

ЗАТВЕРДЖУЮ  
Директор ЗПО «ДІТК»  
31.08.2026 р.  
Олександр СТРИЛЕЦЬ  
ПОГОДЖЕНО  
Заступник директора з навчально-методичної  
роботи ЗПО «ДІТК»  
31.08.2026 р.  
Сергій ГОРДІЄНКО

## МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДЛЯ ПРАКТИЧНИХ ЗАВДАНЬ «АНАЛІТИКА ДАНИХ ТА ЦИФРОВІ ІНСТРУМЕНТИ В СУЧАСНОМУ МАШИНОБУДУВАННІ»

### ВСТУП

Практичні роботи з дисципліни «Аналітика даних та цифрові інструменти в сучасному машинобудуванні» є невід'ємною складовою освітнього процесу. Їхня мета — не просто закріпити теоретичний матеріал, а сформувати у вас практичні навички, які ви зможете застосувати безпосередньо на виробництві: від аналізу якості зварних швів до прогнозування відмов обладнання.

Сучасне машинобудування неможливо уявити без роботи з даними. Кожен верстат, кожен зварювальний апарат, кожен датчик генерує інформацію, яка потребує обробки та аналізу. Інженер, який вміє працювати з даними, має значну перевагу: він приймає рішення на основі фактів, а не інтуїції, він може виявити проблему до того, як вона призведе до аварії, він може оптимізувати процес і знизити витрати.

Практичні роботи цього курсу побудовані за принципом «від простого до складного». Ви починаєте з налаштування середовища Google Colab, опановуєте основи Python та бібліотеки Pandas, вчитеся завантажувати, очищати та аналізувати виробничі дані. Поступово ви переходите до статистичного аналізу, візуалізації, регресійного моделювання та основ машинного навчання. Кожна робота є кроком до формування цілісної аналітичної компетентності.

### Практична робота №1

**Мета роботи:** ознайомитися з основними кроками по роботі з даними. Встановити необхідне програмне забезпечення для подальшої роботи з даними. Ознайомитись з програмним середовищем. Ознайомитись з бібліотекою **NumPy**.

## NumPy: Створення масивів

**Масив** — це структурована сукупність фіксованої кількості елементів одного типу, доступ до яких здійснюється за допомогою індексів. Елементи масиву називаються індексними змінними.

```
numpy.array(object, dtype=None, *, copy=True, order='K',  
subok=False, ndmin=0, like=None).
```

**Параметри функції `array()`:**

- **object (array\_like)** - Масив, будь-який об'єкт, що відповідає інтерфейсу масиву, об'єкт, чий метод `array` повертає масив, або будь-яка (вкладена)
- послідовність. Якщо об'єкт є скаляром, повертається 0-вимірний масив, що містить об'єкт;
- **dtype (data-type, optional)** - Бажаний тип даних для масиву. Якщо не вказано, тип буде визначено як мінімальний тип, необхідний для зберігання об'єктів у послідовності;
- **copy (bool, optional)** - Якщо значення `true` (за замовчуванням), об'єкт копіюється. В іншому випадку копія буде створена, лише якщо `array` повертає копію, якщо `obj` є вкладеною послідовністю або якщо копія потрібна для задоволення
- будь-яких інших вимог (`dtype`, порядок тощо);
- **order ({'K', 'A', 'C', 'F'}, optional)** - Вкажіть структуру пам'яті масиву. Якщо об'єкт не є масивом, новостворений масив буде в порядку `C` (основний рядок).

Якщо об'єкт є масивом, має місце наступне.

| order | no copy  | copy=True                                        |
|-------|----------|--------------------------------------------------|
|       |          | е                                                |
| 'K'   | без змін | Порядок F & C збережено, інакше порядок схожий   |
| 'A'   | без змін | F-порядок, якщо вхід F, а не C, інакше C-порядок |
| 'C'   | C order  | C order                                          |
| 'F'   | F order  | F order                                          |

- **subok (bool, optional)** - Якщо `True`, то підкласи будуть передані наскрізно, інакше повернений масив буде змушений бути масивом базового класу (за замовчуванням);

- **ndmin (optional)** - Визначає мінімальну кількість розмірів, яку повинен мати отриманий масив. Щоб відповідати цій вимозі, їх буде додано до фігури;
- **like (optional)** - Посилальний об'єкт, який дозволяє створювати масиви, які не є масивами NumPy. Якщо array-like, переданий як like, підтримує протокол array\_function, результат буде визначено ним. У цьому випадку він забезпечує створення об'єкта масиву, сумісного з тим, що передається через цей аргумент.

```
numpy.zeros(shape, dtype=float, order='C', *,
like=None) numpy.ones(shape, dtype=None,
order='C', *, like=None)
```

**Параметри функції zeros() та ones():**

- **shape (int or tuple of ints)** - Форма нового масиву, наприклад, (2, 3) або 2;
- **dtype (data-type, optional)** - Бажаний тип даних для масиву, наприклад, numpy.int8. Типовим є numpy.float64;
- **order ({'C', 'F'}, optional)** - Чи потрібно зберігати багатовимірні дані в пам'яті в порядку старших рядків (стиль C) або основних стовпців (стиль Фортрана);
- **like (array\_like, optional)** - Посилальний об'єкт, який дозволяє створювати масиви, які не є масивами NumPy. Якщо array-like, переданий як like, підтримує протокол array\_function\_, результат буде визначено ним. У цьому випадку він забезпечує створення об'єкта масиву, сумісного з тим, що передається через цей аргумент.

```
numpy.full(shape, fill_value, dtype=None, order='C', *, like=None)
```

**Параметри функції full():**

- **shape (int or sequence of ints)** - Форма нового масиву, наприклад, (2, 3) або 2.
- **fill\_value (scalar or array\_like)** - Значення заповнення.
- **dtype (data-type, optional)** - Бажаний тип даних для масиву.
- **order (array\_like, optional)** - Чи зберігати в пам'яті багатовимірні дані в C- або F-суміжному порядку (по рядках або стовпцях).
- **like (array\_like, optional)** - Посилальний об'єкт, який дозволяє створювати масиви, які не є масивами NumPy. Якщо array-like, переданий як like, підтримує протокол

- `array_function_`, результат буде визначено ним. У цьому випадку він забезпечує створення об'єкта масиву, сумісного з тим, що передається через цей аргумент.

**`numpy.arange([start, ]stop, [step, ]dtype=None, *, like=None)`**

**Параметри функції `arange()`:**

- **`start (integer or real, optional)`** - Початок інтервалу. Інтервал включає це значення. Початкове значення за умовчанням — 0.
- **`stop (integer or real)`** - Кінець інтервалу. Інтервал не включає це значення, за винятком деяких випадків, коли крок не є цілим числом і округлення з плаваючою комою впливає на довжину вихідного.
- **`step (integer or real, optional)`** - Інтервал між значеннями. Для будь-якого виводу `out` це відстань між двома суміжними значеннями, `out[i+1]` - `out[i]`. Розмір кроку за замовчуванням дорівнює 1. Якщо крок вказано як аргумент позиції, необхідно також вказати початок.
- **`dtype (dtype, optional)`** - Тип вихідного масиву. Якщо `dtype` не задано, виведіть тип даних з інших вхідних аргументів.
- **`like (array_like, optional)`** - Посилальний об'єкт, який дозволяє створювати масиви, які не є масивами NumPy. Якщо `array-like`, переданий як `like`, підтримує протокол
- `array_function_`, результат буде визначено ним. У цьому випадку він забезпечує створення об'єкта масиву, сумісного з тим, що передається через цей аргумент.

**`numpy.eye(N, M=None, k=0, dtype=<class 'float'>, order='C', *, like=None)`**

**Параметри функції `eye()`:**

- **`N (int)`** - Кількість рядків у виведенні.
- **`M (int, optional)`** - Кількість стовпців у вихідних даних. Якщо немає, за замовчуванням `N`.
- **`k (int, optional)`** - Індекс діагоналі: 0 (за замовчуванням) відноситься до головної діагоналі, позитивне значення відноситься до верхньої діагоналі, а негативне значення відноситься до нижньої діагоналі.
- **`dtype (data-type, optional)`** - Тип даних отриманого масиву.
- **`order ({'C', 'F'}, optional)`** - Визначає, чи слід зберігати в пам'яті вихідні дані в
- порядку основних рядків (у стилі C) чи в порядку стовпців (у стилі Фортрана).
- **`like (array_like, optional)`** - Посилальний об'єкт, який дозволяє

створювати масиви, які не є масивами NumPy. Якщо array-like, переданий як like, підтримує протокол array\_function\_, результат буде визначено ним. У цьому випадку він забезпечує створення об'єкта масиву, сумісного з тим, що передається через цей аргумент.

**numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)**

**Параметри функції linspace():**

- **start (array\_like)** - Початкове значення послідовності.
- **stop (array\_like)** - Кінцеве значення послідовності, якщо для кінцевої точки не встановлено значення False. У цьому випадку послідовність складається з усіх, крім останнього,  $num + 1$  рівномірно розташованих вибірок, так що зупинка виключається. Зауважте, що розмір кроку змінюється, коли кінцева точка має значення False.
- **num (int, optional)** - Кількість зразків для створення. За замовчуванням 50. Має бути невід'ємним.
- **endpoint (bool, optional)** - Якщо True, зупинка є останнім зразком. В іншому випадку він не включається. Типовим значенням є True.
- **retstep (bool, optional)** - Якщо True, повернути (зразки, крок), де крок — це відстань між вибірками.
- **dtype (dtype, optional)** - Тип вихідного масиву. Якщо dtype не вказано, тип даних виводиться з початку та зупинки. Виведений dtype ніколи не буде цілим; float вибирається, навіть якщо аргументи створять масив цілих чисел.
- **axis (int, optional)** - Вісь результату для зберігання зразків. Релевантно, лише якщо початок або зупинка подібні до масиву. За замовчуванням (0) зразки будуть уздовж нової осі, вставленої на початку. Використовуйте -1, щоб отримати вісь у кінці.

**numpy.random.rand(d0, d1, ..., dn)**

**Параметри функції rand():**

- **d0, d1, ..., dn (int, optional)** - Розміри поверненого масиву мають бути невід'ємними. Якщо не вказано жодного аргументу, повертається єдине число з плаваючою точкою Python.

`numpy.random.randn(d0, d1, ..., dn)`

**Параметри функції `randn()`:**

- `d0, d1, ..., dn (int, optional)` - Розміри поверненого масиву мають бути невід'ємними. Якщо не вказано жодного аргументу, повертається єдине число з плаваючою точкою Python.

`numpy.random.normal(loc=0.0, scale=1.0, size=None)`

**Параметри функції `normal()`:**

- `loc (float or array_like of floats)` - Середнє («центр») розподілу.
- `scale (float or array_like of floats)` - Стандартне відхилення (розкид або «ширина»)
- розподілу. Має бути невід'ємним.
- `size (int or tuple of ints, optional)` - Вихідна форма. Якщо задана форма, наприклад, (m, n, k), то буде взято  $m * n * k$  зразків. Якщо значення «Size» — «Немає» (за замовчуванням), повертається одне значення, якщо `loc` і `scale` є скалярними.

`numpy.random.randint(low, high=None, size=None, dtype=int)`

**Параметри функції `randint()`:**

- `low (int or array-like of ints)` - Найменші (зі знаком) цілі числа, які будуть взяті з розподілу (якщо тільки `high=None`, у цьому випадку цей параметр на одиницю перевищує найвище таке ціле число).
- `high (int or array-like of ints, optional)` - Якщо надано, один над найбільшим (зі знаком) цілим числом, яке буде взято з розподілу (див. вище поведінку, якщо `high=None`). Якщо подібний до масиву, має містити цілі значення
- `size (int or tuple of ints, optional)` - Вихідна форма. Якщо задана форма, наприклад, (m, n, k), то буде взято  $m * n * k$  зразків. За замовчуванням значення «Немає», у цьому випадку повертається одне значення.
- `dtype (dtype, optional)` - Бажаний dtype результату. Порядок байтів має бути рідним. Значенням за замовчуванням є `int`.

### Приклад

Запускаємо редактор Jupyter Notebook та створюємо новий документ. Незабуваємо підключити бібліотеку `import numpy as np`.

Функція `np.array()`

```
In [7]: new_variable = np.array([5.8,9,7,5])

In [8]: new_variable
Out[8]: array([5.8, 9. , 7. , 5. ])

In [9]: new_variable2 = np.array([5.8,9,7,5], dtype = 'int')

In [10]: new_variable2
Out[10]: array([5, 9, 7, 5])
```

Функція `np.zeros()`

```
In [11]: np.zeros(5)
Out[11]: array([0., 0., 0., 0., 0.])

In [12]: np.zeros((3, 5), dtype = 'int')
Out[12]: array([[0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0],
                [0, 0, 0, 0, 0]])
```

Функція `np.ones()`

```
In [13]: np.ones(10)
Out[13]: array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])

In [14]: np.ones((3,4,5), dtype = 'int')
Out[14]: array([[[1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1]],
                [[1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1]],
                [[1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1],
                 [1, 1, 1, 1, 1]]])
```

Функція `np.full()`

```
In [15]: np.full(8, 5)
```

```
Out[15]: array([5, 5, 5, 5, 5, 5, 5, 5])
```

```
In [16]: np.full((2, 3), 6, dtype = 'float')
```

```
Out[16]: array([[6., 6., 6.],  
               [6., 6., 6.]])
```

Функція `np.arange()`

```
In [17]: np.arange(5)
```

```
Out[17]: array([0, 1, 2, 3, 4])
```

```
In [18]: example = np.arange(3, 10, dtype = 'float')
```

```
In [19]: print(example)
```

```
[3. 4. 5. 6. 7. 8. 9.]
```

Функція `np.eye()`

```
In [21]: np.eye(3, 2)
```

```
Out[21]: array([[1., 0.],  
               [0., 1.],  
               [0., 0.]])
```

```
In [22]: np.eye(3, 2, dtype = 'bool')
```

```
Out[22]: array([[ True, False],  
               [False,  True],  
               [False, False]])
```

```
In [23]: np.eye(3)
```

```
Out[23]: array([[1., 0., 0.],  
               [0., 1., 0.],  
               [0., 0., 1.]])
```

Функція `np.linspace()`

```
In [24]: np.linspace(10, 30, 10)
```

```
Out[24]: array([10., 12.22222222, 14.44444444, 16.66666667, 18.88888889,  
               21.11111111, 23.33333333, 25.55555556, 27.77777778, 30.])
```

Функції `np.random.rand()`, `np.random.randn()`, `np.random.normal()`,

## np.random.randint()

```
In [25]: np.random.rand(5)
Out[25]: array([0.20211382, 0.05519226, 0.38375847, 0.70578249, 0.50677819])

In [26]: np.random.rand(2, 3)
Out[26]: array([[0.55695822, 0.10980014, 0.23223474],
               [0.96297729, 0.4540903 , 0.52720781]])

In [27]: np.random.randn(5)
Out[27]: array([-1.01897804, -1.18332752, -1.0048376 ,  0.10713674, -0.30218364])

In [28]: np.random.normal(20, 5, (4, 3))
Out[28]: array([[26.5847506 , 16.76253676, 17.90554931],
               [18.88581537, 29.08242406,  9.37940301],
               [29.30084469, 19.77650537, 15.62365778],
               [22.74737457, 23.99841369, 23.21102703]])

In [29]: np.random.randint(0, 10, (4, 4))
Out[29]: array([[5, 3, 4, 9],
               [9, 3, 1, 0],
               [5, 8, 5, 5],
               [2, 3, 1, 2]])
```

## Практична робота №2

**Мета роботи:** ознайомитися з функціями для роботи з масивами у бібліотеці NumPy.

### Порядок виконання роботи

1. Створіть новий файл. Приєднайте бібліотеку NumPy [1].  
Попрактикуйте навички використання функцій, описаних у попередньому розділі. Та опишіть отримані дані.
2. Отримайте дані з довільного масиву та опишіть їх, використовуючи наступні функції: **ndim, shape, size, dtype**.
3. Попрактикуйте та опишіть результат з іншими функціями з [офіційної документації](#).

## NumPy: Функції з масивами

`numpy.reshape(a, newshape, order='C')`

Параметри функції `reshape()`:

- `a (array_like)` - Масив, який потрібно змінити.
- `newshape (int or tuple of ints)` - Нова форма має бути сумісною з оригінальною формою. Якщо це ціле число, то результатом буде одновимірний масив такої довжини. Один розмір форми може бути - 1. У цьому випадку значення
- виводиться з довжини масиву та інших розмірів.
- `order ({'C', 'F', 'A'}, optional)` - Прочитайте елементи `a`, використовуючи цей порядок індексів, і розмістіть елементи в зміненому масиві, використовуючи цей порядок індексів. «C» означає читати/записувати елементи за допомогою C-подібного порядку індексів, при цьому індекс останньої осі змінюється найшвидше, а назад до індексу першої осі — найповільніше. «F» означає читати/записувати елементи за допомогою порядку індексів.
- змінюється найшвидше, а останній індекс — найповільніше. Зауважте, що параметри «C» і «F» не враховують структуру пам'яті основного масиву, а стосуються лише порядку індексування.

`matrix.max(axis=None, out=None)` `matrix.min(axis=None, out=None)`

Параметри функції `max()` та `min()`:

- `axis (None or int or tuple of ints, optional)` - Вісь або осі, уздовж яких потрібно працювати. За замовчуванням використовується зведений ввід.
- `out (ndarray, optional)` - Альтернативний вихідний масив, у який можна розмістити результат. Має бути такої ж форми та довжини буфера, що й очікуваний результат.

`numpy.argmax(a, axis=None, out=None, *, keepdims=<no value>)`

`numpy.argmin(a, axis=None, out=None, *, keepdims=<no value>)`

Параметри функції `argmax()` та `argmin()`:

- `a (array_like)` - Вхідний масив.
- `axis (int, optional)` - За замовчуванням індекс знаходиться у зведеному

масиві, інакше вздовж вказаної осі.

- `out (array, optional)` - Якщо надано, результат буде вставлено в цей масив. Він має бути відповідної форми та dtype.
- `keepdims (bool, optional)` - Якщо для цього значення встановлено значення True, зменшені осі залишаються в результаті як розміри з розміром один. За допомогою цього параметра результат буде правильно транслюватися щодо масиву.

```
numpy.concatenate((a1, a2, ...), axis=0, out=None, dtype=None, casting="same_kind")
```

Параметри функції `concatenate()`:

- `a1, a2, ... (sequence of array_like)` - Масиви повинні мати однакову форму, за винятком розміру, що відповідає осі (перша, за замовчуванням).
- `axis (int, optional)` - Вісь, по якій будуть з'єднуватися масиви. Якщо axis має значення None, масиви зводяться перед використанням. За замовчуванням 0.
- `out (ndarray, optional)` - Якщо вказано, призначення для розміщення результату. Форма має бути правильною, збігатися з формою, яку б повернув об'єднаний, якби не було вказано вихідний аргумент.
- `dtype (str or dtype)` - Якщо вказано, цільовий масив матиме цей dtype. Не можна надати разом із out.
- `casting ({'no', 'equiv', 'safe', 'same_kind', 'unsafe'}, optional)` - Контролює тип передачі даних. За умовчанням встановлено «same\_kind».

```
numpy.split(ary, indices_or_sections, axis=0)
```

```
numpy.hsplit(ary, indices_or_sections, axis=0)
```

```
numpy.vsplit(ary, indices_or_sections, axis=0)
```

Параметри функції `split()`:

- `ary (ndarray)` - Масив, який потрібно розділити на підмасиви.
- `indices_or_sections (int or 1-D array)` - Якщо `indices_or_sections` є цілим числом, N, масив буде розділено на N рівних масивів уздовж осі. Якщо такий розподіл неможливий, виникає помилка. Якщо `indices_or_sections` є одновимірним масивом відсортованих цілих чисел, записи вказують, де вздовж осі розділено масив.

Наприклад, [2, 3] для `axis=0` призведе до

`ary[:2]`

`ary [2:3]`

`ary [3:]`

Якщо індекс перевищує розмір масиву вздовж осі, відповідно повертається порожній підмасив.

`axis (int, optional)` - Вісь, уздовж якої потрібно розділити, за умовчанням дорівнює 0.

`numpy.sort(a, axis=-1, kind=None, order=None)`

Параметри функції `sort()`:

- `a (array_like)` - Масив для сортування
- `axis (int or None, optional)` - Вісь, уздовж якої сортується. Якщо немає, масив зрівнюється перед сортуванням. Типовим значенням є -1, який сортує вздовж останньої осі.
- `kind ({'quicksort', 'mergesort', 'heapsort', 'stable'}, optional)` - Алгоритм сортування. Типовим є «quicksort». Зауважте, що як «stable», так і «mergesort» під обкладинками використовуються сортування за системою `tim` або `radix`, і, загалом, фактична реалізація буде відрізнятися залежно від типу даних.
- Параметр «mergesort» збережено для зворотної сумісності.
- `order (str or list of str, optional)` - Коли `a` є масивом із визначеними полями, цей аргумент визначає, які поля порівнювати першими, другими тощо. Окреме поле можна вказати як рядок, і не потрібно вказувати всі поля, але невизначені поля все одно використовуватимуться в порядок, у якому вони з'являються в `dtype`, щоб розірвати зв'язки.

### Приклад

Запускаємо редактор Jupyter Notebook та створюємо новий документ. Незабуваємо приєднати бібліотеку `import numpy as np`.

Функція `reshape()`

```
In [4]: example.reshape((3,5))
```

```
Out[4]: array([[ 1,  2,  3,  4,  5],
               [ 6,  7,  8,  9, 10],
               [11, 12, 13, 14, 15]])
```

Функція **max()** та **argmax()**

```
Out[11]: array([40, 36, 34, 22,  5,  2,  7, 15, 48,  7])
```

```
In [12]: array.max()
```

```
Out[12]: 48
```

```
In [13]: array.argmax()
```

```
Out[13]: 8
```

Функція **min()** та **argmin()**

```
Out[19]: array([35, 21, 57, 58, 35, 40, 51, 34, 61, 63])
```

```
In [20]: array.min()
```

```
Out[20]: 21
```

```
In [21]: array.argmin()
```

```
Out[21]: 1
```

Функція **concatenate()**

```
In [27]: array1 = np.array([1, 2, 3, 4])
```

```
In [28]: array2 = np.array([5, 6, 7, 8])
```

```
In [29]: np.concatenate([array1, array2])
```

```
Out[29]: array([1, 2, 3, 4, 5, 6, 7, 8])
```

Функція **np.split()**, **np.hsplit()** та **np.vsplit()**

```
In [36]: array = np.array([1, 3, 5, 50, 50, 2, 4, 6])
```

```
In [37]: np.split(array, [3,5])
```

```
Out[37]: [array([1, 3, 5]), array([50, 50]), array([2, 4, 6])]
```

```
In [43]: array = np.arange(20).reshape(5, 4)
```

```
In [44]: array
```

```
Out[44]: array([[ 0,  1,  2,  3],
                [ 4,  5,  6,  7],
                [ 8,  9, 10, 11],
                [12, 13, 14, 15],
                [16, 17, 18, 19]])
```

```
In [45]: np.split(array, [1, 3])
```

```
Out[45]: [array([[0, 1, 2, 3]]),
          array([[ 4,  5,  6,  7],
                 [ 8,  9, 10, 11]]),
          array([[12, 13, 14, 15],
                 [16, 17, 18, 19]])]
```

```
In [53]: np.vsplit(array, 5)
```

```
Out[53]: [array([[0, 1, 2, 3]]),
          array([[4, 5, 6, 7]]),
          array([[ 8,  9, 10, 11]]),
          array([[12, 13, 14, 15]]),
          array([[16, 17, 18, 19]])]
```

```
In [51]: np.hsplit(array, 4)
```

```
Out[51]: [array([[ 0],
                [ 4],
                [ 8],
                [12],
                [16]]),
          array([[ 1],
                [ 5],
                [ 9],
                [13],
                [17]]),
          array([[ 2],
                [ 6],
                [10],
                [14],
                [18]]),
          array([[ 3],
                [ 7],
                [11],
                [15],
                [19]])]
```

## Функція `np.sort()`

```
In [72]: example = np.random.normal(30, 10, (4, 4))
```

```
In [73]: example
```

```
Out[73]: array([[19.45447124, 37.05669862, 26.52466308, 34.66670306],
                [32.07414191, 23.70572052, 33.8136543 , 27.26563817],
                [50.22261404, 18.99548472, 29.69887488, 38.34999885],
                [35.62523607, 39.44571749, 41.02167529, 29.67967764]])
```

```
In [74]: np.sort(example , axis = 1)

Out[74]: array([[19.45447124, 26.52466308, 34.66670306, 37.05669862],
               [23.70572052, 27.26563817, 32.07414191, 33.8136543 ],
               [18.99548472, 29.69887488, 38.34999885, 50.22261404],
               [29.67967764, 35.62523607, 39.44571749, 41.02167529]])

In [75]: np.sort(example , axis = 0)

Out[75]: array([[19.45447124, 18.99548472, 26.52466308, 27.26563817],
               [32.07414191, 23.70572052, 29.69887488, 29.67967764],
               [35.62523607, 37.05669862, 33.8136543 , 34.66670306],
               [50.22261404, 39.44571749, 41.02167529, 38.34999885]])
```

### Практична робота №3

**Мета роботи:** ознайомитися з функціями для роботи з масивами у бібліотеці **NumPy**.

#### Порядок виконання роботи

1. Створіть новий файл. Підключіть бібліотеку **NumPy**.
2. Виконайте наступні завдання:
  - 2.1. Напишіть програму **NumPy** для створення одновимірного масиву з 20 елементів. Тепер створіть новий масив форми (5, 4) із зазначеного масиву.
  - 2.2. З попереднього масиву отримайте максимальне та мінімальне значення, а також їхні індекси значення.
  - 2.3. Напишіть програму **NumPy** для створення об'єднання двох заданих масивів форми (2, 2) і (2,1).
  - 2.4. Напишіть програму **NumPy**, щоб розділити масив з 14 елементів на 3 масиви, кожен з яких містить 2, 4 і 8 елементів у вихідному порядку.
  - 2.5. Напишіть програму **NumPy**, щоб розділити масив розміром 4x4 на два масиви вздовж другої осі.
  - 2.6. Напишіть програму **NumPy**, щоб розділити заданий масив на кілька підмасивів по вертикалі (по рядках).
  - 2.7. Напишіть програму **NumPy** для сортування багатовимірного вздовж першої та останньої осі масиву.
3. Оформіть звіт

## NumPy: Функції з масивами

Важливо зазначити, що індексація **Python** починається з 0.

Синтаксис індексації такий:

- `x[start:end:step]`: елементи в масиві `x` від стартового індекса `i` до кінцевого індекса (але кінець не включено), значення кроку за умовчанням дорівнює 1.
- `x[start:end]`: елементи в масиві `x` від стартового індекса `i` до кінцевого індекса (але кінець не включено)
- `x[start:]`: елементи від стартового індекса до кінця масиву
- `x[:end]`: елементи від початку масиву `i` до кінцевого індекса (але кінець не включено)

### Приклад

Запускаємо редактор Jupyter Notebook та створюємо новий документ.

Незабуваємо підключити бібліотеку `import numpy as np`.

Якщо ми хочемо отримати 3-й елемент, ми запишемо індекс як 2, оскільки він починається з 0.

```
x = np.arange(10)
```

```
x[2]
```

```
x[2:5]
```

```
x[::2]
```

```
x[1::2]
```

```
x
```

```
Вихід: [0 1 2 3 4 5 6 7 8 9]
```

```
x[2]
```

```
Вихід: 2
```

```
x[2:5]
```

```
Вихід: array([2, 3, 4])
```

```
x[::2]
```

```
Вихід: array([0, 2, 4, 6, 8])
```

```
x[1::2]
```

```
Вихід: array([1, 3, 5, 7, 9])
```

Зверніть увагу, що в `x[2:5]` вибираються елементи, починаючи з 2-го індексу до 5-го індексу (не включає). Якщо ми хочемо змінити значення всіх елементів від початку до індексу 7, за винятком 7, з кроком 3 як 123 ми пишемо:

```
x[:7:3] = 123
```

x

Вихід: `array([123, 1, 2, 123, 4, 5, 123, 7, 8, 9])`

Щоб звернути заданий масив, ми пишемо:

`x = np.arange(10)`

`x[::-1]` # повернуто x

Вихід: `array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])`

Зауважте, що наведена вище команда не змінює вихідний масив.

**Розглянемо 1D масив.**

`x = np.arange(11,35,2)`

x

Вихід: `array([11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33])`

Ми формуємо одновимірний масив i, який містить підмножини елементів x наступним чином:

`i = np.array([0,1,5,3,7,9])x[i]`

Вихід: `array([11, 13, 21, 17, 25, 29])`

Подібним чином ми створюємо двовимірний масив j індексів для підмножини x.

`j = np.array( [ [ 0, 1], [ 6, 2 ] ] )`

`x[j]`

Вихід: `array([[11, 13],  
[23, 15]])`

Подібним чином ми можемо створити як i, так і j як двовимірні масиви індексів для x

`x = np.arange(15).reshape(3,5)`

x

`i = np.array( [ [0,1], [2,0] ] )`

# індекси для першого dim

`j = np.array( [ [1,1], [2,0] ] )`

# індекси для другого розм

Щоб отримати i-й індекс у рядку та j-й індекс для стовпців, ми пишемо: `x[i,j]`

# i і j повинні мати однакову форму

Вихід: `array([[ 1, 6], [12, 0]])`

Щоб отримати індекс i з 3-го стовпця, ми пишемо: `x[i,2]`

Вихід: `array([[ 2, 7], [12, 2]])`

Для кожного рядка, якщо ми хочемо знайти j-й індекс, ми пишемо: `x[:,j]`

Вихід: `array([[[ 1, 1],  
[ 2, 0]],`

```
[[ 6, 6],  
 [ 7, 5]],  
 [[11, 11],  
 [12, 10]]])
```

Фіксація 1-го рядка та j-го індексу, фіксація 2-го рядка j-го індексу, фіксація 3-го рядка та j-го індексу.

Ви також можете використовувати індексування з масивами для призначення значень:

```
x = np.arange(10)  
x[[4,5,8,1,2]] = 0  
x
```

Вихід: `array([0, 0, 0, 3, 0, 0, 6, 7, 0, 9])`  
0 присвоюється 4, 5, 8, 1 і 2 індексам x.

Якщо список індексів містить повторення, він присвоює цьому індексу останнє значення:

```
x = np.arange(10)  
x[[4,4,2,3]] = [100,200,300,400]  
x
```

Вихід: `array([ 0, 1, 300, 400, 200, 5, 6, 7, 8, 9])`

Зауважте, що для 5-го елемента (тобто 4-го індексу) присвоєно значення 200, а не 100.

Застереження: якщо оператор += використовується для повторюваних індексів, тоді оператор виконується лише один раз для повторюваних індексів.

```
x = np.arange(10)  
x[[1,1,1,7,7]]+=1  
x
```

Вихід: `array([0, 2, 2, 3, 4, 5, 6, 8, 8, 9])`

Хоча індекси 1 і 7 повторюються, але вони збільшуються лише один раз.

## Практична робота №4

**Мета роботи:** ознайомитися з базовим та розширеним індексуванням масивів у бібліотеці NumPy

### Порядок виконання роботи

1. Створіть новий файл. Підключіть бібліотеку **NumPy**.
2. Виконайте наступні завдання:
  - 2.1. Напишіть програму **NumPy** для створення одновимірного масиву з 20 елементів. Тепер створіть новий масив форми (5, 4) із зазначеного масиву. Отримайте масив елементів з третьої колонки всіх рядків.
  - 2.2. З попереднього завдання використайте оригінальний масив та отримайте

з нього новий підмасив, що складається з непарних рядків і парних стовпців.

2.3. Отримайте перші чотири стовпці цього 2-D масиву

`exercise = np.arange(100).reshape(5,-1)`

2.4. Замініть усі непарні числа у заданому масиві на -1 `exercise =`

`np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])`

2.5. Дано наступний масив: `y = np.array([ 0, -1, -2, -3, -4, -5])`

Використовуйте розширене індексування, використовуючи масивицілих чисел, щоб створити такі масиви:

#1 `array([-1])`

#2 `array([-1, -2, -1, -2])`

#3 `array([[ 0, -5],  
[-1, -4]])`

#4 `array([[ -2],  
[ -3],  
[ -2]])`

2.6. Дано наступний масив: `z = np.array([[[ 0, 1, 2, 3],`

`[ 4, 5, 6, 7],`

`[ 8, 9, 10, 11]],`

`[[12, 13, 14, 15],`

`[16, 17, 18, 19],`

`[20, 21, 22, 23]]])`

Використовуйте розширене індексування, використовуючи масивицілих чисел, щоб створити такі масиви:

#1 `array([[ 0, 5, 10],  
[12, 17, 22]])`

#2 `array([[ 0, 23],  
[23, 0]])`

3. Оформіть звіт

## NumPy: Операції з масивами

Операції з операторами порівняння

Оператори:

- > - більше
- < - меншу
  - == - рівне
- != - не рівне
- >= - більше рівне
- <= - менше рівне

Порівняння масивів і скалярів

```
import numpy as np
arr = np.array([-1, 0, 1])
arr > 0
# array([False, False,  True])
arr >= 0
array([False,  True,  True])
arr < 0
# array([ True, False, False])
arr <= 0
# array([ True,  True, False])
arr == 0
# array([False,  True, False])
arr != 0
# array([ True, False,  True])
```

Порівняння масивів з масивами

Результатом логічної операції є булевий масив, що містить двійкові значення True або False. Цей конкретний випадок показує нам логічні операції масивів зі скалярами. Усі булеві операції виконуються поелементно, тому кожен елемент

порівнюється зі скаляром. Ми також можемо порівнювати масиви з масивами, якщо це дозволяє їх форма.

```
arr_one = np.array([-1, 0, 1])
arr_two = np.array([1, 0, -1])
arr_one > arr_two
# array([False, False,  True])
arr_one >= arr_two
# array([False,  True,  True])
arr_one < arr_two
# array([ True, False, False])
arr_one <= arr_two
# array([ True,  True, False])
arr_one == arr_two
# array([False,  True, False])
arr_one != arr_two
array([ True, False,  True])
```

Арифметичні операції

`numpy.subtract(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj])`

Параметри функції `subtract()`:

`x1, x2 (array_like)` - Масиви, які потрібно відняти один від одного. Якщо `x1.shape != x2.shape`, вони повинні транслюватись у загальну форму (яка стає формою виведення).

`out (ndarray, None, or tuple of ndarray and None, optional)` - місце, де зберігається результат. Якщо передбачено, воно має мати форму, до якої транслюються вхідні дані. Якщо не надано або немає, повертається щойно виділений масив. Довжина кортежу (можлива лише як ключовий аргумент) повинна дорівнювати кількості виходів.

`where (array_like, optional)` - ця умова транслюється через вхід. У місцях, де умова має значення `True`, вихідний масив буде встановлено на результат `ufunc`. В інших місцях вихідний масив збереже своє початкове значення. Зауважте, що якщо неініціалізований вихідний масив створюється за допомогою параметра за замовчуванням `out=None`, місця в ньому, де умова має значення `False`, залишаться неініціалізованими.

`numpy.add(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj])`

Параметри функції `add()`:

`x1, x2 (array_like)` - Масиви, які потрібно додати. Якщо `x1.shape != x2.shape`, вони повинні транслюватись у загальну форму (яка стає формою виведення).

`out (ndarray, None, or tuple of ndarray and None, optional)` - Місце, де зберігається результат. Якщо передбачено, воно має мати форму, до якої транслюються вхідні дані. Якщо не надано або немає, повертається щойно виділений масив. Довжина кортежу (можлива лише як ключовий аргумент) повинна дорівнювати кількості виходів.

`where (array_like, optional)` - Ця умова транслюється через вхід. У місцях, де умова має значення `True`, вихідний масив буде встановлено на результат `ufunc`. В інших місцях вихідний масив збереже своє початкове значення. Зауважте, що якщо неініціалізований вихідний масив створюється за допомогою параметра за замовчуванням `out=None`, місця в ньому, де умова має значення `False`, залишаться неініціалізованими.

`numpy.divide(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj])`

Параметри функції `divide()`:

`x1 (array_like)` - Масив дивідендів.

`x2 (array_like)` - Масив дільника. Якщо `x1.shape != x2.shape`, вони повинні транслятись у загальну форму (яка стає формою виведення).

`out (ndarray, None, or tuple of ndarray and None, optional)` - Місце, де зберігається результат. Якщо передбачено, воно має мати форму, до якої транслуються вхідні дані. Якщо не надано або немає, повертається щойно виділений масив. Довжина кортежу (можлива лише як ключовий аргумент) повинна дорівнювати кількості виходів.

`where (array_like, optional)` - Ця умова транслюється через вхід. У місцях, де умова має значення `True`, вихідний масив буде встановлено на результат `ufunc`. В інших місцях вихідний масив збереже своє початкове значення. Зауважте, що якщо неініціалізований вихідний масив створюється за допомогою параметра за замовчуванням `out=None`, місця в ньому, де умова має значення `False`, залишаться неініціалізованими.

`numpy.multiply(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj])`

Параметри функції `multiply()`:

`x1, x2 (array_like)` - Вхідні масиви для множення. Якщо `x1.shape != x2.shape`, вони повинні транслятись у загальну форму (яка стає формою виведення).

`out (ndarray, None, or tuple of ndarray and None, optional)` - Місце, де зберігається результат. Якщо передбачено, воно має мати форму, до якої транслуються вхідні дані. Якщо не надано або немає, повертається щойно виділений масив. Довжина кортежу (можлива лише як ключовий аргумент) повинна дорівнювати кількості виходів.

`where (array_like, optional)` - Ця умова транслюється через вхід. У місцях, де умова має значення `True`, вихідний масив буде встановлено на результат `ufunc`. В інших місцях вихідний масив збереже своє початкове значення. Зауважте, що якщо неініціалізований вихідний масив створюється за допомогою параметра за замовчуванням `out=None`, місця в ньому, де умова має значення `False`, залишаться неініціалізованими.

`numpy.exp(x, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj])`

Параметри функції `exp()`:

- `x (array_like)` - Вхідні значення.
- `out (ndarray, None, or tuple of ndarray and None, optional)` - Місце, де зберігається результат. Якщо передбачено, воно має мати форму, до якої транслюються вхідні дані. Якщо не надано або немає, повертається щойно виділений масив. Довжина кортежу (можлива лише як ключовий аргумент) повинна дорівнювати кількості виходів.
- `where (array_like, optional)` - Ця умова транслюється через вхід. У місцях, де умова має значення True, вихідний масив буде встановлено на результат `ufunc`. В інших місцях вихідний масив збереже своє початкове значення. Зауважте, що якщо неініціалізований вихідний масив створюється за допомогою параметра за замовчуванням `out=None`, місця в ньому, де умова має значення False, залишаться неініціалізованими.

`numpy.sqrt(x, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj])`

Параметри функції `sqrt()`:

- `x (array_like)` - Значення, для яких потрібні квадратні корені.
- `out (ndarray, None, or tuple of ndarray and None, optional)` - Місце, де зберігається результат. Якщо передбачено, воно має мати форму, до якої транслюються вхідні дані. Якщо не надано або немає, повертається щойно виділений масив. Довжина кортежу (можлива лише як ключовий аргумент) повинна дорівнювати кількості виходів.
- `where (array_like, optional)` - Ця умова транслюється через вхід. У місцях, де умова має значення True, вихідний масив буде встановлено на результат `ufunc`. В інших місцях вихідний масив збереже своє початкове значення. Зауважте, що якщо неініціалізований вихідний масив створюється за допомогою параметра за замовчуванням `out=None`, місця в ньому, де умова має значення False, залишаться неініціалізованими.

Статистичні операції (агрегатні функції)

`numpy.sum(a, axis=None, dtype=None, out=None, keepdims=<no value>, initial=<no value>, where=<no value>)`

Параметри функції `sum()`:

- `a (array_like)` - Елементи для підсумовування.
- `axis (None or int or tuple of ints, optional)` - Вісь або осі, по яких виконується підсумовування. За замовчуванням `axis=None` підсумовує всі елементи вхідного масиву. Якщо вісь від'ємна, відлік ведеться від останньої до першої осі. Якщо `axis` кортежем `ints`, сума виконується на всіх осях, указаних у кортежі, замість однієї осі або всіх осей, як раніше.
- `dtype (dtype, optional)` - Тип повернутого масиву та накопичувача, у

якому

- підсумовуються елементи. Dtype а використовується за замовчуванням, якщо а не має ціле dtype меншої точності, ніж ціле число платформи за замовчуванням. У цьому випадку, якщо а зі знаком, використовується ціле число платформи, тоді як, якщо а без знака, використовується ціле число без знаку такої ж точності, як і ціле число платформи.
- **out (ndarray, optional)** - Альтернативний вихідний масив, у який можна розмістити результат. Він повинен мати ту саму форму, що й очікуваний вихід, але тип вихідних значень буде приведено, якщо необхідно.
- **keepdims (bool, optional)** - Якщо для цього значення встановлено значення True, зменшені осі залишаються в результаті як розміри з розміром один. За допомогою цього параметра результат буде правильно трансляватися щодо вхідного масиву.
- Якщо передано значення за замовчуванням, keepdims не буде передано до методу sum підкласів ndarray, однак будь-яке значення, відмінне від замовчування, буде. Якщо метод підкласу не реалізує keepdims, будуть створені будь-які винятки.

**initial (scalar, optional)** - Початкове значення для суми.

**where (array\_like of bool, optional)** - Елементи для включення в суму.

**numpy.cumsum(a, axis=None, dtype=None, out=None)**

Параметри функції **cumsum()**:

- **a (array\_like)** - Вхідний масив.
- **axis (int, optional)** - Вісь, уздовж якої обчислюється кумулятивна сума. За замовчуванням (немає) обчислюється загальна сума для зведеного масиву.
- **dtype (dtype, optional)** - Тип поверненого масиву та накопичувача, у якому
- підсумовуються елементи. Якщо dtype не вказано, за замовчуванням буде dtypea, якщо а не має цілого dtype з точністю, нижчою за ціле число платформи за замовчуванням. У цьому випадку використовується ціле число платформи за замовчуванням.
- **out (ndarray, optional)** - Альтернативний вихідний масив, у який можна розмістити результат. Він повинен мати таку саму форму та довжину буфера, що й очікуваний результат, але тип буде приведено, якщо це необхідно.

**numpy.mean(a, axis=None, dtype=None, out=None, keepdims=<no value>, \*, where=<no value>)**

Параметри функції **mean()**:

- **a (array\_like)** - Масив чисел із бажаним середнім. Якщо а не є масивом, виконується спроба перетворення.

- **axis (None or int or tuple of ints, optional)** - Вісь або осі, уздовж яких обчислюються середні значення. За умовчанням обчислюється середнє значення зведеного масиву. Якщо це кортеж із числами, середнє значення виконується по кількох осях, а не по одній осі чи по всіх осях, як раніше.
- **dtype (dtype, optional)** - Тип для використання в обчисленні середнього значення.
- Для цілочисельних введів за замовчуванням є float64; для вхідних даних з плаваючою комою це те саме, що вхід dtype.

**out (ndarray, optional)** - Альтернативний вихідний масив, у який потрібно розмістити результат. Типовим значенням є None; якщо він наданий, він повинен мати ту саму форму, що й очікуваний результат, але тип буде відлитий, якщо необхідно. **keepdims (bool, optional)** - Якщо для цього значення встановлено значення True, зменшені осі залишаються в результаті як розміри з розміром один. За допомогою цього параметра результат буде правильно транслюватися щодо вхідного масиву.

Якщо передано значення за замовчуванням, keepdims не буде передано до методу mean підкласів ndarray, однак будь-яке значення не за замовчуванням буде. Якщо метод підкласу не реалізує keepdims, будуть створені будь-які винятки. **where (array\_like of bool, optional)** - Елементи для включення в середнє значення.

**numpy.median(a, axis=None, out=None, overwrite\_input=False, keepdims=False)**

Параметри функції **median()**:

- **a (array\_like)** - Вхідний масив або об'єкт, який можна перетворити на масив.
- **axis ({int, sequence of int, None}, optional)** - Вісь або осі, уздовж яких обчислюються медіани. За замовчуванням медіана обчислюється вздовж згладженої версії масиву. Послідовність осей підтримується з версії 1.9.0.
- **out (ndarray, optional)** - Альтернативний вихідний масив, у який можна розмістити результат. Він повинен мати ту саму форму та довжину буфера, що й очікуваний вихід, але тип (виходу) буде наведено, якщо необхідно.
- **overwrite\_input (bool, optional)** - Якщо True, то дозволяється використання пам'яті вхідного масиву а для обчислень. Вхідний масив буде змінено викликом медіани. Це заощадить пам'ять, коли вам не потрібно зберігати вміст вхідного масиву.
- Розглядати вхідні дані як невизначені, але вони, ймовірно, будуть повністю або частково відсортовані. Типовим значенням є False. Якщо

`overwrite_input` має значення `True`, а `a` ще не є `ndarray`, виникне помилка.

- `keepdims (bool, optional)` - Якщо для цього значення встановлено значення `True`, зменшені осі залишаються в результаті як розміри з розміром один. З цією опцією результат транслюватиметься коректно порівняно з оригінальним масивом.

`numpy.std(a, axis=None, dtype=None, out=None, ddof=0, keepdims=<no value>, *, where=<no value>)`

Параметри функції `std()`:

- `a (array_like)` - Обчисліть стандартне відхилення цих значень.
- `axis (None or int or tuple of ints, optional)` - Вісь або осі, уздовж яких обчислюється стандартне відхилення. За умовчанням обчислюється стандартне відхилення зведеного масиву. Якщо це кортеж із числами, стандартне відхилення виконується по кількох осях замість однієї осі чи всіх осей, як раніше.
- `dtype (dtype, optional)` - Тип для використання в обчисленні стандартного відхилення. Для масивів цілочисельного типу значенням за замовчуванням є `float64`, для масивів типів `float` воно збігається з типом масиву.

`out (ndarray, optional)` - Альтернативний вихідний масив, у який можна розмістити результат. Він повинен мати ту саму форму, що й очікуваний вихід, але тип (обчислених значень) буде приведено, якщо це необхідно.

`ddof (int, optional)` - Означає дельта ступенів свободи. Дільник, який використовується в обчисленнях:  $N - ddof$ , де  $N$  означає кількість елементів. За замовчуванням `ddof` дорівнює нулю.

`keepdims (bool, optional)` - Якщо для цього значення встановлено значення `True`, зменшені осі залишаються в результаті як розміри з розміром один. За допомогою цього параметра результат буде правильно транслюватися щодо вхідного масиву.

Якщо передано значення за замовчуванням, `keepdims` не буде передано до методу `std` підкласів `ndarray`, однак будь-яке значення не за замовчуванням буде. Якщо метод підкласу не реалізує `keepdims`, будуть створені будь-які винятки. `where (array_like of bool, optional)` - Елементи для включення в стандартне відхилення.

## Приклад

Нижче наведені приклади операцій над масивами. більш детально можна глянути на “[шпаргалки](https://assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf)” для роботи у бібліотеці NumPy ([https://assets.datacamp.com/blog\\_assets/Numpy\\_Python\\_Cheat\\_Sheet.pdf](https://assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf))

### Arithmetic Operations

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                    |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>&gt;&gt;&gt; g = a - b array([[ -0.5,  0. ,  0. ],        [-3. , -3. , -3. ]]) &gt;&gt;&gt; np.subtract(a,b) &gt;&gt;&gt; b + a array([[ 2.5,  4. ,  6. ],        [ 5. ,  7. ,  9. ]]) &gt;&gt;&gt; np.add(b,a) &gt;&gt;&gt; a / b array([[ 0.66666667,  1. ,  1.5 ],        [ 0.25 ,  0.4 ,  0.5 ]]) &gt;&gt;&gt; np.divide(a,b) &gt;&gt;&gt; a * b array([[ 1.5,  4. ,  9. ],        [ 4. , 10. , 18. ]]) &gt;&gt;&gt; np.multiply(a,b) &gt;&gt;&gt; np.exp(b) &gt;&gt;&gt; np.sqrt(b) &gt;&gt;&gt; np.sin(a) &gt;&gt;&gt; np.cos(b) &gt;&gt;&gt; np.log(a) &gt;&gt;&gt; e.dot(f) array([[ 7.,  7.],        [ 7.,  7.]])</pre> | <p>Subtraction</p> <p>Subtraction<br/>Addition</p> <p>Addition<br/>Division</p> <p>Division<br/>Multiplication</p> <p>Multiplication<br/>Exponentiation<br/>Square root<br/>Print sines of an array<br/>Element-wise cosine<br/>Element-wise natural logarithm<br/>Dot product</p> |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### Comparison

|                                                                                                                                                                                                          |                                                                                            |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
| <pre>&gt;&gt;&gt; a == b array([[False,  True,  True],        [False, False, False]], dtype=bool) &gt;&gt;&gt; a &lt; 2 array([ True, False, False], dtype=bool) &gt;&gt;&gt; np.array_equal(a, b)</pre> | <p>Element-wise comparison</p> <p>Element-wise comparison</p> <p>Array-wise comparison</p> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|

### Aggregate Functions

|                                                                                                                                                                                                              |                                                                                                                                                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>&gt;&gt;&gt; a.sum() &gt;&gt;&gt; a.min() &gt;&gt;&gt; b.max(axis=0) &gt;&gt;&gt; b.cumsum(axis=1) &gt;&gt;&gt; a.mean() &gt;&gt;&gt; b.median() &gt;&gt;&gt; a.corrcoef() &gt;&gt;&gt; np.std(b)</pre> | <p>Array-wise sum</p> <p>Array-wise minimum value</p> <p>Maximum value of an array row</p> <p>Cumulative sum of the elements</p> <p>Mean</p> <p>Median</p> <p>Correlation coefficient</p> <p>Standard deviation</p> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## Практична робота №5

**Мета роботи:** ознайомитися та засвоїти роботу основних операцій з масивами у бібліотеці **NumPy**.

### Порядок виконання роботи

1. Створіть новий файл. Підключіть бібліотеку **NumPy**.
2. Виконайте наступні завдання:
  - 2.1. Напишіть програму **NumPy** для поелементного додавання, віднімання, множення та ділення аргументів.  
Очікуваний результат:  
Додавання: 5.0  
Віднімання: -3,0  
Множення: 4.0  
Ділення: 0,25
  - 2.2. Напишіть програму **NumPy** для обчислення  $\exp(x) - 1$  для всіх елементів заданого масиву. Опишіть отриманий результат. Вхідний масив: `[1. 2. 3. 4.]`
  - 2.3. Розглядаючи чотиривимірний масив, як отримати суму по двох останніх осях одночасно. Опишіть отриманий результат. Вхідні дані: `A = np.random.randint(0,10,(3,4,3,4))`
  - 2.4. Відніміть середнє значення кожного рядка матриці. Опишіть отриманий результат  
Вхідні дані: `X = np.random.rand(5, 10)`
3. Оформіть звіт

## Pandas: Об'єкт Series

**Pandas** — програмна бібліотека, написана для мови програмування **Python** для маніпулювання даними та їхнього аналізу [2]. Вона, зокрема, пропонує структури даних та операції для маніпулювання чисельними таблицями та часовими рядами.

**Pandas Series** схожа на стовпець у таблиці. Це одновимірний масив, що містить дані будь-якого типу.

```
pandas.Series(data=None, index=None, dtype=None, name=None,
copy=False, fastpath=False)
```

Параметри функції **Series()**:

- **data(array-like, Iterable, dict, or scalar value)** - Містить дані, що зберігаються в серії. Якщо дані є dict, порядок аргументів зберігається.
- **index(array-like or Index (1d))** - Значення мають бути хешованими та мати таку саму довжину, що й дані. Допускаються неунікальні значення індексу. За замовчуванням буде RangeIndex (0, 1, 2, ..., n), якщо не вказано. Якщо дані є dict-подібними, а індекс має значення None, тоді ключі в даних використовуються як індекс. Якщо індекс не має значення «Немає», отримана серія повторно індексується зі значеннями індексу.
- **dtype(str, numpy.dtype, or ExtensionDtype, optional)** - Тип даних для вихідної серії. Якщо не вказано, це буде зроблено з даних.
- **name(str, optional)** - Назва для серії.
- **copy(bool, default False)** - Копіювати вхідні дані. Впливає лише на ряд або 1d ndarray.

Подивіться приклади.

## Python Dictionaries

Словники використовуються для зберігання значень даних у парах *ключ:значення*.

Словник — це збірка, яка є впорядкованою, змінюваною та не допускає дублікатів.

Словники пишуться у фігурних дужках і мають ключі та значення:

```
thisdict = { "brand": "Ford",
"model": "Mustang", "year": 1964
}
```

## Приклад

Створення **Panadas Series** за допомогою **Python Dictionary**

```
import pandas as pd
dictionary = {'A': 10, 'B': 20, 'C': 30} series = pd.Series(dictionary) print(series)
# output A 10
в 20
с 30
dtype: int64
```

Створення **Panadas Series** за допомогою **NumPy**

```
import pandas as pd
np_array = np.array([10, 20, 30, 40, 50]) print("NumPy array:")
print(np_array)
new_series = pd.Series(np_array) print("Converted Pandas series:")
print(new_series)
# output NumPy array:
[10 20 30 40 50]
Converted Pandas series:
0 10
1 20
2 30
3 40
4 50
dtype: int64
```

Створення даних з типом **object**.

```
import pandas as pd
sr = pd.Series(['New York', 'Chicago', 'Toronto', 'Lisbon']) sr.index = ['City 1', 'City 2', 'City 3', 'City 4']
print(sr) # output
City 1    New York
City 2    Chicago
City 3    Toronto
City 4    Lisbon
dtype: object
```

## Методи з **Panadas Series**

Тепер ми використаємо атрибут **Series.axes**, щоб повернути список міток осей рядків для заданого об'єкта **Series**.

`sr.axes`

`# output`

```
[Index(['City 1', 'City 2', 'City 3', 'City 4'], dtype='object')]
```

У цьому прикладі 5 верхніх рядків кадру даних повертаються та зберігаються в новій змінній. Жоден параметр не передається методу `.head()`, оскільки за умовчанням він дорівнює 5.

```
import pandas as pd
```

```
data = pd.read_csv("nba.csv") data_top = data.head() data_top
```

`# output`

|   | Name          | Team           | Number | Position | Age  | Height | Weight | College           | Salary    |
|---|---------------|----------------|--------|----------|------|--------|--------|-------------------|-----------|
| 0 | Avery Bradley | Boston Celtics | 0.0    | PG       | 25.0 | 6-2    | 180.0  | Texas             | 7730337.0 |
| 1 | Jae Crowder   | Boston Celtics | 99.0   | SF       | 25.0 | 6-6    | 235.0  | Marquette         | 6796117.0 |
| 2 | John Holland  | Boston Celtics | 30.0   | SG       | 27.0 | 6-5    | 205.0  | Boston University | NaN       |
| 3 | R.J. Hunter   | Boston Celtics | 28.0   | SG       | 22.0 | 6-5    | 185.0  | Georgia State     | 1148640.0 |
| 4 | Jonas Jerebko | Boston Celtics | 8.0    | PF       | 29.0 | 6-10   | 231.0  | NaN               | 5000000.0 |

У цьому прикладі 5 нижніх рядків кадру даних повертаються та зберігаються в новій змінній. Жоден параметр не передається методу `.tail()`, оскільки за умовчанням він дорівнює 5.

```
data_bottom = data.tail() data_bottom
```

`# output`

|     | Name         | Team      | Number | Position | Age  | Height | Weight | College | Salary    |
|-----|--------------|-----------|--------|----------|------|--------|--------|---------|-----------|
| 453 | Shelvin Mack | Utah Jazz | 8.0    | PG       | 26.0 | 6-3    | 203.0  | Butler  | 2433333.0 |
| 454 | Raul Neto    | Utah Jazz | 25.0   | PG       | 24.0 | 6-1    | 179.0  | NaN     | 900000.0  |
| 455 | Tibor Pleiss | Utah Jazz | 21.0   | C        | 26.0 | 7-3    | 256.0  | NaN     | 2900000.0 |
| 456 | Jeff Withey  | Utah Jazz | 24.0   | C        | 26.0 | 7-0    | 231.0  | Kansas  | 947276.0  |
| 457 | NaN          | NaN       | NaN    | NaN      | NaN  | NaN    | NaN    | NaN     | NaN       |

Використовуйте метод `Series.sort_index()`, щоб відсортувати мітки індексів даного об'єкта серії.

```
City 5    New York
City 6    Chicago
City 4    Toronto
City 2    Lisbon
City 3     Rio
City 1    Moscow
dtype: object
```

```
City 1    Moscow
City 2    Lisbon
City 3     Rio
City 4    Toronto
City 5    New York
City 6    Chicago
dtype: object
```

Вхідний масив Отриманий результат після сортування  
Використовуйте функцію **Series.sort\_values()**, щоб відсортувати елементи даного об'єкта серії в лексикографічному порядку.

```
2014-08-03 10:00:00+02:00    New York
2014-08-10 10:00:00+02:00    Chicago
2014-08-17 10:00:00+02:00    Toronto
2014-08-24 10:00:00+02:00    Lisbon
2014-08-31 10:00:00+02:00     Rio
2014-09-07 10:00:00+02:00    Moscow
Freq: W-SUN, dtype: object
```

```
2014-08-10 10:00:00+02:00    Chicago
2014-08-24 10:00:00+02:00    Lisbon
2014-09-07 10:00:00+02:00    Moscow
2014-08-03 10:00:00+02:00    New York
2014-08-31 10:00:00+02:00     Rio
2014-08-17 10:00:00+02:00    Toronto
dtype: object
```

Вхідний масив Отриманий результат після сортування  
Тепер ми використаємо функцію **Series.isin()**, щоб перевірити, чи містяться передані значення в об'єкті серії.

```
import pandas as pd
sr = pd.Series([10, 25, 3, 25, 24, 6])
index_ = ['Coca Cola', 'Sprite', 'Coke', 'Fanta', 'Dew', 'ThumbsUp'] sr.index = index_
print(sr)
```

```
Coca Cola    10
Sprite       25
Coke         3
Fanta        25
Dew          24
ThumbsUp     6
dtype: int64
```

```
result = sr.isin([25])print(result)
```

```
Coca Cola    False
Sprite       True
Coke         False
Fanta        True
Dew          False
ThumbsUp     False
dtype: bool
```

Тепер ми використаємо атрибут **Series.values**, щоб повернути значення даного об'єкта **Series** як **ndarray**.

```
import pandas as pd
sr = pd.Series(['New York', 'Chicago', 'Toronto', 'Lisbon', 'Rio'])
```

```
sr.index = ['City 1', 'City 2', 'City 3', 'City 4', 'City 5']
print(sr)
```

---

```
City 1    New York
City 2    Chicago
City 3    Toronto
City 4    Lisbon
City 5     Rio
dtype: object
sr.values
```

---

```
array(['New York', 'Chicago', 'Toronto', 'Lisbon', 'Rio'], dtype=object)
```

## Практична робота №6

**Мета роботи:** приєднати та ознайомитись з бібліотекою **Pandas**.

### Порядок виконання роботи

1. Створіть новий файл. Підключіть бібліотеку **Pandas**.
2. Виконайте наступні завдання:
  - 2.1. Напишіть програму **Pandas** для створення та відображення одновимірного масиву, що містить масив даних.
  - 2.2. Напишіть програму **Pandas** для перетворення **Dictionary** на серію **Pandas**.  
Sample dictionary: d1 = {'a': 100, 'b': 200, 'c':300, 'd':400, 'e':800}
  - 2.3. Напишіть програму **Pandas** для перетворення масиву **NumPy** на серію **Pandas**.  
Sample NumPy array: d1 = [10, 20, 30, 40, 50]
  - 2.4. Напишіть програму **Pandas** для сортування даної серії.  
Sample NumPy array: d1 = ['100', '200', 'python', '300.12', '400']
  - 2.5. Напишіть програму **Pandas**, щоб отримати елементи даної серії, яких немає в іншій даній серії.  
sr1 = pd.Series([1, 2, 3, 4, 5])  
sr2 = pd.Series([2, 4, 6, 8, 10])
  - 2.6. Напишіть програму **Pandas**, щоб отримати перших 10 елементів даної серії  
ser = pd.Series(list('abcdefghijklmnopqrstuvwxyz'))
  - 2.7. Напишіть програму **Pandas**, щоб отримати останні 3 елементи даної серії  
ser = pd.Series(list('abcdefghijklmnopqrstuvwxyz'))
3. Оформіть звіт

## Pandas: Об'єкт DataFrame та операції вибору елементів

**DataFrame** - Двовимірні табличні дані зі змінним розміром, потенційно різнорідні. Структура даних також містить позначені осі (рядки та стовпці). Арифметичні операції вирівнюються на мітках рядків і стовпців. Можна розглядати як dict-подібний контейнер для об'єктів **Series**. Основна структура даних pandas.

`pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=None)`

Параметри функції **DataFrame()**:

- `data(ndarray (structured or homogeneous), Iterable, dict, or DataFrame)`
  - Dict може містити ряди, масиви, константи, класи даних або спискові об'єкти. Якщо дані є dict, порядок стовпців відповідає порядку вставки. Якщо dict містить ряди, які мають визначений індекс, він вирівнюється за його індексом. Це вирівнювання також відбувається, якщо дані є серією або самим **DataFrame**. Вирівнювання виконується на входах **Series/DataFrame**. Якщо дані є списком dicts, порядок стовпців відповідає порядку вставки.
- `index(Index or array-like)` - Індекс для кінцевого кадру. За замовчуванням буде використано **RangeIndex**, якщо у вхідних даних немає інформації про індексування та не надано індекс.
- `columns(Index or array-like)` - Мітки стовпців для результуючого кадру, якщо їх немає в даних, за замовчуванням **RangeIndex(0, 1, 2, ..., n)**. Якщо дані містять мітки стовпців, натомість буде виконано вибір стовпців.
- `dtype(dtype, default None)` - Примусовий тип даних. Дозволяється лише один dtype. `copy(bool or None, default None)` - Копіювати дані з вхідних даних. Для даних dict стандартне значення None поводить як `copy=True`. Для вхідних даних **DataFrame** або 2d ndarray значення за умовчанням None поводить як `copy=False`. Якщо дані є dict, що містить одну або більше серій (можливо, різних dtypes), `copy=False` гарантує, що ці вхідні дані не копіюються.

### DataFrame.loc

Доступ до групи рядків і стовпців за мітками або логічним масивом.

`.loc[]` переважно базується на мітках, але також може використовуватися з логічним масивом.

Дозволені вхідні дані:

- Одна мітка, напр. 5 або «a» (зверніть увагу, що 5 інтерпретується як мітка індексу, а не як ціла позиція в індексі).
- Список або масив міток, напр. ['a', 'b', 'c'].
- Об'єкт фрагмента з мітками, напр. 'a':'f'.
- Логічний масив такої ж довжини, що й вісь, яку розрізають, наприклад. [True, False, True].
- Логічний ряд, який можна вирівняти. Індекс ключа буде вирівняно передмаскуванням.
- Вирівнюваний індекс. Індекс повернутого вибору буде вхідним.
- Callable функція з одним аргументом (серія виклику або **DataFrame**), яка повертає дійсний вихід для індексування (один із наведених вище)

### DataFrame.iloc

Індексація на основі цілого розташування для вибору за позицією. **.iloc[]** переважно базується на позиції цілого числа (від 0 до довжини-1 осі), але також може використовуватися з булевим масивом.

Дозволені вхідні дані:

- Ціле число, напр. 5.
- Список або масив цілих чисел, напр. [4, 3, 0].
- Об'єкт фрагмента з ints, напр. 1:7.

Логічний масив.

Callable функція з одним аргументом (серія виклику або **DataFrame**), яка повертає дійсний вихід для індексування (один із наведених вище). Це корисно в ланцюжках методів, коли у вас немає посилання на об'єкт, що викликає, але ви бажаєте базувати свій вибір на якомусь значенні.

Набір індексів рядків і стовпців. Елементи набору складаються з одного з наведених вище входів, наприклад. (0, 1).

**.iloc** викличе **IndexError**, якщо запитуваний індексатор знаходиться поза межами, за винятком індексаторів фрагментів, які дозволяють індексувати поза межами (це відповідає семантиці фрагментів **python/numpy**).

### Приклад

Створення **Panadas DataFrame**.

```
import pandas as pd
lst = ['Geeks', 'For', 'Geeks', 'is',
      'portal', 'for', 'Geeks']
```

```
df = pd.DataFrame(lst)
print(df)
```

| 0 |        |
|---|--------|
| 0 | Geeks  |
| 1 | For    |
| 2 | Geeks  |
| 3 | is     |
| 4 | portal |
| 5 | for    |
| 6 | Geeks  |

Створення **Panadas DataFrame** використовуючи **Dictionaries**.

```
import pandas as pd
dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],
        'degree': ["MBA", "BCA", "M.Tech", "MBA"], 'score': [90,
        40, 80, 98]}
df = pd.DataFrame(dict)
print(df)
```

|   | name   | degree | score |
|---|--------|--------|-------|
| 0 | aparna | MBA    | 90    |
| 1 | pankaj | BCA    | 40    |
| 2 | sudhir | M.Tech | 80    |
| 3 | Geeku  | MBA    | 98    |

Створення **Panadas DataFrame** використовуючи **NumPy**.

```
import pandas as pd
import numpy as np
array = np.array([[1, 1, 1], [2, 4, 8], [3, 9, 27],
                  [4, 16, 64], [5, 25, 125], [6, 36, 216],
                  [7, 49, 343]])
index_values = ['first', 'second', 'third',
                'fourth', 'fifth', 'sixth', 'seventh']
column_values = ['number', 'squares', 'cubes']
df = pd.DataFrame(data = array,
                  index = index_values,
                  columns = column_values)
print(df)
```

|         | number | squares | cubes |
|---------|--------|---------|-------|
| first   | 1      | 1       | 1     |
| second  | 2      | 4       | 8     |
| third   | 3      | 9       | 27    |
| fourth  | 4      | 16      | 64    |
| fifth   | 5      | 25      | 125   |
| sixth   | 6      | 36      | 216   |
| seventh | 7      | 49      | 343   |

Отримання даних з фрейму за певними критеріями використовуючи функцію **loc**

Вхідний **dataFrame**

|   | Brand    | Year | Kms Driven | City      | Mileage |
|---|----------|------|------------|-----------|---------|
| 0 | Maruti   | 2012 | 50000      | Gurgaon   | 28      |
| 1 | Hyundai  | 2014 | 30000      | Delhi     | 27      |
| 2 | Tata     | 2011 | 60000      | Mumbai    | 25      |
| 3 | Mahindra | 2015 | 25000      | Delhi     | 26      |
| 4 | Maruti   | 2012 | 10000      | Mumbai    | 28      |
| 5 | Hyundai  | 2016 | 46000      | Delhi     | 29      |
| 6 | Renault  | 2014 | 31000      | Mumbai    | 24      |
| 7 | Tata     | 2018 | 15000      | Chennai   | 21      |
| 8 | Maruti   | 2019 | 12000      | Ghaziabad | 24      |

`display(data.loc[(data.Brand == 'Maruti') & amp(data.Mileage & gt 25)])`

|   | Brand  | Year | Kms Driven | City    | Mileage |
|---|--------|------|------------|---------|---------|
| 0 | Maruti | 2012 | 50000      | Gurgaon | 28      |
| 4 | Maruti | 2012 | 10000      | Mumbai  | 28      |

Отримання даних з фрейму за індексом використовуючи функцію **iloc**

`display(data.iloc[[0, 2, 4, 7]])`

|   | Brand  | Year | Kms Driven | City    | Mileage |
|---|--------|------|------------|---------|---------|
| 0 | Maruti | 2012 | 50000      | Gurgaon | 22      |
| 2 | Tata   | 2011 | 60000      | Mumbai  | 22      |
| 4 | Maruti | 2012 | 10000      | Mumbai  | 22      |
| 7 | Tata   | 2018 | 15000      | Chennai | 21      |

## Практична робота №7

**Мета роботи:** ознайомитись з об'єктом **DataFrame** та його функціями по вибору елементів.

### Порядок виконання роботи

1. Створіть новий файл. Підключіть бібліотеку **Pandas**.
2. Виконайте наступні завдання:

Напишіть програму **Pandas**, щоб створити фрейм даних з даними студентів вашої групи починаючи з трьох студентів

перед вами та чотирьох після вас. Індекс колонки зазначати як "ПІБ", а індекс рядка використовувати порядковий номер студента.

- 2.1. Напишіть програму **Pandas**, щоб створити фрейм даних зі словниката відобразити його.  
`Sample dictionary: {'X':[78,85,96,80,86], 'Y':[84,94,89,83,86], 'Z':[86,97,96,72,83]}`
- 2.2. Напишіть програму **Pandas** для створення та відображення **DataFrame** із заданих даних словника, які мають мітки індексу. `Sample DataFrame: exam_data = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'], 'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19], 'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1], 'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']}`  
`labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']`
- 2.3. Напишіть програму **Pandas** для вибору стовпців «name» та «score» із попереднього **DataFrame**.
- 2.4. Напишіть програму **Pandas** для вибору рядків з попереднього **DataFrame**, у яких кількість спроб перевірки перевищує 2.
- 2.5. Напишіть програму **Pandas**, щоб змінити оцінку в рядку «d» на 11,5 у попередньому **DataFrame**.
- 2.6. Напишіть програму **Pandas** для отримання нового фрейму з значенням оцінки менше 10. Використовуйте попередній фрейм
- 2.7. Напишіть програму **Pandas**, щоб отримати дані з колонки відвідування для рядків від **b** до **h**.
- 2.8. Напишіть програму **Pandas**, щоб отримати дані по індексу **3** колонки та індексів рядків з **1** по **4**.

3. Оформіть звіт

## Pandas: Структурні операції та мультиіндексні об'єкти DataFrame

**Pandas.isnull()** ("DataFrame Name") or **DataFrame.isnull()**

**Pandas.notnull()** ("DataFrame Name") or **DataFrame.notnull()**

Методи **Pandas isnull()** і **notnull()** використовуються для перевірки та керування значеннями NULL у кадрі даних.

**DataFrame.dropna(\*, axis=0, how=\_NoDefault.no\_default, thresh=\_NoDefault.no\_default, subset=None, inplace=False, ignore\_index=False)**

Параметри функції **dropna()**:

**axis** ({0 or 'index', 1 or 'columns'}, default 0) - Визначте, чи видалено рядки або стовпці, які містять відсутні значення.

- 0 або «індекс»: видалення рядків, які містять відсутні значення.
- 1, або «стовпці»: видалить стовпці, які містять відсутнє значення.

Передайте кортеж або список для розміщення на кількох осях. Допускається лише одна вісь.

**how** ({'any', 'all'}, default 'any') - Визначте, видалено рядок або стовпець з **DataFrame**, якщо ми маємо принаймні одну NA або всі NA.

- 'any': якщо наявні будь-які значення NA, видалить цей рядок або стовпець.
- 'all': якщо всі значення є NA, видалить цей рядок або стовпець.

**thresh**(int, optional) - Вимагайте стільки значень, які не є NA. Не можна поєднувати з як.

**subset**(column label or sequence of labels, optional) - Мітки вздовж іншої осі, яку слід враховувати, напр. якщо ви відкидаєте рядки, це буде список стовпців для включення.

**inplace**(bool, default False) - Чи змінювати **DataFrame** замість створення нового. **ignore\_index**(bool, default) - Якщо значення True, отримана вісь буде позначена 0, 1, ..., n - 1.

**DataFrame.fillna(value=None, \*, method=None, axis=None, inplace=False, limit=None, downcast=None)**

Параметри функції **fillna()**:

**value**(scalar, dict, Series, or DataFrame) - Значення, яке потрібно використовувати для заповнення дірок (наприклад, 0), по черзі dict/Series/DataFrame значень, які вказують, яке значення використовувати для кожного індексу (для **Series**) або стовпця (для **DataFrame**). Значення не в dict/Series/DataFrame не будуть заповнені. Це значення не може бути списком.

**method** ({'backfill', 'bfill', 'ffill', None}, default None) - Спосіб

використання для заповнення отворів у переіндексованих серіях:

- ffill: передавати останнє дійсне спостереження до наступного

дійсного.

- `backfill / bfill`: використовуйте наступне дійсне спостереження, щоб заповнити прогалину.

`axis({0 or 'index', 1 or 'columns'})` - Вісь, уздовж якої потрібно заповнити відсутні значення. Для серії цей параметр не використовується і за умовчанням дорівнює 0.

`inplace(bool, default False)` - Якщо True, заповнить на місці. Примітка: це призведе до зміни будь-яких інших переглядів цього об'єкта (наприклад, зріз без копіювання для стовпця в **DataFrame**).

`limit(int, default None)` - Якщо вказано метод, це максимальна кількість послідовних значень NaN для заповнення вперед/назад. Іншими словами, якщо є проміжок із більшою кількістю послідовних NaN, він буде заповнений лише частково. Якщо метод не вказано, це максимальна кількість записів уздовж всієї осі, де будуть заповнені NaN. Має бути більше 0, якщо немає.

`downcast(dict, default is None)` - Команда `item->dtype` того, що потрібно перетворити, якщо це можливо, або рядок «infer», який спробує привести до відповідного рівного типу (наприклад, `float64` до `int64`, якщо це можливо).

**DataFrame.reset\_index(level=None, \*, drop=False, inplace=False, col\_level=0, col\_fill="", allow\_duplicates=\_NoDefault.no\_default, names=None)**

Параметри функції `reset_index()`:

- `level(int, str, tuple, or list, default None)` - З індексу видаляйте лише вказані рівні. За замовчуванням видаляє всі рівні.
- `drop(bool, default False)` - Не намагайтеся вставити індекс у стовпці **DataFrame**. Це скидає індекс до стандартного цілого індексу.
- `inplace(bool, default False)` - Чи змінювати **DataFrame** замість створення нового.
- `col_level(int or str, default 0)` - Якщо стовпці мають кілька рівнів, визначає, на який рівень вставляються мітки. За замовчуванням він вставляється на перший рівень.
- `col_fill(object, default)` - Якщо стовпці мають кілька рівнів, визначає, як називаються інші рівні. Якщо немає, тоді ім'я індексу повторюється.
- `allow_duplicates(bool, optional, default lib.no_default)` - Дозволити створення дублікатів міток стовпців.
- `names(int, str or 1-dimensional list, default None)` - Використовуючи вказаний рядок, перейменуйте стовпець **DataFrame**, який містить дані індексу. Якщо **DataFrame** має **MultiIndex**, це має бути список або кортеж, довжина якого дорівнює кількості рівнів.

**MultiIndex.from\_tuples(tuples, sortorder=None, names=None)**

Параметри функції **from\_tuples()**:

- **tuples(list / sequence of tuple-likes)** - Кожен кортеж є індексом одного рядка/стовпця. **sortorder(int or None)** - Рівень упорядкованості (повинен бути відсортований лексикографічно за цим рівнем).
- **names(list / sequence of str, optional)** - Назви рівнів в індексі.

**DataFrame.xs(key, axis=0, level=None, drop\_level=True)**

Параметри функції **xs()**:

- **key(label or tuple of label)** - Мітка міститься в індексі або частково в **MultiIndex**. **axis({0 or 'index', 1 or 'columns'}, default 0)** - Вісь для отримання поперечного перерізу.
- **level(object, defaults to first n levels (n=1 or len(key)))** - Якщо ключ частково міститься в **MultiIndex**, укажіть, які рівні використовуються. Рівні можна посилатися за міткою або позицією.
- **drop\_level(bool, default True)** - Якщо False, повертає об'єкт із тим самим рівнем, що й self.

## Приклад

Приклад використання методу **isnull()**

```
import pandas as pd
```

```
data = pd.read_csv("employees.csv")
```

```
bool_series = pd.isnull(data["Team"])data[bool_series]
```

```
bool_series = pd.notnull(data["Gender"])
```

|     | First Name  | Gender | Start Date | Last Login Time | Salary | Bonus % | Senior Management | Team |
|-----|-------------|--------|------------|-----------------|--------|---------|-------------------|------|
| 1   | Thomas      | Male   | 3/31/1996  | 6:53 AM         | 61933  | 4.170   | True              | NaN  |
| 10  | Louise      | Female | 8/12/1980  | 9:01 AM         | 63241  | 15.132  | True              | NaN  |
| 23  | NaN         | Male   | 6/14/2012  | 4:19 PM         | 125792 | 5.042   | NaN               | NaN  |
| 32  | NaN         | Male   | 8/21/1998  | 2:27 PM         | 122340 | 6.417   | NaN               | NaN  |
| 91  | James       | NaN    | 1/26/2005  | 11:00 PM        | 128771 | 8.309   | False             | NaN  |
| 109 | Christopher | Male   | 4/22/2000  | 10:15 AM        | 37919  | 11.449  | False             | NaN  |
| 139 | NaN         | Female | 10/3/1990  | 1:08 AM         | 132373 | 10.527  | NaN               | NaN  |
| 199 | Jonathan    | Male   | 7/17/2009  | 8:15 AM         | 130581 | 16.736  | True              | NaN  |
| 258 | Michael     | Male   | 1/24/2002  | 3:04 AM         | 43586  | 12.659  | False             | NaN  |
| 290 | Jeremy      | Male   | 6/14/1988  | 6:20 PM         | 129460 | 13.657  | True              | NaN  |

|    | First Name | Gender | Start Date | Last Login Time | Salary | Bonus % | Senior Management | Team                 |
|----|------------|--------|------------|-----------------|--------|---------|-------------------|----------------------|
| 0  | Douglas    | Male   | 8/6/1993   | 12:42 PM        | 97308  | 6.945   | True              | Marketing            |
| 1  | Thomas     | Male   | 3/31/1996  | 6:53 AM         | 61933  | 4.170   | True              | NaN                  |
| 2  | Maria      | Female | 4/23/1993  | 11:17 AM        | 130590 | 11.858  | False             | Finance              |
| 3  | Jerry      | Male   | 3/4/2005   | 1:00 PM         | 138705 | 9.340   | True              | Finance              |
| 4  | Larry      | Male   | 1/24/1998  | 4:47 PM         | 101004 | 1.389   | True              | Client Services      |
| 5  | Dennis     | Male   | 4/18/1987  | 1:35 AM         | 115163 | 10.125  | False             | Legal                |
| 6  | Ruby       | Female | 8/17/1987  | 4:20 PM         | 65476  | 10.012  | True              | Product              |
| 7  | NaN        | Female | 7/20/2015  | 10:43 AM        | 45906  | 11.598  | NaN               | Finance              |
| 8  | Angela     | Female | 11/22/2005 | 6:29 AM         | 95570  | 18.523  | True              | Engineering          |
| 9  | Frances    | Female | 8/8/2002   | 6:51 AM         | 139852 | 7.524   | True              | Business Development |
| 10 | Louise     | Female | 8/12/1980  | 9:01 AM         | 63241  | 15.132  | True              | NaN                  |
| 11 | Julie      | Female | 10/26/1997 | 3:19 PM         | 102508 | 12.637  | True              | Legal                |
| 12 | Brandon    | Male   | 12/1/1980  | 1:08 AM         | 112807 | 17.492  | True              | Human Resources      |
| 13 | Gary       | Male   | 1/27/2008  | 11:40 PM        | 109831 | 5.831   | False             | Sales                |
| 14 | Kimberly   | Female | 1/14/1999  | 7:13 AM         | 41426  | 14.543  | True              | Finance              |
| 15 | Lillian    | Female | 6/5/2016   | 6:09 AM         | 59414  | 1.256   | False             | Product              |
| 16 | Jeremy     | Male   | 9/21/2010  | 5:56 AM         | 90370  | 7.369   | False             | Human Resources      |

### Джерела для отримання тестових даних

CSV-файл із даними про гравців NBA, що містить інформацію про імена, команди, позиції, зарплати, вік тощо. Наприклад, [NBA.csv](https://media.geeksforgeeks.org/wp-content/uploads/nba.csv)

#### Як використати:

```
import pandas as pd
```

```
# Завантаження даних
```

```
url = "https://media.geeksforgeeks.org/wp-content/uploads/nba.csv"
```

```
data = pd.read_csv(url)
```

```
# Перегляд перших рядків
```

```
print(data.head())
```

CSV-файл із даними про співробітників, що містить поля, як-от ім'я, посада, зарплата, відділ тощо. Можна використовувати готові набори даних для тестування, наприклад [Employees.csv](https://www.codingexplorations.com/s/employees.csv).

#### Як використати:

```
import pandas as pd
```

```
# Завантаження даних
```

```
url = "https://www.codingexplorations.com/s/employees.csv"
```

```
data = pd.read_csv(url)
```

```
# Вивід інформації про дані
```

```
print(data.info())
```

### Інші джерела тестових даних:

- [Kaggle](#) - платформа для завантаження великих і якісних наборів даних для навчання і тестування.
- [Data.gov](#) - набори даних для аналізу у сфері бізнесу, економіки, здоров'я тощо.
- [Mockaroo](#) - інструмент для генерації тестових даних за заданими параметрами.
- [Open Data Portal](#) - колекція відкритих наборів даних для аналізу та тестування.

### Рекомендації:

1. Перед завантаженням даних перевірте формат файлу, щоб переконатися, що він відповідає вашим вимогам.
2. Використовуйте бібліотеки, як-от pandas чи numpy, для обробки даних.
3. Якщо дані потрібно зберегти локально, можна завантажити їх і зберегти на диск:

```
import requests
```

```
url = "https://media.geeksforgeeks.org/wp-content/uploads/nba.csv"  
response = requests.get(url)
```

```
# Збереження файлу  
with open("nba.csv", "wb") as file:  
    file.write(response.content)  
print("Файл завантажено!")
```

Приклад використання методу `fillna()`

```
import pandas as pd  
data = pd.read_csv("nba.csv")  
new_data = data.dropna(axis=0,  
how='any')print("Old data frame  
length:", len(data),  
"\nNew data frame  
length:",  
len(new_data),  
"\nNumber of rows with at least 1  
NA value: ",(len(data)-  
len(new_data)))
```

#Output

Old data frame length:  
458 New data frame  
length: 364  
Number of rows with at least 1 NA value: 94

```
import pandas as pd
import numpy as np
dataframe = pd.DataFrame({'Count': [1, np.nan, np.nan, 4, 2,
                                   np.nan, np.nan, 5, 6],
                          'Name': ['Geeks', 'for',
                                   'Geeks', 'a', 'portal', 'for',
                                   'computer', 'Science', 'Geeks'],
                          'Category': list('ppqqrss')})
constant_values = {'Count': 10}
dataframe = dataframe.fillna(value =
constant_values)
display(dataframe)
```

|   | Count | Name     | Category |
|---|-------|----------|----------|
| 0 | 1.0   | Geeks    | p        |
| 1 | 10.0  | for      | p        |
| 2 | 10.0  | Geeks    | q        |
| 3 | 4.0   | a        | q        |
| 4 | 2.0   | portal   | r        |
| 5 | 10.0  | for      | r        |
| 6 | 10.0  | computer | s        |
| 7 | 5.0   | Science  | s        |
| 8 | 6.0   | Geeks    | s        |

Приклад використання методу

```
reset_index()
import pandas as pd
data = pd.read_csv("employees.csv")
data.set_index(["First Name"], inplace
= True,
               append = True, drop =
True)
data.reset_index(inplace
= True)
data.head()
```

Перед використанням методу `reset_index()`

|   |            | Gender | Start Date | Last Login Time | Salary | Bonus % | Senior Management | Team            |
|---|------------|--------|------------|-----------------|--------|---------|-------------------|-----------------|
|   | First Name |        |            |                 |        |         |                   |                 |
| 0 | Douglas    | Male   | 8/6/1993   | 12:42 PM        | 97308  | 6.945   | True              | Marketing       |
| 1 | Thomas     | Male   | 3/31/1996  | 6:53 AM         | 61933  | 4.170   | True              | NaN             |
| 2 | Maria      | Female | 4/23/1993  | 11:17 AM        | 130590 | 11.858  | False             | Finance         |
| 3 | Jerry      | Male   | 3/4/2005   | 1:00 PM         | 138705 | 9.340   | True              | Finance         |
| 4 | Larry      | Male   | 1/24/1998  | 4:47 PM         | 101004 | 1.389   | True              | Client Services |

Після використання методу `reset_index()`

|   | level_0 | First Name | Gender | Start Date | Last Login Time | Salary | Bonus % | Senior Management | Team            |
|---|---------|------------|--------|------------|-----------------|--------|---------|-------------------|-----------------|
| 0 | 0       | Douglas    | Male   | 8/6/1993   | 12:42 PM        | 97308  | 6.945   | True              | Marketing       |
| 1 | 1       | Thomas     | Male   | 3/31/1996  | 6:53 AM         | 61933  | 4.170   | True              | NaN             |
| 2 | 2       | Maria      | Female | 4/23/1993  | 11:17 AM        | 130590 | 11.858  | False             | Finance         |
| 3 | 3       | Jerry      | Male   | 3/4/2005   | 1:00 PM         | 138705 | 9.340   | True              | Finance         |
| 4 | 4       | Larry      | Male   | 1/24/1998  | 4:47 PM         | 101004 | 1.389   | True              | Client Services |

Перед використанням методу

```
from _tuples()import pandas as pd
tuples=[(30, 'Larry'), (20, 'Mike'),
        (18, 'David'), (25, 'Tim')]
print(tuples)
[(30, 'Larry'), (20, 'Mike'), (18, 'David'), (25, 'Tim')]
```

```
midx = pd.MultiIndex.from_tuples(tuples, names=('Age',
'Name'))print(midx)
MultiIndex(levels=[[18, 20, 25, 30], ['David', 'Larry', 'Mike', 'Tim']],
            labels=[[3, 1, 0, 2], [1, 2, 0, 3]],
            names=['Age', 'Name'])
```

Перед використанням методу `xs()`

```
import pandas as pd
animal_dict = {'num_of_legs': [4, 0, 4, 2, 2, 2],
               'num_of_wings': [0, 0, 0, 2, 2, 2],
               'class': ['Reptiles', 'Reptiles', 'Reptiles',
                        'Birds', 'Birds', 'Birds'],
               'animal': ['Turtle', 'Snake', 'Crocodile',
                        'Parrot', 'Owl',
                        'Hummingbird'], 'locomotion': ['swim_walk',
               'swim_crawl', 'swim_walk',
               'flies', 'flies', 'flies']}
df = pd.DataFrame(data=animal_dict)
```

```
df = df.set_index(['class', 'animal',
'locomotion'])
df
```

|          |             |            | num_of_legs | num_of_wings |
|----------|-------------|------------|-------------|--------------|
| class    | animal      | locomotion |             |              |
| Reptiles | Turtle      | swim_walk  | 4           | 0            |
|          | Snake       | swim_crawl | 0           | 0            |
|          | Crocodile   | swim_walk  | 4           | 0            |
| Birds    | Parrot      | flies      | 2           | 2            |
|          | Owl         | flies      | 2           | 2            |
|          | Hummingbird | flies      | 2           | 2            |

```
df.xs('Reptiles')
```

|  |           |            | num_of_legs | num_of_wings |
|--|-----------|------------|-------------|--------------|
|  | animal    | locomotion |             |              |
|  | Turtle    | swim_walk  | 4           | 0            |
|  | Snake     | swim_crawl | 0           | 0            |
|  | Crocodile | swim_walk  | 4           | 0            |

## Практична робота №8

**Мета роботи:** ознайомитись та отримати навички роботи з структурними операціями та мультиіндексними об'єктами у **DataFrame**

### Порядок виконання роботи

- Створіть новий файл. Підключіть бібліотеку **Pandas**.
- Виконайте наступні завдання:
  - Напишіть програму **Pandas** для створення та відображення **DataFrame** із заданих даних словника, які мають мітки індексу. **Sample DataFrame:** `exam_data = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'], 'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19], 'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1], 'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']} labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']`
  - Напишіть програму **Pandas**, щоб вставити новий стовпець у `exam_data DataFrame`.
  - Напишіть програму **Pandas**, щоб вставити новий стовпець з довільними значеннями отриманих за допомогою бібліотеки **NumPy** у `exam_data DataFrame`.
  - Напишіть програму **Pandas**, щоб видалити список колонок із

`exam_data` **DataFrame**.

- 2.5. Напишіть програму **Pandas**, щоб видалити список рядків із `exam_data` **DataFrame**.
- 2.6. Напишіть програму **Pandas** для скидання індексу в `exam_data`. Але перед тим як скинути індекс, спершу видаліть перший і другий рядки.
- 2.7. Напишіть програму **Pandas** для того щоб перевірити, чи немає у `exam_data` **DataFrame** відсутніх значень?
- 2.8. Напишіть програму **Pandas** для того щоб замінити пропущені значення декількох числових стовпців на 0?
- 2.9. Напишіть програму **Pandas** для створення **DataFrame**, використовуючи рівні **MultiIndex** як стовпець та індекс. Очікуваний результат:

|        |            | score |
|--------|------------|-------|
| cours  | semester   |       |
| cours1 | semester1  | 90    |
|        | semester2  | 85    |
| cours2 | semester3  | 56    |
|        | semester4  | 72    |
| cours3 | semester5  | 81    |
|        | semester6  | 74    |
| cours4 | semester7  | 85    |
|        | semester8  | 91    |
| cours5 | semester9  | 99    |
|        | semester10 | 93    |

- 2.10. Напишіть програму **Pandas** для вилучення одного рядка, рядків і певного значення з **MultiIndex** рівнів **DataFrame**.
  - 2.11. Напишіть програму **Pandas** для сортування мультиіндексу **DataFrame**. Також сортуйте за різними рівнями індексу.
3. Оформіть звіт

## Pandas: Структурна конкатенація операцій

*Конкатенація* (лат. *concatēnāre*, об'єднання, з'єднання) — це об'єднання (злиття) двох або більше рядків (строк) в один.

**pandas.concat**(objs, \*, axis=0, join='outer', ignore\_index=False, keys=None, levels=None, names=None, verify\_integrity=False, sort=False, copy=None)

**Параметри функції **concat**():**

- **objs**(Series or DataFrame) - Якщо передано відображення, відсортовані ключі використовуватимуться як аргумент ключів, якщо воно не передано, у цьому випадку значення будуть вибрані (див. нижче). Будь-які об'єкти None буде скинуто мовчки, якщо всі вони не є None, у цьому випадку буде викликано ValueError.
- **axis**({0/'index', 1/'columns'}, default 0) - Вісь для об'єднання.
- **join**({'inner', 'outer'}, default 'outer') - Як обробляти індекси на іншій осі (або осях).
- **ignore\_index**(bool, default False) - Якщо True, не використовуйте значення індексу вздовж осі конкатенації. Отримана вісь буде позначена 0, ..., n - 1. Це корисно, якщо ви об'єднуєте об'єкти, де вісь об'єднання не має значущої інформації про індексування. Зверніть увагу, що значення індексів на інших осях все ще враховуються в об'єднанні.
- **keys**(sequence, default None) - Якщо пройдено кілька рівнів, має містити кортежі. Побудуйте ієрархічний індекс, використовуючи передані ключі як крайній рівень.
- **levels**(list of sequences, default None) - Конкретні рівні (унікальні значення), які використовуються для побудови MultiIndex. Інакше вони будуть виведені з ключів.
- **names**(list, default None) - Назви рівнів у результуючому ієрархічному індексі.
- **verify\_integrity**(bool, default False) - Перевірте, чи нова об'єднана вісь містить дублікати. Це може бути дуже дорогим порівняно з фактичним об'єднанням даних.
- **sort**(bool, default False) - Сортувати вісь без конкатенації, якщо вона ще невіривняна.
- **copy**(bool, default True) – Якщо False, не копіюйте дані без потреби.

**Pandas.merge**(left, right, how='inner', on=None, left\_on=None, right\_on=None, left\_index=False, right\_index=False, sort=False, suffixes=('\_x', '\_y'),

copy=None, indicator=False, validate=None)

### Параметри функції **merge()**:

- **left(DataFrame or named Series)** - Об'єкт для об'єднання.
- **right(DataFrame or named Series)** - Об'єкт для об'єднання.
- **how({'left', 'right', 'outer', 'inner', 'cross'}, default 'inner')** - Тип злиття, яке буде виконано.
  - **left**: використовуйте лише ключі з лівого кадру, подібно до лівого зовнішнього об'єднання SQL; зберегти порядок ключів.
  - **right**: використовуйте лише ключі з правого кадру, подібно до правого зовнішнього об'єднання SQL; зберегти порядок ключів.
  - **outer**: використовувати об'єднання ключів з обох фреймів, подібно до повного зовнішнього об'єднання SQL; сортувати ключі лексикографічно.
  - **inner**: використовувати перетин ключів з обох кадрів, подібно до внутрішнього з'єднання SQL; зберегти порядок лівих клавіш.
  - **cross**: створює декартовий добуток з обох кадрів, зберігає порядок лівих клавіш.
- **on(label or list)** - Назви рівнів стовпців або індексів для приєднання. Вони повинні бути знайдені в обох **DataFrames**. Якщо on має значення None і індекси не

об'єднуються, тоді за замовчуванням буде перетин стовпців в обох **DataFrames**.

- **left\_on(label or list, or array-like)** - Назви рівнів стовпців або індексів для об'єднання в лівому **DataFrame**. Також може бути масивом або списком масивів довжини лівого DataFrame. Ці масиви розглядаються як стовпці.
- **right\_on(label or list, or array-like)** - Назви рівнів стовпців або індексів для об'єднання в правому **DataFrame**. Також може бути масивом або списком масивів довжини

правого **DataFrame**. Ці масиви розглядаються як стовпці.

- **left\_index(bool, default False)** - Використовуйте індекс з лівого **DataFrame** як ключ(и) об'єднання. Якщо це MultiIndex, кількість ключів в іншому **DataFrame** (або індекс, або кількість стовпців) має відповідати кількості рівнів.
- **right\_index(bool, default False)** - Використовуйте індекс із правого **DataFrame** як ключ об'єднання. Ті самі застереження, що й left\_index.
- **sort(bool, default False)** - Відсортуйте ключі об'єднання за лексикографією в отриманому DataFrame. Якщо значення False,

порядок ключів об'єднання залежить від типу об'єднання (ключове слово `how`).

- `suffixes(list-like, default is (“_x”, “_y”))` - Послідовність довжини 2, де кожен елемент необов'язково є рядком, що вказує суфікс, який потрібно додати до імен стовпців, що перекриваються, ліворуч і праворуч відповідно. Передайте значення `None` замість рядка, щоб вказати, що ім'я стовпця зліва чи справа слід залишити без суфікса. Принаймні одне зі значень не має бути жодним.
- `copy(bool, default True)` - Якщо `False`, уникайте копіювання, якщо це можливо. `indicator(bool or str, default False)` - Якщо `True`, додає стовпець до вихідного **DataFrame** під назвою «`_merge`» з інформацією про джерело кожного рядка.

Стовпцю можна дати іншу назву, надавши рядковий аргумент. Стовпець матиме категорійний тип зі значенням «`left_only`» для спостережень, чий ключ злиття відображається лише в лівому **DataFrame**, «`right_only`» для спостережень, чий ключ злиття відображається лише в правому **DataFrame**, і «`both`», якщо ключ злиття спостереження знаходиться в обох **DataFrames**.

- `validate(str, optional)` - Якщо вказано, перевіряє, чи злиття має вказаний тип.
  - «`one_to_one`» або «`1:1`»: перевірте, чи унікальні ключі злиття в лівому та правому наборах даних.
  - «`one_to_many`» або «`1:m`»: перевірте, чи унікальні ключі злиття в лівому наборі даних.
  - “`many_to_one`” або “`m:1`”: перевірте, чи унікальні ключі злиття в правильному наборі даних.
  - “`many_to_many`” або “`m:m`”: дозволено, але не призводить до перевірок.

**DataFrame.join(other, on=None, how='left', lsuffix="", rsuffix="", sort=False, validate=None)**

Параметри функції **join()**:

- `other(DataFrame, Series або список, що містить будь-яку їх комбінацію)` - Індекс має бути схожим на один зі стовпців цього. Якщо серія передана, її атрибут `name` має бути встановлено, і це буде використано як ім'я стовпця в результатуючому

об'єднаному **DataFrame**.

- `on(str, list of str, or array-like, optional)` - Ім'я (імена) рівня стовпця або індексу в абоненті для приєднання до індексу в іншому, інакше об'єднується індекс за

індексом. Якщо подано кілька значень, інший **DataFrame** повинен мати **MultiIndex**. Може передати масив як ключ об'єднання, якщо він ще не міститься у викликаючому **DataFrame**. Як операція Excel VLOOKUP.

- `how({'left', 'right', 'outer', 'inner', 'cross'}, default 'left')` - Як керувати роботою двох об'єктів.
  - `left`: використовувати індекс кадру виклику (або стовпець, якщо вказано `on`)
  - `right`: використовувати чужий індекс.
  - `outer`: формує об'єднання індексу кадру виклику (або стовпця, якщо вказано `on`) з індексом іншого та сортує його лексикографічно.
  - `inner`: утворює перетин індексу викликаючого кадру (або стовпця, якщо вказано `on`) з індексом іншого, зберігаючи порядок викликаючого.
  - `cross`: створює декартовий добуток з обох кадрів, зберігає порядок лівих клавіш.
- `lsuffix(str, default '')` - Суфікс для використання з стовпців лівого фрейму, що перекриваються.
- `rsuffix(str, default '')` - Суфікс для використання з стовпців правого фрейму, що перекриваються.
- `sort(bool, default False)` - Упорядкуйте результат **DataFrame** лексикографічно за ключем об'єднання. Якщо `False`, порядок ключа об'єднання залежить від типу об'єднання (ключове слово `how`).
- `validate(str, optional)` - Якщо вказано, перевіряє, чи об'єднання має вказаний тип.

### Приклад використання функції `concat()`:

```
import pandas as pd
import numpy as np
df1 = pd.DataFrame(np.random.randint(25,
    size=(4, 4)), index=["1", "2", "3",
    "4"],
    columns=["A", "B", "C", "D"])
df2 = pd.DataFrame(np.random.randint(25,
    size=(6, 4)), index=["5", "6", "7",
    "8", "9", "10"], columns=["A",
    "B", "C", "D"])
df3 = pd.DataFrame(np.random.randint(25,
    size=(4, 4)), columns=["A", "B",
    "C", "D"])
```

```
df4 = pd.DataFrame(np.random.randint(25,
size=(4, 4)), columns=["E", "F",
"G", "H"])
display(df1, df2, df3, df4)
```

|   | A  | B  | C  | D  |    | A  | B  | C  | D  |   | A  | B  | C | D  |   | E  | F  | G  | H  |
|---|----|----|----|----|----|----|----|----|----|---|----|----|---|----|---|----|----|----|----|
| 1 | 23 | 12 | 22 | 13 | 5  | 11 | 15 | 9  | 14 | 0 | 6  | 10 | 7 | 3  | 0 | 6  | 24 | 1  | 21 |
| 2 | 12 | 4  | 7  | 21 | 6  | 7  | 4  | 15 | 14 | 1 | 1  | 21 | 3 | 19 | 1 | 13 | 17 | 24 | 19 |
| 3 | 4  | 11 | 11 | 13 | 7  | 21 | 21 | 7  | 8  | 2 | 0  | 17 | 8 | 10 | 2 | 2  | 16 | 8  | 22 |
| 4 | 24 | 17 | 15 | 11 | 8  | 13 | 2  | 2  | 0  | 3 | 11 | 20 | 2 | 20 | 3 | 12 | 7  | 17 | 17 |
|   |    |    |    |    | 9  | 20 | 23 | 19 | 8  |   |    |    |   |    |   |    |    |    |    |
|   |    |    |    |    | 10 | 1  | 18 | 23 | 1  |   |    |    |   |    |   |    |    |    |    |

```
# concatenating df1 and df2 along
rows vertical_concat =
pd.concat([df1, df2], axis=0)#
concatenating df3 and df4 along
columns
horizontal_concat = pd.concat([df3, df4], axis=1)
display(vertical_concat, horizontal_concat)
```

|    | A  | B  | C  | D  |   | A  | B  | C  | D  | E  | F  | G  | H  |
|----|----|----|----|----|---|----|----|----|----|----|----|----|----|
| 1  | 10 | 8  | 4  | 10 | 0 | 2  | 16 | 0  | 13 | 8  | 0  | 10 | 4  |
| 2  | 3  | 3  | 11 | 5  | 1 | 4  | 1  | 15 | 20 | 23 | 22 | 18 | 15 |
| 3  | 23 | 19 | 14 | 23 | 2 | 4  | 24 | 6  | 2  | 17 | 21 | 10 | 24 |
| 4  | 2  | 0  | 8  | 22 | 3 | 19 | 22 | 14 | 6  | 11 | 16 | 6  | 22 |
| 5  | 1  | 12 | 23 | 3  |   |    |    |    |    |    |    |    |    |
| 6  | 16 | 6  | 1  | 18 |   |    |    |    |    |    |    |    |    |
| 7  | 18 | 5  | 12 | 17 |   |    |    |    |    |    |    |    |    |
| 8  | 7  | 21 | 17 | 19 |   |    |    |    |    |    |    |    |    |
| 9  | 12 | 6  | 9  | 24 |   |    |    |    |    |    |    |    |    |
| 10 | 3  | 22 | 5  | 15 |   |    |    |    |    |    |    |    |    |

Приклад використання функції `merge()`

```
import pandas as pd
data1 = pd.DataFrame({'id': [1, 2, 3, 4],
'name': ['manoj', 'manoja', 'manoji', 'manij']},
index=['one', 'two', 'three', 'four'])
data2 = pd.DataFrame({'s_id': [1, 2, 3, 6, 7],
'marks': [98, 90, 78, 86, 78]},
```

```
index=['one', 'two', 'three', 'siz', 'seven'])
print(data1.join(data2))
```

|       | id | name   | s_id | marks |
|-------|----|--------|------|-------|
| one   | 1  | manoj  | 1.0  | 98.0  |
| two   | 2  | manoja | 2.0  | 90.0  |
| three | 3  | manoji | 3.0  | 78.0  |
| four  | 4  | manij  | NaN  | NaN   |

```
print(pd.merge(data1, data2, left_index=True, right_index=True))
```

|       | id | name   | s_id | marks |
|-------|----|--------|------|-------|
| one   | 1  | manoj  | 1    | 98    |
| two   | 2  | manoja | 2    | 90    |
| three | 3  | manoji | 3    | 78    |

Приклад використання функції `merge()`

```
import pandas as pd
```

```
technologies = {
```

```
    'Courses':["Spark","PySpark","Python","pandas"], 'Fee'
```

```
    :[20000,25000,22000,30000],
```

```
    'Duration':['30days','40days','35days','50days'],
```

```
    }
```

```
index_labels=['r1','r2','r3','r4']
```

```
df1 = pd.DataFrame(technologies,index=index_labels)
```

```
technologies2 = {
```

```
    'Courses':["Spark","Java","Python","Go"],
```

```
    'Discount':[2000,2300,1200,2000]
```

```
    }
```

```
index_labels2=['r1','r6','r3','r5']
```

```
df2 = pd.DataFrame(technologies2,index=index_labels2)
```

|    | Courses | Fee   | Duration |
|----|---------|-------|----------|
| r1 | Spark   | 20000 | 30days   |
| r2 | PySpark | 25000 | 40days   |
| r3 | Python  | 22000 | 35days   |
| r4 | pandas  | 30000 | 50days   |

|    | Courses | Discount |
|----|---------|----------|
| r1 | Spark   | 2000     |
| r6 | Java    | 2300     |
| r3 | Python  | 1200     |
| r5 | Go      | 2000     |

```
df3=df1.join(df2, lsuffix="_left",
rsuffix="_right")print(df3)
```

#Output

|          | Courses_left     | Fee    | Duration   | Courses_right |        |
|----------|------------------|--------|------------|---------------|--------|
| Discount | r1               |        |            | Spark         | 20000  |
|          | 30days           |        |            | Spark         | 2000.0 |
| r2       | PySpark<br>25000 | 40days | NaN        | NaN           |        |
| r3       | Python<br>22000  | 35days | Pytho<br>n | 1200.<br>0    |        |
| r4       | pandas<br>30000  | 50days | NaN        | NaN           |        |

## Практична робота №9

**Мета роботи:** отримати навички у об'єднаннях багатовимірних масивів.

### Порядок виконання роботи

1. Створіть новий файл.
2. Виконайте наступні завдання:
  - 2.1. Напишіть програму **Pandas**, щоб об'єднати два фрейми даних уздовж рядків і призначити всі дані.  
student\_data1:

| Index | student_id | name             | marks |
|-------|------------|------------------|-------|
| 0     | S1         | Danniella Fenton | 200   |
| 1     | S2         | Ryder Storey     | 210   |
| 2     | S3         | Bryce Jensen     | 190   |
| 3     | S4         | Ed Bernal        | 222   |
| 4     | S5         | Kwame Morin      | 199   |

student\_data2:

| Index | student_id | name             | marks |
|-------|------------|------------------|-------|
| 0     | S4         | Scarlette Fisher | 201   |
| 1     | S5         | Carla Williamson | 200   |
| 2     | S6         | Dante Morse      | 198   |
| 3     | S7         | Kaiser William   | 219   |
| 4     | S8         | Madeeha Preston  | 201   |

- 2.2. Напишіть програму **Pandas**, щоб об'єднати два фрейми даних уздовж стовпців і призначити всі дані.  
Використовуйте фрейми student\_data1 та student\_data2
  - 2.3. Напишіть програму **Pandas** для додавання рядків до існуючого **DataFrame** і відображення об'єднаних даних.

Використовуйте фрейми student\_data1 та student\_data2

| Параметр   | Значення  |
|------------|-----------|
| student_id | S6        |
| name       | Scarlette |
| Fisher     | 205       |

- 2.4. Напишіть програму **Pandas**, щоб додати список **dictionaries** або **series**  
до існуючого **DataFrame** і відобразити об'єднані дані.

Використовуйте фрейми student\_data1 та student\_data2

| Параметр   | Значення  |
|------------|-----------|
| student_id | S6        |
| name       | Scarlette |
| Fisher     | 205       |

- 2.5. Напишіть програму **Pandas** для об'єднання двох фреймів даних уздовж рядків і об'єднання з іншим фреймом даних уздовж загального ідентифікатора стовпця.  
Використовуйте фрейми student\_data1 та student\_data2

```
exam_data:
  student_id exam_id
0      S1      23
1      S2      45
2      S3      12
3      S4      67
4      S5      21
5      S7      55
6      S8      33
7      S9      14
8     S10      56
9     S11      83
10    S12      88
11    S13      12
```

2.6. Напишіть програму **Pandas** для об'єднання двох фреймів даних за допомогою спільного стовпця обох фреймів даних.

Використовуйте фрейми `student_data1` та `student_data2`

2.7. Напишіть програму **Pandas**, щоб об'єднати два фрейми даних із відповідними записами з обох сторін, де це можливо.

Використовуйте фрейми `student_data1` та `student_data2`

2.8. Напишіть програму **Pandas** для об'єднання (з'єднання зліва) двохкадрів даних, використовуючи ключі лише з лівого кадру даних.

```
data1:
  key1 key2 P Q
0  K0  K0 P0 Q0
1  K0  K1 P1 Q1
2  K1  K0 P2 Q2
3  K2  K1 P3 Q3
data2:
  key1 key2 R S
0  K0  K0 R0 S0
1  K1  K0 R1 S1
2  K1  K0 R2 S2
3  K2  K0 R3 S3
```

2.9. Напишіть програму **Pandas** для об'єднання двох фреймів даних, використовуючи ключі лише правого фрейму даних.

Використовуйте фрейми `data1` та `data2`

2.10. Напишіть програму **Pandas** для об'єднання двох заданих наборів даних за допомогою кількох ключів об'єднання.

Використовуйте фрейми `data1` та `data2`

2.11. Напишіть програму **Pandas** для створення нового **DataFrame** на основі існуючих **Series**, використовуючи вказаний аргумент і замінивши існуючі назви стовпців.

Використовуйте фрейми `data1` та `data2`

2.12. Напишіть програму **Pandas** для створення комбінації з двох

фреймів даних, де комбінація ідентифікатора стовпця з'являється більше одного разу в обох фреймах даних.

Використовуйте фрейми data1 та data2

- 2.13. Напишіть програму **Pandas**, щоб об'єднати стовпці двох потенційно різноіндексованих **DataFrame** в один результат **DataFrame**.

```
data1:
  A  B
K0  A0 B0
K1  A1 B1
K2  A2 B2
data2:
  C  D
K0  C0 D0
K2  C2 D2
K3  C3 D3
```

- 2.14. Напишіть програму **Pandas** для об'єднання двох фреймів даних із різними стовпцями.

```
data1:
  key1 key2  P  Q
0  K0  K0  P0  Q0
1  K0  K1  P1  Q1
2  K1  K0  P2  Q2
3  K2  K1  P3  Q3
data2:
  key1 key2  R  S
0  K0  K0  R0  S0
1  K1  K0  R1  S1
2  K1  K0  R2  S2
3  K2  K0  R3  S3
```

3. Оформіть звіт

## Pandas: Зведена таблиця та файлові операції

`pandas.pivot_table(data, values=None, index=None, columns=None, aggfunc='mean', fill_value=None, margins=False, dropna=True, margins_name='All', observed=False, sort=True)`

**Параметри функції `pivot_table()`:**

- `data` - `DataFrame`.
- `values(list-like or scalar, optional)` - Стовець або стовпці для агрегування.  
`index(column, Grouper, array, or list of the previous)` - Якщо передається масив, він повинен мати таку саму довжину, що й дані. Список може містити будь-який інший тип (окрім списку). Ключі для групування в індексі зведеної таблиці. Якщо передається масив, він використовується так само, як і значення стовпців.
- `columns(column, Grouper, array, or list of the previous)` - Якщо передається масив, він повинен мати таку саму довжину, що й дані. Список може містити будь-який інший тип (окрім списку). Ключі для групування в стовпці зведеної таблиці. Якщо передається масив, він використовується так само, як і значення стовпців.
- `aggfunc(function, list of functions, dict, default numpy.mean)` - Якщо передано список функцій, результуюча зведена таблиця матиме ієрархічні стовпці, верхнім рівнем яких є імена функцій (виведені з самих об'єктів функції). Якщо передано dict, ключ – стовець для агрегування, а значення – функція або список функцій. Якщо `margin=True`, `aggfunc` використовуватиметься для обчислення часткових агрегатів.
- `fill_value(scalar, default None)` - Значення, яким потрібно замінити відсутні значення (у результуючій зведеній таблиці після агрегування).
- `margins(bool, default False)` - Якщо `margins=True`, спеціальні ALL стовпці та рядки будуть додані з частковими груповими агрегатами для категорій у рядках і стовпцях.
- `dropna(bool, default True)` - Не включайте стовпці, усі записи яких є NaN. Якщо значення `True`, рядки зі значенням NaN у будь-якому стовпці будуть пропущені перед обчисленням полів.
- `margins_name(str, default 'All')` - Ім'я рядка/стовпця, які міститимуть підсумки, коли поля мають значення `True`.
- `observed(bool, default False)` - Це стосується лише тих випадків, коли будь-який групер є категоричним. Якщо `True`: показувати лише спостережувані значення для групувальників категорій. Якщо `False`: показати всі значення для групувальників категорій.
- `sort(bool, default True)` - Визначає, чи слід сортувати результат.

`pandas.read_csv(filepath_or_buffer, *, sep=_NoDefault.no_default, delimiter=None, header='infer', names=_NoDefault.no_default, index_col=None, usecols=None, dtype=None, engine=None, converters=None, true_values=None, false_values=None, skipinitialspace=False, skiprows=None, skipfooter=0, nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=None, infer_datetime_format=_NoDefault.no_default, keep_date_col=False, date_parser=_NoDefault.no_default, date_format=None, dayfirst=False, cache_dates=True, iterator=False, chunksize=None, compression='infer', thousands=None, decimal='.', lineterminator=None, quotechar='"', quoting=0, doublequote=True, escapechar=None, comment=None, encoding=None, encoding_errors='strict', dialect=None,`

`on_bad_lines='error', delim_whitespace=False, low_memory=True, memory_map=False, float_precision=None, storage_options=None, dtype_backend=_NoDefault.no_default)`

- Читання файлу зі значеннями, розділеними комами (csv) у DataFrame. Також підтримує додаткову ітерацію або розбиття файлу на частини.

`pandas.read_excel(io, sheet_name=0, *, header=0, names=None, index_col=None, usecols=None, dtype=None, engine=None, converters=None, true_values=None, false_values=None, skiprows=None, nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False, parse_dates=False, date_parser=_NoDefault.no_default, date_format=None, thousands=None, decimal='.', comment=None, skipfooter=0, storage_options=None, dtype_backend=_NoDefault.no_default)`

- Читання файлу Excel у **pandas DataFrame**. Підтримує xls, xlsx, xlsxm, xlsb, odf, ods і розширення файлів odt, які зчитуються з локальної файлової системи або URL-адреси. Підтримує опцію читання окремого аркуша або списку аркушів.

`DataFrame.to_csv(path_or_buf=None, sep=',', na_rep="", float_format=None, columns=None, header=True, index=True, index_label=None, mode='w', encoding=None, compression='infer', quoting=None, quotechar='"', lineterminator=None, chunksize=None, date_format=None, doublequote=True, escapechar=None, decimal='.', errors='strict', storage_options=None)`

- Запис об'єкту у файл із значеннями, розділеними комами (csv).

`DataFrame.to_excel(excel_writer, sheet_name='Sheet1', na_rep="", float_format=None, columns=None, header=True, index=True, index_label=None, startrow=0, startcol=0, engine=None, merge_cells=True, inf_rep='inf', freeze_panes=None, storage_options=None)`

Запис об'єкту на аркуш Excel. Щоб записати один об'єкт у файл Excel .xlsx, потрібно лише вказати ім'я цільового файлу. Щоб писати на кількох аркушах, необхідно створити об'єкт ExcelWriter з іменем цільового файлу та вказати аркуш у файлі для запису.

## Приклади

Вхідні дані

Out[1]:

|   | A     | B        | C  |
|---|-------|----------|----|
| 0 | John  | Masters  | 27 |
| 1 | Boby  | Graduate | 23 |
| 2 | Mina  | Graduate | 21 |
| 3 | Peter | Masters  | 23 |
| 4 | Nicky | Graduate | 24 |

```
table = pd.pivot_table(df, index=['A', 'B'])
table
```

Out[2]:

|       |          | C  |
|-------|----------|----|
| A     | B        |    |
| Boby  | Graduate | 23 |
| John  | Masters  | 27 |
| Mina  | Graduate | 21 |
| Nicky | Graduate | 24 |
| Peter | Masters  | 23 |

```
table = pd.pivot_table(df, values='A', index=['B', 'C'],
                        columns=['B'], aggfunc = np.sum)
```

table

Out[3]:

|          |    | B | Graduate | Masters |
|----------|----|---|----------|---------|
| B        | C  |   |          |         |
| Graduate | 21 |   | Mina     | NaN     |
|          | 23 |   | Boby     | NaN     |
|          | 24 |   | Nicky    | NaN     |
| Masters  | 23 |   | NaN      | Peter   |
|          | 27 |   | NaN      | John    |

```
pd.read_csv("example1.csv")
```

|    | total_bill | tip  | sex    | smoker | day | time   | size |
|----|------------|------|--------|--------|-----|--------|------|
| 0  | 16.99      | 1.01 | Female | No     | Sun | Dinner | 2    |
| 1  | 10.34      | 1.66 | Male   | No     | Sun | Dinner | 3    |
| 2  | 21.01      | 3.50 | Male   | No     | Sun | Dinner | 3    |
| 3  | 23.68      | 3.31 | Male   | No     | Sun | Dinner | 2    |
| 4  | 24.59      | 3.61 | Female | No     | Sun | Dinner | 4    |
| 5  | 25.29      | 4.71 | Male   | No     | Sun | Dinner | 4    |
| 6  | 8.77       | 2.00 | Male   | No     | Sun | Dinner | 2    |
| 7  | 26.88      | 3.12 | Male   | No     | Sun | Dinner | 4    |
| 8  | 15.04      | 1.96 | Male   | No     | Sun | Dinner | 2    |
| 9  | 14.78      | 3.23 | Male   | No     | Sun | Dinner | 2    |
| 10 | 10.27      | 1.71 | Male   | No     | Sun | Dinner | 2    |
| 11 | 35.26      | 5.00 | Female | No     | Sun | Dinner | 4    |

```
df = pd.read_csv('headbrain1.csv',
                 sep='[:,|_]', engine='python')
df
```

|       | totalbill | tip  | Unnamed: 2 | sex  | smoker | Unnamed: 5 | day    | time   | Unnamed: 8 | size |
|-------|-----------|------|------------|------|--------|------------|--------|--------|------------|------|
| 16.99 | NaN       | 1.01 | Female     | No   | NaN    | Sun        | NaN    | Dinner | NaN        | 2.0  |
| 10.34 | NaN       | 1.66 | NaN        | Male | NaN    | No         | Sun    | Dinner | NaN        | 3.0  |
| 21.01 | 3.50      | Male | NaN        | No   | Sun    | NaN        | Dinner | NaN    | 3.0        | NaN  |
| 23.68 | NaN       | 3.31 | NaN        | Male | No     | NaN        | Sun    | Dinner | NaN        | 2.0  |
| 24.59 | 3.61      | NaN  | Female     | No   | NaN    | Sun        | NaN    | Dinner | NaN        | 4.0  |
| 25.29 | NaN       | 4.71 | Male       | NaN  | No     | Sun        | NaN    | Dinner | NaN        | 4.0  |

```
df = pd.read_csv('example1.csv',
                 header=0,
                 usecols=["tip", "sex", "time"])
df
```

|    | tip  | sex    | time   |
|----|------|--------|--------|
| 0  | 1.01 | Female | Dinner |
| 1  | 1.66 | Male   | Dinner |
| 2  | 3.50 | Male   | Dinner |
| 3  | 3.31 | Male   | Dinner |
| 4  | 3.61 | Female | Dinner |
| 5  | 4.71 | Male   | Dinner |
| 6  | 2.00 | Male   | Dinner |
| 7  | 3.12 | Male   | Dinner |
| 8  | 1.96 | Male   | Dinner |
| 9  | 3.23 | Male   | Dinner |
| 10 | 1.71 | Male   | Dinner |
| 11 | 5.00 | Female | Dinner |

```
nme = ["aparna", "pankaj", "sudhir", "Geeku"]
deg = ["MBA", "BCA", "M.Tech",
      "MBA"]scr = [90, 40, 80, 98]
dict = {'name': nme, 'degree': deg, 'score': scr} df
= pd.DataFrame(dict)
df.to_csv('file1.csv')
```

|   | name   | degree | score |
|---|--------|--------|-------|
| 0 | aparna | MBA    | 90    |
| 1 | pankaj | BCA    | 40    |
| 2 | sudhir | M.Tech | 80    |
| 3 | Geeku  | MBA    | 98    |

```
df.to_csv('file2.csv', header=False, index=False)
```

```
file=('path_of_excel_file')
```

```
newData = pds.read_excel(file)
```

```
newData
```

jupyter file2.csv a few seconds ago

File Edit View Language

```

1 1 1 aparna,MBA,90
2 2 2 pankaj,BCA,40
3 3 3 sudhir,M.Tech,80
4 4 4 Geeku,MBA,98
5 5 5

```

|   | Sl. No. | Stream             | Name     | Height | Weight |
|---|---------|--------------------|----------|--------|--------|
| 0 | 1       | Science            | Ayush    | 5.60   | 67     |
| 1 | 2       | Science            | Anirban  | 5.80   | 56     |
| 2 | 3       | Commerce           | Saurav   | 6.10   | 76     |
| 3 | 4       | Humanities         | Laxman   | 5.90   | 58     |
| 4 | 5       | Commerce           | Rahul    | 5.80   | 49     |
| 5 | 6       | Science            | Sneha    | 6.00   | 55     |
| 6 | 7       | Hotel Management   | Harshit  | 5.11   | 68     |
| 7 | 8       | Humanities         | Bhaibhav | 5.90   | 63     |
| 8 | 9       | Aviation           | Karan    | 5.80   | 59     |
| 9 | 10      | Physical Education | Tyagi    | 5.50   | 65     |

```
sheet1 = pds.read_excel(file,
    sheet_name = 0,
    index_col = 0)
```

```
sheet2 = pds.read_excel(file,
    sheet_name = 1,
    index_col = 0)
```

```
newData = pds.concat([sheet1, sheet2])
```

```
newData
```

|    | Sl. No. | Stream             | Name     | Height | Weight |
|----|---------|--------------------|----------|--------|--------|
| 1  | 1       | Science            | Ayush    | 5.60   | 67     |
| 2  | 2       | Science            | Anirban  | 5.80   | 56     |
| 3  | 3       | Commerce           | Saurav   | 6.10   | 76     |
| 4  | 4       | Humanities         | Laxman   | 5.90   | 58     |
| 5  | 5       | Commerce           | Rahul    | 5.80   | 49     |
| 6  | 6       | Science            | Sneha    | 6.00   | 55     |
| 7  | 7       | Hotel Management   | Harshit  | 5.11   | 68     |
| 8  | 8       | Humanities         | Bhaibhav | 5.90   | 63     |
| 9  | 9       | Aviation           | Karan    | 5.80   | 59     |
| 10 | 10      | Physical Education | Tyagi    | 5.50   | 65     |
| 11 | 11      | Humanities         | Ankita   | 5.10   | 67     |
| 12 | 12      | Aviation           | Nakul    | 5.00   | 64     |
| 13 | 13      | Business           | Rohan    | 5.60   | 56     |
| 14 | 14      | Science            | Satkarsh | 6.00   | 47     |
| 15 | 15      | Science            | Yuva     | 5.90   | 55     |
| 16 | 16      | Commerce           | Piyush   | 5.40   | 58     |
| 17 | 17      | Hotel Management   | Sreeram  | 5.50   | 62     |
| 18 | 18      | Hotel Management   | Rakul    | 5.60   | 54     |

```
dct = {'ID': {0: 23, 1: 43, 2: 12,
3: 13, 4: 67, 5: 89,
6: 90, 7: 56, 8: 34},
'Name': {0: 'Ram', 1: 'Deep',
2: 'Yash', 3: 'Aman',
4: 'Arjun', 5: 'Aditya',
6: 'Divya', 7: 'Chalsea',
8: 'Akash' },
'Marks': {0: 89, 1: 97, 2: 45, 3: 78,
4: 56, 5: 76, 6: 100, 7: 87,
8: 81},
'Grade': {0: 'B', 1: 'A', 2: 'F', 3: 'C',
4: 'E', 5: 'C', 6: 'A', 7: 'B',
8: 'B'}}
data = pd.DataFrame(dct)
data.to_excel("output.xlsx")
```

|    | A | B  | C       | D     | E     |
|----|---|----|---------|-------|-------|
| 1  |   | ID | Name    | Marks | Grade |
| 2  | 0 | 23 | Ram     | 89    | B     |
| 3  | 1 | 43 | Deep    | 97    | A     |
| 4  | 2 | 12 | Yash    | 45    | F     |
| 5  | 3 | 13 | Aman    | 78    | C     |
| 6  | 4 | 67 | Arjun   | 56    | E     |
| 7  | 5 | 89 | Aditya  | 76    | C     |
| 8  | 6 | 90 | Divya   | 100   | A     |
| 9  | 7 | 56 | Chalsea | 87    | B     |
| 10 | 8 | 34 | Akash   | 81    | B     |
| 11 |   |    |         |       |       |

## Практична робота №10

**Мета роботи:** ознайомитись та засвоїти практичні навички роботи із зведеними таблицями. Набути навички роботи з файловими операціями.

### Порядок виконання роботи

1. Створіть новий файл.
2. Виконайте наступні завдання:
  - 2.1. Напишіть програму **Pandas**, щоб створити зведену таблицю та знайти загальну суму продажу для регіону, менеджера, продавця. (SaleData.xlsx)
  - 2.2. Напишіть програму **Pandas**, щоб створити зведену таблицю та знайти продану одиницю товару (SaleData.xlsx)
  - 2.3. Напишіть програму **Pandas**, щоб створити зведену таблицю та знайти продану одиницю за регіоном і товаром (SaleData.xlsx)
  - 2.4. Напишіть програму **Pandas** для створення зведеної таблиці та підрахунку обсягу продажів менеджером і середнього значення суми продажу. (SaleData.xlsx)
  - 2.5. Напишіть програму **Pandas**, щоб створити зведену таблицю та знайти загальний обсяг продажів для менеджерів і продавців, а також відобразити суму всіх продажів внизу. (SaleData.xlsx)
  - 2.6. Напишіть програму **Pandas**, щоб створити зведену таблицю та знайти загальну суму продажу за регіоном, менеджером, продавцем, де менеджер = "Douglas" (SaleData.xlsx)
  - 2.7. Напишіть програму **Pandas**, щоб створити зведену таблицю та знайти продані телевізори та домашні кінотеатри в регіоні. (SaleData.xlsx)
  - 2.8. Напишіть програму **Pandas**, щоб створити зведену таблицю та знайти максимальну вартість продажу товарів. (SaleData.xlsx)
  - 2.9. Напишіть програму **Pandas**, щоб створити зведену таблицю та знайти мінімальну вартість продажу товарів. (SaleData.xlsx)
  - 2.10. Напишіть програму **Pandas**, щоб створити зведену таблицю та знайти максимальну та мінімальну вартість

продажу товарів. (SaleData.xlsx)

- 2.11. Напишіть програму Pandas для запису DataFrame у файл CSV за допомогою розділювача табуляції.

Вхідні дані:

Original

DataFrame col1

col2 col3

0 1 4 7

1 4 5 8

2 3 6 9

3 4 7 0

4 5 8 1

- 2.12. Напишіть програму **Pandas** для імпорту заданих даних Excel (employee.xlsx) у фрейм даних **Pandas** і знайдіть список працівників, де hire\_date між двома конкретними місяцем і роком.

- 2.13. Напишіть програму **Pandas** для імпорту заданих даних Excel (employee.xlsx) у фрейм даних **Pandas** і перетворення даних для використання hire\_date як індексу.

- 2.14. Напишіть програму **Pandas** для імпорту заданих даних Excel (employee.xlsx) у фрейм даних **Pandas** і сортування на основі кількох заданих стовпців.

- 2.15. Напишіть програму **Pandas** для імпорту даних sheet2 із заданих даних Excel (employee.xlsx) у фрейм даних **Pandas**.

- 2.16. Напишіть програму **Pandas**, щоб імпортувати три таблиці даних із заданих даних Excel (coalpublic2013.xlsx) і об'єднати в **dataframe**. Примітка. Структура трьох таблиць даних однакова.

- 2.17. Напишіть програму **Pandas**, щоб імпортувати три таблиці даних із заданих даних Excel (employee.xlsx) в один кадр даних і експортувати результат у новий файл Excel.  
*Примітка.* Структура трьох таблиць даних однакова.

### 3.Оформіть звіт