можуть повторюватись.

```
i=pd.Index(np.random.randint(25,size=(6)))
i
```

```
Int64Index([9, 0, 16, 12, 8, 5], dtype='int64’)
```

Об’єкт  Index може мати різні типи даних:
Index: загальний тип індексів, саме такий тип має індекс стовпців.
RangeIndex: тип індексу за замовчуванням, являє собою зростаючі цілі числа.
Int64Index: індекс, що складається з цілих чисел. На відміну від RangeIndex, вони не обов’язково повинні зростати та розташовуватись рівномірно.

# Структури даних Pandas

Float64Index: індекс з чисел з плаваючою комою.

IntervalIndex: індекс з інтервалів.

CategoricalIndex: індекс має обмежену та скінченну множину значень.

DateTimeIndex: індекс являє собою дату та час.

PeriodIndex: індекс являє собою період часу, наприклад, рік.

TimedeltaIndex: представляє проміжок часу між двома відліками часу, наприклад, двома датами.

MultiIndex: ієрархічний індекс з кількома рівнями.

# Структури даних Pandas

Об'єкт Index має такі ж атрибути, як ndarray і до його елементів так само можна звертатись за індексами, виділяти підмасиви

```
Int64Index([9, 0, 16, 12, 8, 5], dtype='int64')
i.size
6

i[-2]
8

i[:1:-1]
Int64Index([5, 8, 12, 16], dtype='int64')
```

# Структури даних Pandas

Але елементи об’єкту Index, на відміну від елементів масиву ndarray, не так легко змінити.
Наприклад, команда
```
i[0]=3
```
викличе помилку.

Індекс стовпців має деякі корисні атрибути:

1) атрибут values: повертає назви стовпців

```
periodic_table=pd.DataFrame({'Element':['Hydrogen',
'Helium','Lithium','Beryllium','Boron']},index=['H',
'He','Li','Be','B'])
column_index=periodic_table.columns
column_index.values
```

array([‘Element‘], dtype=object)

# Структури даних Pandas

2) Атрибут hasnans: повертає True або False, в залежності від наявності null значень.

```
column_index.hasnans
```

False

3) Атрибут nbytes: повертає зайнятий об’єм пам’яті у байтах

```
column_index.nbytes
```

Об’єкт Index, по суті, є множиною (хоча його елементи можуть повторюватись), тому над ним можна виконувати всі операції над множинами.

# Структури даних Pandas

Об’єднання множин:

```
a=pd.Index(np.random.randint(25,size=(6)))
b=pd.Index(np.random.randint(25,size=(6)))
a,b, a|b,a.union(b)
```

(Int64Index([10, 9, 13, 9, 14, 2], dtype='int64'),
Int64Index([23, 7, 19, 1, 14, 16], dtype='int64'),
Int64Index([1, 2, 7, 9, 9, 10, 13, 14, 16, 19, 23], dtype='int64'),
Int64Index([1, 2, 7, 9, 9, 10, 13, 14, 16, 19, 23], dtype='int64’))

Перетин множин:

```
a=pd.Index(np.random.randint(25,size=(6)))
b=pd.Index(np.random.randint(25,size=(6)))
a,b, a&b,a.intersection(b)
```

(Int64Index([2, 4, 23, 21, 12, 2], dtype='int64'),

Int64Index([0, 6, 22, 16, 23, 12], dtype='int64'),

Int64Index([23, 12], dtype='int64'),

Int64Index([23, 12], dtype='int64'))

# Структури даних Pandas

Різниця множин:

```
a=pd.Index(np.random.randint(25,size=(6)))
b=pd.Index(np.random.randint(25,size=(6)))
a,b,a.difference(b), b.difference(a)
```

(Int64Index([12, 1, 24, 9, 19, 10], dtype='int64'),

Int64Index([4, 1, 15, 15, 17, 18], dtype='int64'),

Int64Index([9, 10, 12, 19, 24], dtype='int64'),

Int64Index([4, 15, 17, 18], dtype='int64’))

Симетрична різниця

```
a. symmetric_difference(b)
```

# Структури даних Pandas

Також можна створити індекс зі значень стовпців:

```
data = pd.read_csv('Fish.csv’)
```

| | Species | Weight | Length1 | Length2 | Length3 | Height | Width |
|---|---|---|---|---|---|---|---|
| **0** | Bream | 242.0 | 23.2 | 25.4 | 30.0 | 11.5200 | 4.0200 |
| **1** | Bream | 290.0 | 24.0 | 26.3 | 31.2 | 12.4800 | 4.3056 |
| **2** | Bream | 340.0 | 23.9 | 26.5 | 31.1 | 12.3778 | 4.6961 |
| **3** | Bream | 363.0 | 26.3 | 29.0 | 33.5 | 12.7300 | 4.4555 |
| **4** | Bream | 430.0 | 26.5 | 29.0 | 34.0 | 12.4440 | 5.1340 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **154** | Smelt | 12.2 | 11.5 | 12.2 | 13.4 | 2.0904 | 1.3936 |
| **155** | Smelt | 13.4 | 11.7 | 12.4 | 13.5 | 2.4300 | 1.2690 |
| **156** | Smelt | 12.2 | 12.1 | 13.0 | 13.8 | 2.2770 | 1.2558 |
| **157** | Smelt | 19.7 | 13.2 | 14.3 | 15.2 | 2.8728 | 2.0672 |
| **158** | Smelt | 19.9 | 13.8 | 15.0 | 16.2 | 2.9322 | 1.8792 |

```
data.set_index(['Species’])
```

| | Weight | Length1 | Length2 | Length3 | Height | Width |
|---|---|---|---|---|---|---|
| **Species** | | | | | | |
| **Bream** | 242.0 | 23.2 | 25.4 | 30.0 | 11.5200 | 4.0200 |
| **Bream** | 290.0 | 24.0 | 26.3 | 31.2 | 12.4800 | 4.3056 |
| **Bream** | 340.0 | 23.9 | 26.5 | 31.1 | 12.3778 | 4.6961 |
| **Bream** | 363.0 | 26.3 | 29.0 | 33.5 | 12.7300 | 4.4555 |
| **Bream** | 430.0 | 26.5 | 29.0 | 34.0 | 12.4440 | 5.1340 |
| **...** | ... | ... | ... | ... | ... | ... |
| **Smelt** | 12.2 | 11.5 | 12.2 | 13.4 | 2.0904 | 1.3936 |
| **Smelt** | 13.4 | 11.7 | 12.4 | 13.5 | 2.4300 | 1.2690 |
| **Smelt** | 12.2 | 12.1 | 13.0 | 13.8 | 2.2770 | 1.2558 |
| **Smelt** | 19.7 | 13.2 | 14.3 | 15.2 | 2.8728 | 2.0672 |
| **Smelt** | 19.9 | 13.8 | 15.0 | 16.2 | 2.9322 | 1.8792 |

Або встановити індекс відразу при читанні файлу

```
data=pd.read_csv(‘Fish.csv’,index_col=‘Species’)
```

# Структури даних Pandas

З об’єктами Series та DataFrame працюють ті ж самі арифметичні операції, що й з масивами NumPy.

**`np.divide(data['Weight'],1000)`**

```
0      0.2420
1      0.2900
2      0.3400
3      0.3630
4      0.4300
        ...
154    0.0122
155    0.0134
156    0.0122
157    0.0197
158    0.0199
Name: Weight, Length: 159, dtype: float64
```

# Структури даних Pandas

При бінарних операціях над двома об'єктами Series або DataFrame бібліотека Pandas буде вирівнювати індекси в процесі виконання операції. Це дуже зручно при роботі з неповними даними.

```
a = pd.Series([1, 2, 3],index=['I', 'II', 'III'])
b = pd.Series([3, 4, 5],index=['II', 'III', 'IV'])
a+b
```

```
I     NaN
II    5.0
III   7.0
IV    NaN
dtype: float64
```

# Структури даних Pandas

Відсутні дані можна заповнити певним значенням:

```
a.multiply(b,fill_value=0)
```

I 0.0

II 6.0

III 12.0

IV 0.0

dtype: float64

```
a.multiply(b,fill_value=a.min())
```

I 1.0

II 6.0

III 12.0

IV 5.0

dtype: float64

# Структури даних Pandas

Так само вирівнювання відбувається при роботі з DataFrame

```
a = pd.DataFrame(np.random.randint(25, size=(4, 4)),
columns=['I','II','III','IV'])
b = pd.DataFrame(np.random.randint(25, size=(2, 2)),
columns=['III','II’])
```

**a**

| | I | II | III | IV |
|---|---|---|---|---|
| **0** | 9 | 18 | 12 | 1 |
| **1** | 10 | 11 | 18 | 13 |
| **2** | 6 | 17 | 11 | 17 |
| **3** | 1 | 8 | 20 | 0 |

**b**

| | III | II |
|---|---|---|
| **0** | 23 | 21 |
| **1** | 24 | 11 |

**a+b**

| | I | II | III | IV |
|---|---|---|---|---|
| **0** | NaN | 39.0 | 35.0 | NaN |
| **1** | NaN | 22.0 | 42.0 | NaN |
| **2** | NaN | NaN | NaN | NaN |
| **3** | NaN | NaN | NaN | NaN |

# Структури даних Pandas

Можливі також операції між об’єктами DataFrame та Series.

```
a=np.random.randint(50,size=(4))
b=np.random.randint(50,size=(4))
frame = pd.DataFrame({'a': a, 'b': b })
```

frame  frame-frame.iloc[0]  frame.add(frame.iloc[1:3])

| | a | b |
|---|---|---|
| 0 | 25 | 1 |
| 1 | 21 | 34 |
| 2 | 17 | 7 |
| 3 | 8 | 19 |

| | a | b |
|---|---|---|
| 0 | 0 | 0 |
| 1 | -4 | 33 |
| 2 | -8 | 6 |
| 3 | -17 | 18 |

| | a | b |
|---|---|---|
| 0 | NaN | NaN |
| 1 | 42.0 | 68.0 |
| 2 | 34.0 | 14.0 |
| 3 | NaN | NaN |

# Структури даних Pandas

Найбільш поширені типи даних в Pandas:

object – для суміші з різноманітних даних

int64 – для цілих чисел

float64 – для чисел з плаваючою комою

Datetime – для дати та часу

Category – для ознак, які приймають лише декілька різних значень

За допомогою атрибуту dtypes можна отримати типи стовпців в DataFrame.

# Структури даних Pandas

Щоб отримати кількість стовпців, які містять дані кожного типу, можна за допомогою методу get_dtype_counts:

```
df.get_dtype_counts()
```

За допомогою методу select_dtypes можна відфільтрувати стовпці на основі типу даних

```
df.select_dtypes(include='number')
```

Таким чином буде обрано всі стовпці, що містять цілі числа та числа з плаваючою комою.

Для того, щоб змінити тип даних, використовується метод astype:

```
fish['Species']=fish['Species'].astype('category')
```

# Структури даних Pandas

Просте об’єднання об’єктів Pandas можна виконати за допомогою методу pd.concat()

```
a = pd.Series([1, 2, 3],index=['I', 'II', 'III'])
b = pd.Series([4, 5, 6],index=['IV', 'V', 'VI'])
pd.concat([a, b])
```

```
I      1
II     2
III    3
IV     4
V      5
VI     6
dtype: int64
```

# Структури даних Pandas

За замовчанням конкатенація двох DataFrame відбувається за рядками. Об’єднати їх за стовпцями можна за допомогою параметра axis.

```
frame1 = pd.DataFrame({'a': a, 'b': b })
frame2 = pd.DataFrame({'a': b, 'b': a })
pd.concat([frame1,frame2])
```

| | a | b |
|---|---|---|
| I | 1 | 4 |
| II | 2 | 5 |
| III | 3 | 6 |
| I | 4 | 1 |
| II | 5 | 2 |
| III | 6 | 3 |

```
pd.concat([frame1,frame2],axis=1)
```

| | a | b | a | b |
|---|---|---|---|---|
| I | 1 | 4 | 4 | 1 |
| II | 2 | 5 | 5 | 2 |
| III | 3 | 6 | 6 | 3 |

# Структури даних Pandas

Якщо індекс не важливий (наприклад, це просто нумерація рядків в кожному з об’єктів), його можна проігнорувати при об’єднанні. Тоді у рядків результату буде утворена нова індексація:

```
a = pd.Series([1, 2, 3],index=[1,2,3])
b = pd.Series([4, 5, 6],index=[1,2,3])
frame1 = pd.DataFrame({'a': a, 'b': b })
frame2 = pd.DataFrame({'a': b, 'b': a })
```

pd.concat([frame1,frame2]),pd.concat([frame1,frame2], ignore_index=True)

| | a | b |
|---|---|---|
| **1** | 1 | 4 |
| **2** | 2 | 5 |
| **3** | 3 | 6 |
| **1** | 4 | 1 |
| **2** | 5 | 2 |
| **3** | 6 | 3 |

| | a | b |
|---|---|---|
| **0** | 1 | 4 |
| **1** | 2 | 5 |
| **2** | 3 | 6 |
| **3** | 4 | 1 |
| **4** | 5 | 2 |
| **5** | 6 | 3 |

# Структури даних Pandas

Якщо назви стовпців (або рядків, якщо об’єднання відбувається за ними) відрізняються, то при конкатенації утворюються неіснуючі значення:

```
frame1 = pd.DataFrame({'a': a, 'b': b })
frame2 = pd.DataFrame({'a': b, 'c': a })
pd.concat([frame1,frame2])
```

| | a | b | c |
|---|---|---|---|
| I | 1 | 4.0 | NaN |
| II | 2 | 5.0 | NaN |
| III | 3 | 6.0 | NaN |
| I | 4 | NaN | 1.0 |
| II | 5 | NaN | 2.0 |
| III | 6 | NaN | 3.0 |

# Структури даних Pandas

В таких випадках можна змінити значення параметру join на 'inner’. Тоді результат буде містити тільки ті стовпці (рядки), які є у обох початкових об’єктів, тобто виконується операція перетину множин. За замовчанням значення параметру join 'outer’, тобто відбувається об’єднання.

```
pd.concat([frame1,frame2],join='inner’)
```

| | a |
|---|---|
| I | 1 |
| II | 2 |
| III | 3 |
| I | 4 |
| II | 5 |
| III | 6 |

# Структури даних Pandas

Функція pd.merge () реалізує декілька типів з'єднань: «один-до-одного», «багато-до-одного» і «багато-до-багатьох». Всі ці три типи з'єднань доступні через один і той же виклик pd.merge (), тип виконуваного з'єднання залежить від форми вхідних даних.

Найпростіший тип злиття - з'єднання «один-до-одного», багато в чому нагадує конкатенацію за стовпцями.

```
a = pd.DataFrame({'City': ['A', 'B', 'C', 'D'],'population': [1000000, 245653,355223,4532434]})
b = pd.DataFrame({'City': ['A', 'C', 'D', 'B'],'region': ['a', 'b', 'c', 'c']})
c = pd.merge(a, b)
```

| | City | population | region |
|---|---|---|---|
| 0 | A | 1000000 | a |
| 1 | B | 245653 | c |
| 2 | C | 355223 | b |
| 3 | D | 4532434 | c |

# Структури даних Pandas

«Багато-до-одного» - з'єднання, при якому один з двох ключових стовпців містить значення, що дублюються. У разі з'єднання «багато-до-одного» в підсумковому об'єкті DataFrame ці два записи будуть збережені.

```
d = pd.DataFrame({'region': ['a', 'b', 'c'],'MainCity':
['mainA', 'mainB', 'mainC']})
pd.merge(c, d)
```

| | City | population | region | MainCity |
|---|---|---|---|---|
| **0** | A | 1000000 | a | mainA |
| **1** | B | 245653 | c | mainC |
| **2** | D | 4532434 | c | mainC |
| **3** | C | 355223 | b | mainB |

# Структури даних Pandas

Якщо ключовий стовпець як в лівому, так і в правому масивах містить повторювані значення, результат виявиться злиттям типу «багато-до-багатьох».

**d = pd.DataFrame({'region': ['a', 'b','c','a','b'],'Year': ['2020', '2020', '2020','2021','2021'],'Income':[230,654,435,270,640]})**

**pd.merge(c, d)**

| | City | population | region |
|---|---|---|---|
| **0** | A | 1000000 | a |
| **1** | B | 245653 | c |
| **2** | C | 355223 | b |
| **3** | D | 4532434 | c |

| | region | Year | Income |
|---|---|---|---|
| **0** | a | 2020 | 230 |
| **1** | b | 2020 | 654 |
| **2** | c | 2020 | 435 |
| **3** | a | 2021 | 270 |
| **4** | b | 2021 | 640 |

| | City | population | region | Year | Income |
|---|---|---|---|---|---|
| **0** | A | 1000000 | a | 2020 | 230 |
| **1** | A | 1000000 | a | 2021 | 270 |
| **2** | B | 245653 | c | 2020 | 435 |
| **3** | D | 4532434 | c | 2020 | 435 |
| **4** | C | 355223 | b | 2020 | 654 |
| **5** | C | 355223 | b | 2021 | 640 |

# Структури даних Pandas

За умовчанням метод pd.merge () виконує пошук в двох вхідних об'єктах відповідних назв стовпців і використовує знайдене в якості ключа. Однак найчастіше імена стовпців не збігаються точно, і в методі pd.merge () є чимало параметрів для такої ситуації.

По-перше, можна явно вказати назву стовпця, за якими проводити злиття.

```
pd.merge(a, b, on ='City’)
```

# Структури даних Pandas

Якщо назви стовпців відрізняються, то вони вказуються в параметрах left_on та right_on

```
a = pd.DataFrame({'City': ['A', 'B', 'C', 'D'],'population': [1000000, 245653,355223,4532434]})
b = pd.DataFrame({'Town': ['A', 'C', 'D', 'B'],'region': ['a', 'b', 'c', 'c']})
c=pd.merge(a, b, left_on="City", right_on="Town")
```

| | City | population | Town | region |
|---|---|---|---|---|
| **0** | A | 1000000 | A | a |
| **1** | B | 245653 | B | c |
| **2** | C | 355223 | C | b |
| **3** | D | 4532434 | D | c |

# Структури даних Pandas

Далі можна видалити зайвий стовпець

```
c=c.drop('Town',axis=1)
c
```

| | City | population | region |
|---|---|---|---|
| **0** | A | 1000000 | a |
| **1** | B | 245653 | c |
| **2** | C | 355223 | b |
| **3** | D | 4532434 | c |

# Структури даних Pandas

Параметр how методу merge вказує яка саме операція виконується над вмістом стовпця, за яким відбувається злиття. За замовчанням його значення 'inner', тобто перетин. Інші можливі значення - 'outer', 'left' та 'right'.

```
a = pd.DataFrame({'City': ['A', 'B', 'C',
'D'],'population': [1000000, 245653,355223,4532434]})
b = pd.DataFrame({'City': ['D', 'F', 'G', 'H'],'region':
['a', 'b', 'c', 'c']})
pd.merge(a, b)
```

| | City | population | region |
|---|---|---|---|
| 0 | D | 4532434 | a |

# Структури даних Pandas

```
pd.merge(a,b,how='outer')
```

| | City | population | region |
|---|---|---|---|
| 0 | A | 1000000.0 | NaN |
| 1 | B | 245653.0 | NaN |
| 2 | C | 355223.0 | NaN |
| 3 | D | 4532434.0 | a |
| 4 | F | NaN | b |
| 5 | G | NaN | c |
| 6 | H | NaN | c |

```
pd.merge(a,b,how='left')
```

| | City | population | region |
|---|---|---|---|
| 0 | A | 1000000 | NaN |
| 1 | B | 245653 | NaN |
| 2 | C | 355223 | NaN |
| 3 | D | 4532434 | a |

```
pd.merge(a, b,how='right')
```

| | City | population | region |
|---|---|---|---|
| 0 | D | 4532434.0 | a |
| 1 | F | NaN | b |
| 2 | G | NaN | c |
| 3 | H | NaN | c |

# Структури даних Pandas

Якщо дані містять однакові назви стовпців (за якими не відбувається злиття), то результат буде містити обидва ці стовпця з автоматично доданим суфіксом. Суфікс можна додати самостійно за допомогою параметру suffixes.

```
a = pd.DataFrame({'City': ['A', 'B', 'C',
'D'],'population': [1000000, 245653,355223,4532434]})
c = pd.DataFrame({'City': ['A', 'B', 'C',
'D'],'population': [1000020, 245700,355345,4532000]})
pd.merge(a, c,on='City')
```

| | City | population_x | population_y |
|---|---|---|---|
| 0 | A | 1000000 | 1000020 |
| 1 | B | 245653 | 245700 |
| 2 | C | 355223 | 355345 |
| 3 | D | 4532434 | 4532000 |

# Структури даних Pandas

```
pd.merge(a, c,on='City',suffixes=["_2000", "_2021"])
```

| | City | population_2000 | population_2021 |
|---|---|---|---|
| **0** | A | 1000000 | 1000020 |
| **1** | B | 245653 | 245700 |
| **2** | C | 355223 | 355345 |
| **3** | D | 4532434 | 4532000 |

# Структури даних Pandas

Дані можна розділити на певні групи за значенням за допомогою функції pd.cut().

```
pd.cut(data['Weight'], [0, 300, 1000,2000])
```

```
0         (0, 300]
1         (0, 300]
2      (300, 1000]
3      (300, 1000]
4      (300, 1000]
          ...
154       (0, 300]
155       (0, 300]
156       (0, 300]
157       (0, 300]
158       (0, 300]
Name: Weight, Length: 159, dtype: category
Categories (3, interval[int64]): [(0, 300] < (300, 1000] < (1000, 2000]]
```

# Структури даних Pandas

Функція pd.qcut () розділяє дані на квантилі. Окрім самих даних, вона приймає аргумент, що дорівнює кількості частин, на які розділяються дані, а масив з долями одиниці.

**`pd.qcut(data['Weight'],4)`**

```
0       (120.0, 273.0]
1       (273.0, 650.0]
2       (273.0, 650.0]
3       (273.0, 650.0]
4       (273.0, 650.0]
             ...
154    (-0.001, 120.0]
155    (-0.001, 120.0]
156    (-0.001, 120.0]
157    (-0.001, 120.0]
158    (-0.001, 120.0]
Name: Weight, Length: 159, dtype: category
Categories (4, interval[float64]): [(-0.001, 120.0] < (120.0, 273.0] < (273.0, 650.0] < (650.0, 1650.0]]
```

**`pd.qcut(data['Weight'],[0,0.25,0.5,0.75,1])`** поверне аналогічний результат

# Структури даних Pandas

Для об’єкту DataFrame можна викликати різні функції агрегування, в тому числі функції дескриптивної статистики, наприклад, mean чи std. Якщо одночасно потрібні всі основні статистичні характеристики, можна використати метод describe. Цей метод має параметр include.

```
data.describe()
```

| | Weight | Length1 | Length2 | Length3 | Height | Width |
|---|---|---|---|---|---|---|
| **count** | 159.000000 | 159.000000 | 159.000000 | 159.000000 | 159.000000 | 159.000000 |
| **mean** | 398.326415 | 26.247170 | 28.415723 | 31.227044 | 8.970994 | 4.417486 |
| **std** | 357.978317 | 9.996441 | 10.716328 | 11.610246 | 4.286208 | 1.685804 |
| **min** | 0.000000 | 7.500000 | 8.400000 | 8.800000 | 1.728400 | 1.047600 |
| **25%** | 120.000000 | 19.050000 | 21.000000 | 23.150000 | 5.944800 | 3.385650 |
| **50%** | 273.000000 | 25.200000 | 27.300000 | 29.400000 | 7.786000 | 4.248500 |
| **75%** | 650.000000 | 32.700000 | 35.500000 | 39.650000 | 12.365900 | 5.584500 |
| **max** | 1650.000000 | 59.000000 | 63.400000 | 68.000000 | 18.957000 | 8.142000 |

# Структури даних Pandas

Якщо є потреба отримати значення функцій агрегування (а також деяких інших) відносно підмасивів, виділених за певною ознакою, можна використати метод groupby. Сам метод groupby по суті створює об’єкт DataFrameGroupBy, у якого вже можна викликати інші методи.

```
data = pd.read_csv('Fish.csv’)
data.groupby('Species').mean()
```

| Species | Weight | Length1 | Length2 | Length3 | Height | Width |
|---|---|---|---|---|---|---|
| **Bream** | 617.828571 | 30.305714 | 33.108571 | 38.354286 | 15.183211 | 5.427614 |
| **Parkki** | 154.818182 | 18.727273 | 20.345455 | 22.790909 | 8.962427 | 3.220736 |
| **Perch** | 382.239286 | 25.735714 | 27.892857 | 29.571429 | 7.861870 | 4.745723 |
| **Pike** | 718.705882 | 42.476471 | 45.482353 | 48.717647 | 7.713771 | 5.086382 |
| **Roach** | 152.050000 | 20.645000 | 22.275000 | 24.970000 | 6.694795 | 3.657850 |
| **Smelt** | 11.178571 | 11.257143 | 11.921429 | 13.035714 | 2.209371 | 1.340093 |
| **Whitefish** | 531.000000 | 28.800000 | 31.316667 | 34.316667 | 10.027167 | 5.473050 |

# Структури даних Pandas

```
data.groupby('Species'). describe()
```

| | Weight | | | | | | | | Length1 | | ... | Height | | Width | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | count | mean | std | min | 25% | 50% | 75% | max | count | mean | ... | 75% | max | count | mean | std |
| Species | | | | | | | | | | | | | | | | |
| **Bream** | 35.0 | 617.828571 | 209.205709 | 242.0 | 462.50 | 610.00 | 717.00 | 1000.0 | 35.0 | 30.305714 | ... | 16.360900 | 18.9570 | 35.0 | 5.427614 | 0.721509 |
| **Parkki** | 11.0 | 154.818182 | 78.755086 | 55.0 | 105.00 | 145.00 | 185.00 | 300.0 | 11.0 | 18.727273 | ... | 10.041100 | 11.3680 | 11.0 | 3.220736 | 0.643347 |
| **Perch** | 56.0 | 382.239286 | 347.617717 | 5.9 | 120.00 | 207.50 | 692.50 | 1100.0 | 56.0 | 25.735714 | ... | 10.846500 | 12.8002 | 56.0 | 4.745723 | 1.774626 |
| **Pike** | 17.0 | 718.705882 | 494.140765 | 200.0 | 345.00 | 510.00 | 950.00 | 1650.0 | 17.0 | 42.476471 | ... | 8.926200 | 10.8120 | 17.0 | 5.086382 | 1.140269 |
| **Roach** | 20.0 | 152.050000 | 88.828916 | 0.0 | 104.25 | 147.50 | 171.75 | 390.0 | 20.0 | 20.645000 | ... | 7.164150 | 9.4850 | 20.0 | 3.657850 | 0.690371 |
| **Smelt** | 14.0 | 11.178571 | 4.131526 | 6.7 | 8.95 | 9.85 | 12.20 | 19.9 | 14.0 | 11.257143 | ... | 2.261225 | 2.9322 | 14.0 | 1.340093 | 0.286611 |
| **Whitefish** | 6.0 | 531.000000 | 309.602972 | 270.0 | 279.00 | 423.00 | 735.00 | 1000.0 | 6.0 | 28.800000 | ... | 11.506900 | 12.3540 | 6.0 | 5.473050 | 1.194258 |

7 rows × 48 columns

# Структури даних Pandas

```
data.groupby('Species')['Weight'].std()
```

```
Species
Bream        209.205709
Parkki        78.755086
Perch        347.617717
Pike         494.140765
Roach         88.828916
Smelt          4.131526
Whitefish    309.602972
Name: Weight, dtype: float64
```

Об’єкт DataFrameGroupBy по суті містить кілька DataFrame. Отримати доступ до однієї з них можна наступним чином:

```
grouped_fish=data.groupby(‘Species’)
grouped_fish.get_group(‘Smelt’)
```

# Структури даних Pandas

Об’єкт DataFrameGroupBy має метод aggregate() або agg(), що дозволяє, наприклад, виконувати кілька операцій агрегування:

```
data.groupby('Species')['Weight’].agg([sum,np.prod,np.mean])
```

| Species | sum | prod | mean |
|---|---|---|---|
| Bream | 21624.0 | 5.756482e+96 | 617.828571 |
| Parkki | 1703.0 | 3.021954e+23 | 154.818182 |
| Perch | 21405.4 | 6.110171e+132 | 382.239286 |
| Pike | 12218.0 | 1.067144e+47 | 718.705882 |
| Roach | 3041.0 | 0.000000e+00 | 152.050000 |
| Smelt | 156.5 | 2.206766e+14 | 11.178571 |
| Whitefish | 3186.0 | 9.636797e+15 | 531.000000 |

# Структури даних Pandas

Або застосовувати різні операції до різних стовпців:

```
data.groupby('Species').agg({'Weight':
max,'Height':np.median})
```

| Species | Weight | Height |
|---|---|---|
| Bream | 1000.0 | 14.9544 |
| Parkki | 300.0 | 8.8928 |
| Perch | 1100.0 | 6.9218 |
| Pike | 1650.0 | 7.2900 |
| Roach | 390.0 | 6.5126 |
| Smelt | 19.9 | 2.2002 |
| Whitefish | 1000.0 | 9.7610 |

# Структури даних Pandas

Об’єкт DataFrameGroupBy має метод filter(), що дозволяє фільтрувати групи. Вхідним аргументом методу filter() є попередньо визначена функція або лямбда функція.

```
def filtmax(d):
    return d['Weight'].max()<900
data.groupby('Species').filter(filtmax)
```

| | Species | Weight | Length1 | Length2 | Length3 | Height | Width |
|---|---|---|---|---|---|---|---|
| **35** | Roach | 40.0 | 12.9 | 14.1 | 16.2 | 4.1472 | 2.2680 |
| **36** | Roach | 69.0 | 16.5 | 18.2 | 20.3 | 5.2983 | 2.8217 |
| **37** | Roach | 78.0 | 17.5 | 18.8 | 21.2 | 5.5756 | 2.9044 |
| **38** | Roach | 87.0 | 18.2 | 19.8 | 22.2 | 5.6166 | 3.1746 |
| **39** | Roach | 120.0 | 18.6 | 20.0 | 22.2 | 6.2160 | 3.5742 |
| **40** | Roach | 0.0 | 19.0 | 20.5 | 22.8 | 6.4752 | 3.3516 |
| **41** | Roach | 110.0 | 19.1 | 20.8 | 23.1 | 6.1677 | 3.3957 |
| **42** | Roach | 120.0 | 19.4 | 21.0 | 23.7 | 6.1146 | 3.2943 |

```
Grouped_fish.filter(lambda x:x[‘Weight’].max()<900)
```

# Структури даних Pandas

Наступний метод об’єкту DataFrameGroupBy - transform(), який дозволяє перетворювати дані, тобто застосовувати операції до всіх даних. Наприклад, можна центрувати дані:

```
data.groupby('Species').transform(lambda x: x - x.mean())
```

| | Weight | Length1 | Length2 | Length3 | Height | Width |
|---|---|---|---|---|---|---|
| **0** | -375.828571 | -7.105714 | -7.708571 | -8.354286 | -3.663211 | -1.407614 |
| **1** | -327.828571 | -6.305714 | -6.808571 | -7.154286 | -2.703211 | -1.122014 |
| **2** | -277.828571 | -6.405714 | -6.608571 | -7.254286 | -2.805411 | -0.731514 |
| **3** | -254.828571 | -4.005714 | -4.108571 | -4.854286 | -2.453211 | -0.972114 |
| **4** | -187.828571 | -3.805714 | -4.108571 | -4.354286 | -2.739211 | -0.293614 |
| **...** | ... | ... | ... | ... | ... | ... |
| **154** | 1.021429 | 0.242857 | 0.278571 | 0.364286 | -0.118971 | 0.053507 |
| **155** | 2.221429 | 0.442857 | 0.478571 | 0.464286 | 0.220629 | -0.071093 |
| **156** | 1.021429 | 0.842857 | 1.078571 | 0.764286 | 0.067629 | -0.084293 |

# Структури даних Pandas

Наступний метод об’єкту DataFrameGroupBy - apply(), дозволяє застосовувати будь-яку функцію.

```
def applyf (d):
    d['Weight'] /= d['Height']
    return d
data.groupby('Species').apply(applyf)
```

| | Species | Weight | Length1 | Length2 | Length3 | Height | Width |
|---|---|---|---|---|---|---|---|
| **0** | Bream | 21.006944 | 23.2 | 25.4 | 30.0 | 11.5200 | 4.0200 |
| **1** | Bream | 23.237179 | 24.0 | 26.3 | 31.2 | 12.4800 | 4.3056 |
| **2** | Bream | 27.468532 | 23.9 | 26.5 | 31.1 | 12.3778 | 4.6961 |
| **3** | Bream | 28.515318 | 26.3 | 29.0 | 33.5 | 12.7300 | 4.4555 |
| **4** | Bream | 34.554806 | 26.5 | 29.0 | 34.0 | 12.4440 | 5.1340 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **154** | Smelt | 5.836204 | 11.5 | 12.2 | 13.4 | 2.0904 | 1.3936 |
| **155** | Smelt | 5.514403 | 11.7 | 12.4 | 13.5 | 2.4300 | 1.2690 |

# Структури даних Pandas

Метод apply застосовує функцію до кожної групи. Метод apply є більш гнучким, ніж метод transform, оскільки він може повертати об’єкт будь-якої форми, а метод transform повинен повернути об’єкт такої ж форми. До того ж, метод transform застосовує функцію до кожного стовпця групи, а метод apply – до всієї групи.

```
data.groupby('Species').apply(lambda
x:x.isna().sum())
```

# Структури даних Pandas

Для скорочення багатовимірного агрегування можна використовувати метод pivot_table().

```
data = pd.read_csv('Laptop-Users.csv')
```

| | Age | Gender | Region | Occupation | Income | Has Laptop |
|---|---|---|---|---|---|---|
| **0** | 14 | male | city | student | 0 | no |
| **1** | 34 | female | city | teacher | 22000 | no |
| **2** | 42 | male | countryside | banker | 24000 | yes |
| **3** | 30 | male | countryside | teacher | 25000 | no |
| **4** | 16 | male | city | student | 0 | no |
| **5** | 33 | female | city | banker | 20000 | yes |
| **6** | 26 | female | city | student | 8000 | yes |
| **7** | 22 | male | city | student | 4000 | yes |
| **8** | 28 | female | countryside | teacher | 12000 | no |
| **9** | 27 | female | city | student | 7000 | yes |

# Структури даних Pandas

```
data.pivot_table(index='Gender', columns='Occupation')
```

| | Age | | | | Income | | | |
|---|---|---|---|---|---|---|---|---|
| Occupation | banker | officer | student | teacher | banker | officer | student | teacher |
| Gender | | | | | | | | |
| female | 31.000000 | 49.0 | 26.333333 | 31.0 | 19000.000000 | 25000.0 | 8666.666667 | 17000.0 |
| male | 40.333333 | 54.0 | 20.166667 | 28.0 | 25666.666667 | 30000.0 | 4000.000000 | 19500.0 |

# Структури даних Pandas

```
age = pd.cut(data['Age'], [0, 21, 35, 90])
data.pivot_table(index=['Gender',age],
columns='Occupation')
```

| | | Age | | | | Income | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Occupation | banker | officer | student | teacher | banker | officer | student | teacher |
| Gender | Age | | | | | | | | |
| female | (21, 35] | 31.000000 | NaN | 26.333333 | 31.0 | 19000.000000 | NaN | 8666.666667 | 17000.0 |
| | (35, 90] | NaN | 49.0 | NaN | NaN | NaN | 25000.0 | NaN | NaN |
| male | (0, 21] | NaN | NaN | 16.000000 | NaN | NaN | NaN | 3000.000000 | NaN |
| | (21, 35] | NaN | NaN | 28.500000 | 28.0 | NaN | NaN | 6000.000000 | 19500.0 |
| | (35, 90] | 40.333333 | 54.0 | NaN | NaN | 25666.666667 | 30000.0 | NaN | NaN |

# Структури даних Pandas

Якщо деякі вхідні дані є символьними рядками, то для роботи з ними в Pandas існує цілий ряд методів, які аналогічні відповідним методам Python, але працюють відразу з усіма елементами об’єкту, не потребують використання циклів і обробляють порожні значення.

```
a = pd.Series(['Kyiv',np.nan,'Paris','LONDON'])
a.str.capitalize()
```

```
0      Kyiv
1       NaN
2     Pariz
3    London
dtype: object
```

```
a.str.len()
```

```
0    4.0
1    NaN
2    5.0
3    6.0
dtype: float64
```