**ỦY BAN NHÂN DÂN THÀNH PHỐ HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC SÀI GÒN**

A blue circle with white text

Description automatically generated**˗˗˗˗˗˗˗˗˗˗˗□&□˗˗˗˗˗˗˗˗˗˗˗**

**BÁO CÁO NHÓM**

**HỌC PHẦN: PHÂN TÍCH DỮ LIỆU**

**PHÂN TÍCH KHÁM PHÁ VỀ BỆNH ĐÁI THÁO ĐƯỜNG**

|  |  |
| --- | --- |
| **Mã học phần** | **841432** |
| **Học kỳ** | **1** |
| **GVHD** | **Đỗ Như Tài** |
| **MSSV** | **3122410490** |
| **Họ và tên** | **Nguyễn Hoàng Mai Vy** |

**TP. HỒ CHÍ MINH THÁNG 10 NĂM 2025**

**MỤC LỤC**

[BẢNG PHÂN CÔNG CÔNG VIỆC 5](#_Toc211546707)

[13.1. BITLY DATA FROM 1.USA.GOV 6](#_Toc211546708)

[13.1.1. JSON (JavaScript Object Notation) 6](#_Toc211546709)

[13.1.2. Phân tích chuỗi (String Manipulation) 6](#_Toc211546710)

[13.1.3. Tổng hợp và phân tích cắt ngang 7](#_Toc211546711)

[13.1.4. Ví dụ 8](#_Toc211546712)

[13.2. MOVIELENS 1M DATASET 9](#_Toc211546713)

[13.2.1. Cấu trúc dữ liệu 9](#_Toc211546714)

[13.2.2. Nạp dữ liệu vào pandas 9](#_Toc211546715)

[13.2.3. Kết hợp dữ liệu (merge) 9](#_Toc211546716)

[13.2.4. Tính trung bình đánh giá theo giới tính 10](#_Toc211546717)

[13.2.5. Lọc phim có ít nhất 250 lượt đánh giá 10](#_Toc211546718)

[13.2.6. Top phim được nữ giới đánh giá cao nhất 10](#_Toc211546719)

[13.2.7. So sánh mức chênh lệch giữa nam và nữ 10](#_Toc211546720)

[13.2.8. Đo mức độ “bất đồng ý kiến” (variance) 11](#_Toc211546721)

[13.2.9. Phân tích theo thể loại (genre) 11](#_Toc211546722)

[13.3. US BABY NAMES 1880–2010 12](#_Toc211546723)

[13.3.1. Cấu trúc của bộ dữ liệu 12](#_Toc211546724)

[13.3.2. Tiền xử lý và nạp dữ liệu 13](#_Toc211546725)

[13.3.3. Thực hiện nạp và hợp nhất dữ liệu 13](#_Toc211546726)

[13.3.4. Đánh giá chất lượng dữ liệu 14](#_Toc211546727)

[13.3.5. Kết luận 14](#_Toc211546728)

[13.4. USDA FOOD DATABASE 16](#_Toc211546729)

[13.4.1. Mục tiêu 16](#_Toc211546730)

[13.4.2. Dữ liệu 16](#_Toc211546731)

[13.4.3. Các bước thực hiện & Phân tích: 17](#_Toc211546732)

[13.4.3.1. Đọc và làm phẳng dữ liệu 17](#_Toc211546733)

[˗ Tải dữ liệu: Dùng thư viện json để tải toàn bộ tệp vào một danh sách các dictionary. 17](#_Toc211546734)

[13.4.3.2. Hợp nhất và phân tích 23](#_Toc211546735)

[13.4.4. Các hàm/phương thức chính được sử dụng 27](#_Toc211546736)

[13.5. 2012 Federal Election Commission Database 29](#_Toc211546737)

[13.5.1. Khái niệm về Data Wrangling và Data Cleaning 29](#_Toc211546738)

[13.5.2. Phân tích Dữ liệu Tổng hợp (Aggregation) 29](#_Toc211546739)

[13.5.3. Phân tích Chuỗi Thời gian (Time Series Analysis) 29](#_Toc211546740)

[13.5.4. Ví dụ 30](#_Toc211546741)

[13.6. Conclusion 32](#_Toc211546742)

[A.1. ndarray Object Internals 32](#_Toc211546743)

[1. Tổng quan 32](#_Toc211546744)

[2. Cấu trúc nội bộ của ndarray 32](#_Toc211546745)

[3. Ví dụ minh họa 33](#_Toc211546746)

[4. Strides âm và “zero-copy views” 33](#_Toc211546747)

[5. Phân cấp kiểu dữ liệu trong NumPy (Data Type Hierarchy) 34](#_Toc211546748)

[A.2. Advanced Array Manipulation 34](#_Toc211546749)

[1. Giới thiệu chung 34](#_Toc211546750)

[2. Thay đổi hình dạng mảng – reshape 34](#_Toc211546751)

[3. Làm phẳng mảng – ravel() và flatten() 35](#_Toc211546752)

[4. Trật tự bộ nhớ: C Order vs FORTRAN Order 35](#_Toc211546753)

[5. Nối và chia mảng – Concatenation & Splitting 36](#_Toc211546754)

[6. Công cụ r\_ và c\_ 37](#_Toc211546755)

[7. Lặp phần tử – repeat() và tile() 37](#_Toc211546756)

[8. Chỉ số nâng cao – take() và put() 38](#_Toc211546757)

[A.3. Broadcasting 39](#_Toc211546758)

[1. Khái niệm cơ bản 39](#_Toc211546759)

[2. Ví dụ cơ bản: Nhân mảng với số vô hướng (scalar) 39](#_Toc211546760)

[3. Ví dụ thực tế: Chuẩn hóa dữ liệu (demeaning) 39](#_Toc211546761)

[4. Quy tắc Broadcasting 40](#_Toc211546762)

[5. Ví dụ: Demean theo hàng (axis 1) 41](#_Toc211546763)

[6. Broadcasting với mảng 3 chiều 41](#_Toc211546764)

[8.  Dùng np.newaxis để thêm chiều phục vụ broadcasting 42](#_Toc211546765)

[9. Viết hàm tổng quát cho demeaning trên mọi axis 43](#_Toc211546766)

[10.  Broadcasting khi gán giá trị (assignment) 43](#_Toc211546767)

[A.4. Advanced ufunc Usage 44](#_Toc211546768)

[1. Giới thiệu và nạp dữ liệu 44](#_Toc211546769)

[2. Tiền xử lý và tổ chức dữ liệu 44](#_Toc211546770)

[3. Tổng hợp và trực quan hóa số lượng sinh 45](#_Toc211546771)

[4. Chuẩn hóa và phân tích dữ liệu 45](#_Toc211546772)

[5. Phân tích và lọc dữ liệu 46](#_Toc211546773)

[6. Kiểm soát giá trị cực trị bằng np.clip() 46](#_Toc211546774)

[7. Hiệu năng và tối ưu hóa 47](#_Toc211546775)

[A.5. Structured and Record Arrays 48](#_Toc211546776)

[5.1. Structured Array 48](#_Toc211546777)

[5.2. Record Array 52](#_Toc211546778)

[A.6. More About Sorting 57](#_Toc211546779)

[6.1. Giới thiệu về Sorting trong NumPy 57](#_Toc211546780)

[6.2. Sắp xếp cơ bản theo từng trường 58](#_Toc211546781)

[6.3. Sắp xếp đa tiêu chí (multi-field sorting) 58](#_Toc211546782)

[6.4. Sắp xếp sử dụng chỉ số (np.argsort) 59](#_Toc211546783)

[6.5. Sắp xếp giảm dần và lọc nhóm con 59](#_Toc211546784)

[6.6. Tổng kết 60](#_Toc211546785)

[A.7. Writing Fast NumPy Functions with Numba 60](#_Toc211546786)

[7.1. Cách hoạt động 61](#_Toc211546787)

[7.2. Tạo các đối tượng ufunc tùy chỉnh 62](#_Toc211546788)

[A.8. Advanced Array Input and Output 63](#_Toc211546789)

[8.1. Tệp ánh xạ bộ nhớ (Memory-Mapped Files) 63](#_Toc211546790)

[8.2. Giải phóng bộ nhớ và xóa file 66](#_Toc211546791)

[8.3. HDF5 và các tùy chọn lưu trữ mảng khá 66](#_Toc211546792)

[A.9. Performance Tips 67](#_Toc211546793)

[9.1. Các mẹo chính 67](#_Toc211546794)

[9.2. Tầm quan trọng của bộ nhớ liền kề (Contiguous Memory) 67](#_Toc211546795)

BẢNG PHÂN CÔNG CÔNG VIỆC

|  |  |  |
| --- | --- | --- |
| **Họ và tên** | **MSSV** | **Công việc** |
| Nguyễn Hoàng Mai Vy | 3122410490 | * Phân công công cả nhóm * Code * Viết báo cáo và làm slide |
| Dương Bình Minh | 3122410236 | * Viết báo cáo và làm slide * Code |
| Đỗ Thanh Duy | 3120410085 | * Viết báo cáo và làm slide * Code |
| Nguyễn Minh Quang | 3123410290 | * Viết báo cáo và làm slide * Code |

13.1. BITLY DATA FROM 1.USA.GOV

13.1.1. JSON (JavaScript Object Notation)

* **Khái niệm:** JSON là một định dạng trao đổi dữ liệu văn bản phổ biến, dễ đọc và dễ viết cho con người, dễ dàng phân tích và tạo ra bằng máy tính. Nó được sử dụng rộng rãi trong lập trình web (web APIs).
* **Ứng dụng trong Code:**
  + Sử dụng mô-đun **json** của Python.
  + Hàm **json.loads()** được dùng để phân tích cú pháp (parsing) một chuỗi JSON (tức là mỗi dòng trong tệp dữ liệu) và chuyển đổi nó thành **từ điển Python**.
* **JSON (JavaScript Object Notation):** Định dạng trao đổi dữ liệu tiêu chuẩn. Trong Python, được chuyển đổi thành **từ điển** bằng hàm json.loads().
* **Data Wrangling (Làm sạch dữ liệu):** Xử lý giá trị thiếu (.fillna()) và làm sạch chuỗi rỗng ('').
* **Phân tích Cắt ngang:** Sử dụng **value\_counts()** để đếm tần suất và **groupby().unstack()** để tạo bảng tổng hợp so sánh các thuộc tính theo nhiều chiều.

13.1.2. Phân tích chuỗi (String Manipulation)

* **Ứng dụng:** Được sử dụng để làm sạch và trích xuất thông tin từ các chuỗi phức tạp như **User Agent String** (trường a).
* **Kỹ thuật:** Sử dụng các phương thức chuỗi của Pandas (.str.contains()) hoặc các hàm Python thuần (.split()) để kiểm tra sự tồn tại của một chuỗi con (ví dụ: "Windows") và trích xuất token đầu tiên (tên trình duyệt).
* **numpy.where:** Được sử dụng như một hàm điều kiện (tương đương với mệnh đề if-else) để tạo một cột mới (os) dựa trên kết quả của việc kiểm tra chuỗi.

|  |  |  |
| --- | --- | --- |
| **Bước** | **Nội dung** | **Kết quả chính** |
| 1. Tải & Chuyển đổi | Đọc tệp, chuyển đổi từng dòng JSON thành danh sách từ điển, tạo **Pandas DataFrame**. | DataFrame chứa 3560 bản ghi, 18 cột. |
| 2. Phân tích Múi giờ | Làm sạch cột tz (điền 'Missing', 'Unknown'), sau đó dùng .value\_counts() để đếm. | **America/New\_York** là múi giờ phổ biến nhất. |
| 3. Phân loại OS | Trích xuất thông tin từ chuỗi User Agent (a), tạo cột **os** phân loại thành **'Windows'** và **'Not Windows'**. | Phân loại người dùng theo hệ điều hành. |
| 4. Tổng hợp OS & TZ | Nhóm dữ liệu theo tz và os, sau đó dùng **.unstack()** để tạo bảng tổng hợp so sánh số lượng và tỷ lệ % OS trong top 10 múi giờ. | Phân bố tỷ lệ % người dùng Windows trong từng múi giờ. |

13.1.3. Tổng hợp và phân tích cắt ngang

* **value\_counts():** Là phương thức cơ bản nhất để phân tích dữ liệu phân loại (categorical data), dùng để tính tần suất xuất hiện của các giá trị duy nhất trong một cột (ví dụ: đếm múi giờ).
* **groupby() và unstack():**
  + **groupby()** tạo ra các nhóm dữ liệu dựa trên một hoặc nhiều khóa (ví dụ: tz và os).
  + unstack() được dùng sau khi tổng hợp nhóm (ví dụ: dùng .size()) để xoay (pivot) cấp độ index cuối cùng thành cột mới, tạo ra một bảng tổng hợp (cross-tabulation) gọn gàng để so sánh (ví dụ: so sánh số lượng Windows/Non-Windows trong mỗi múi giờ).

13.1.4. Ví dụ

**{**

**import json**

**import pandas as pd**

**import numpy as np**

**# 1. Tải và chuẩn bị**

**records = [json.loads(line) for line in open(path, encoding='utf-8')]**

**frame = pd.DataFrame(records)**

**# 2. Phân tích Múi giờ**

**clean\_tz = frame['tz'].fillna('Missing')**

**clean\_tz[clean\_tz == ''] = 'Unknown'**

**top\_10\_tz = clean\_tz.value\_counts()[:10]**

**# 3. Phân loại OS và Tổng hợp**

**cframe = frame[frame.a.notnull()]**

**cframe['os'] = np.where(cframe['a'].str.contains('Windows'), 'Windows', 'Not Windows')**

**by\_tz\_os = cframe.groupby(['tz', 'os']).size().unstack().fillna(0)**

**normed\_rows = by\_tz\_os.div(by\_tz\_os.sum(1), axis=0)**

**# normed\_rows.plot(kind='barh', stacked=True) # Trực quan hóa tỷ lệ phần trăm**

**}**

13.2. MOVIELENS 1M DATASET

**Nguồn:** Dữ liệu được cung cấp bởi GroupLens Research – chứa 1 triệu lượt đánh giá phim, được thực hiện bởi 6.000 người dùng trên 4.000 bộ phim.

13.2.1. Cấu trúc dữ liệu

* Bộ dữ liệu gồm 3 bảng:

|  |  |  |
| --- | --- | --- |
| **Bảng** | **File** | **Nội dung** |
| Người dùng | users.dat | ID, giới tính, tuổi, nghề nghiệp, mã vùng ZIP |
| Phim | movies.dat | ID, tên phim, thể loại |
| Đánh giá | ratings.dat | ID người dùng, ID phim, điểm đánh giá (1–5), thời gian |

13.2.2. Nạp dữ liệu vào pandas

Dùng pd.read\_table() để đọc file (dấu phân cách là “::”) :

A screen shot of a computer code

AI-generated content may be incorrect.

13.2.3. Kết hợp dữ liệu (merge)

Ta cần gộp 3 bảng thành một bảng duy nhất để dễ phân tích.



13.2.4. Tính trung bình đánh giá theo giới tính

Sử dụng pivot\_table:

A black and green text

AI-generated content may be incorrect.

13.2.5. Lọc phim có ít nhất 250 lượt đánh giá

Tránh nhiễu do phim ít người xem:

A screenshot of a computer code

AI-generated content may be incorrect.

13.2.6. Top phim được nữ giới đánh giá cao nhất

****

Ví dụ kết quả:

|  |  |  |
| --- | --- | --- |
| **Phim** | **F** | **M** |
| A Close Shave (1995) | 4.64 | 4.47 |
| The Wrong Trousers (1993) | 4.68 | 4.47 |

13.2.7. So sánh mức chênh lệch giữa nam và nữ

**A black text on a white background

AI-generated content may be incorrect.**

* Phim được **nữ giới thích hơn nam giới**:
  + Dirty Dancing (1987)
  + Grease (1978)
* Phim được **nam giới thích hơn nữ giới**:
  + The Good, The Bad and The Ugly (1966)
  + Dumb & Dumber (1994)

13.2.8. Đo mức độ “bất đồng ý kiến” (variance)

* Tính độ lệch chuẩn (standard deviation) của điểm đánh giá từng phim:

A screenshot of a computer code

AI-generated content may be incorrect.

* Phim có ý kiến chia rẽ nhất:
* Dumb & Dumber (1994)
* The Blair Witch Project (1999)
* Natural Born Killers (1994)

13.2.9. Phân tích theo thể loại (genre)

* Mỗi phim có thể nhiều thể loại → dùng .str.split("|") rồi explode:

A black and green text

AI-generated content may be incorrect.

* Kết hợp lại toàn bộ:



* Tính trung bình điểm theo **thể loại và nhóm tuổi**:

A screenshot of a computer

AI-generated content may be incorrect.

**→ Kết quả là bảng 2 chiều: hàng là *genre*, cột là *age group*.**

Ví dụ kết quả:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Thể loại | Tuổi 18 | Tuổi 25 | Tuổi 35 | Tuổi 45 | Tuổi 50+ |
| Action | 3.45 | 3.53 | 3.61 | 3.61 | 3.65 |
| Animation | 3.62 | 3.70 | 3.74 | 3.78 | 3.76 |
| Comedy | 3.46 | 3.49 | 3.56 | 3.63 | 3.65 |
| Drama | 3.72 | 3.73 | 3.78 | 3.87 | 3.93 |

13.3. US BABY NAMES 1880–2010

Nguồn gốc của dữ liệu đến từ **Cơ quan An sinh Xã hội Hoa Kỳ (U.S. Social Security Administration – SSA)**, nơi công bố thống kê tên trẻ sơ sinh được đăng ký hằng năm tại Mỹ từ năm **1880 đến 2010**.

Mỗi năm, SSA phát hành một tệp văn bản riêng biệt có định dạng: youXXXX.txt. với XXXX là năm của dữ liệu.

Dữ liệu phản ánh xu hướng đặt tên của người Mỹ qua hơn một thế kỷ, và được sử dụng rộng rãi trong các nghiên cứu về:

* Nhân khẩu học
* Ngôn ngữ học xã hội
* Phân tích xu hướng văn hóa

13.3.1. Cấu trúc của bộ dữ liệu

|  |  |  |  |
| --- | --- | --- | --- |
| **Cột** | **Tên trường** | **Kiểu dữ liệu** | **Mô tả** |
| 1 | name | string | Tên của trẻ được đăng ký trong năm đó |
| 2 | sex | char (M hoặc F) | Giới tính: Nam (M) hoặc Nữ (F) |
| 3 | births | integer | Số lượng trẻ được sinh mang tên này trong năm đó |

**Ví dụ vài dòng dữ liệu từ yob1880.txt:**

Mary,F,7065

John,M,9655

Anna,F,2604

13.3.2. Tiền xử lý và nạp dữ liệu

Mục tiêu của giai đoạn này là:

* Tập hợp tất cả các tệp dữ liệu của từng năm thành một bảng thống nhất.
* Bổ sung cột year để xác định năm tương ứng.
* Chuẩn hóa tên cột, định dạng kiểu dữ liệu và xử lý thiếu sót (nếu có).

13.3.3. Thực hiện nạp và hợp nhất dữ liệu

import pandas as pd

import numpy as np

import os

import matplotlib.pyplot as plt

# Đường dẫn tới thư mục dữ liệu trên Drive

data\_path = "/content/drive/MyDrive/dataset/babynames"

# Đọc toàn bộ dữ liệu từ 1880 -> 2010

pieces = []

for year in range(1880, 2011):

    path = os.path.join(data\_path, f"yob{year}.txt")

    frame = pd.read\_csv(path, names=["name", "sex", "births"])

    frame["year"] = year

    pieces.append(frame)

# Gộp tất cả thành 1 DataFrame

names = pd.concat(pieces, ignore\_index=True)

**Kết quả:**

Tổng số dòng dữ liệu: ≈ 1,8 triệu bản ghi.

Khoảng thời gian: 1880 → 2010 (131 năm).

Ba cột chính: name, sex, births, kèm thêm year.

13.3.4. Đánh giá chất lượng dữ liệu

* Thiếu dữ liệu:
  + Không có giá trị trống (vì mỗi dòng luôn có 3 giá trị).
  + Tuy nhiên, các năm đầu (1880–1890) có ít bản ghi hơn do giới hạn lưu trữ lịch sử.
* Kiểu dữ liệu:
* name: chuỗi ký tự (string).
* sex: ký tự đơn (M, F)
* births: số nguyên (int).
* year: số nguyên (int).
* Độ tin cậy:
* Dữ liệu do **SSA công bố chính thức**, được xác nhận là nguồn chuẩn học thuật, có thể sử dụng trong các nghiên cứu định lượng.
* Nhận xét:
* Bộ dữ liệu có tính **liên tục theo năm**, rất thuận lợi cho việc phân tích chuỗi thời gian (time-series).
* Việc tách riêng các file theo năm phản ánh cấu trúc dữ liệu dạng **longitudinal dataset** — thường thấy trong nghiên cứu xã hội học.
* Số lượng bản ghi lớn cho phép áp dụng các kỹ thuật **vectorization**, **ufuncs** và **groupby aggregation** trong NumPy/Pandas hiệu quả.

13.3.5. Kết luận

Sau khi tiền xử lý, dữ liệu US Baby Names được thống nhất thành một bảng chuẩn có dạng:

|  |  |  |  |
| --- | --- | --- | --- |
| **name** | **sex** | **births** | **year** |
| Mary | F | 7065 | 1880 |
| John | M | 9655 | 1880 |
| Anna | F | 2604 | 1880 |

Bộ dữ liệu này sẵn sàng để phân tích chuyên sâu, bao gồm:

* Xác định tên phổ biến nhất từng năm.
* So sánh xu hướng theo giới tính.
* Đánh giá mức độ đa dạng và biến động văn hóa đặt tên qua thời gian.

13.4. USDA FOOD DATABASE

Phần này hướng dẫn cách xử lý một bộ dữ liệu JSON có cấu trúc phức tạp (nested) để đưa về dạng bảng phẳng, thuận tiện cho việc phân tích.

13.4.1. Mục tiêu

* Đọc và làm phẳng (flatten) dữ liệu JSON lồng nhau.
* Hợp nhất thông tin từ các phần khác nhau của dữ liệu.
* Phân tích thành phần dinh dưỡng theo nhóm thực phẩm.
* Tìm ra các loại thực phẩm giàu một chất dinh dưỡng cụ thể nhất.

13.4.2. Dữ liệu

* **Định dạng:** Một tệp JSON duy nhất chứa một danh sách các đối tượng. Mỗi đối tượng đại diện cho một loại thực phẩm và có cấu trúc lồng nhau (ví dụ: nutrients là một danh sách các dictionary).
* **Bản database sẽ như như thế này:**

    {

        "id": 1008,

        "description": "Cheese, caraway",

        "tags": [],

        "manufacturer": "",

        "group": "Dairy and Egg Products",

        "portions": [

            {

                "amount": 1,

                "unit": "oz",

                "grams": 28.35

            }

        ],

        "nutrients": [

            {

                "value": 25.18,

                "units": "g",

                "description": "Protein",

                "group": "Composition"

            },

…

]}

13.4.3. Các bước thực hiện & Phân tích:

13.4.3.1. Đọc và làm phẳng dữ liệu

* Tải dữ liệu: Dùng thư viện json để tải toàn bộ tệp vào một danh sách các dictionary.

import json

db = json.load(open("/content/drive/My Drive/Colab Notebooks/Lab5/data/database.json"))

len(db)

Out: 6636

* **Trích xuất thông tin chung**: Tạo DataFrame info từ các key cấp cao nhất như description, group, id, manufacturer.

db[0].keys()

db[0]["nutrients"][0]

nutrients = pd.DataFrame(db[0]["nutrients"])

nutrients.head(7)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **value** | **units** | **description** | **group** |
| **0** | 25.18 | g | Protein | Composition |
| **1** | 29.20 | g | Total lipid (fat) | Composition |
| **2** | 3.06 | g | Carbohydrate, by difference | Composition |
| **3** | 3.28 | g | Ash | Other |
| **4** | 376.00 | kcal | Energy | Energy |
| **5** | 39.28 | g | Water | Composition |
| **6** | 1573.00 | kJ | Energy | Energy |

**Nhận xét:**

* Tạo một DataFrame từ danh sách các chất dinh dưỡng của thực phẩm đầu tiên.
* Mỗi dòng trong DataFrame là một chất dinh dưỡng, mỗi cột là một thuộc tính (ví dụ: nutrient\_id, name, unit, value).
* Hiển thị 7 dòng đầu tiên của DataFrame nutrients.
* Giúp xem nhanh các chất dinh dưỡng đầu tiên và cấu trúc dữ liệu.

info\_keys = ["description", "group", "id", "manufacturer"]

info = pd.DataFrame(db, columns=info\_keys)

info.head()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **description** | **group** | **id** | **manufacturer** |
| **0** | Cheese, caraway | Dairy and Egg Products | 1008 |  |
| **1** | Cheese, cheddar | Dairy and Egg Products | 1009 |  |
| **2** | Cheese, edam | Dairy and Egg Products | 1018 |  |
| **3** | Cheese, feta | Dairy and Egg Products | 1019 |  |
| **4** | Cheese, mozzarella, part skim milk | Dairy and Egg Products | 1028 |  |

**Nhận xét:**

* Lấy các trường sẽ tương ứng các cột: "description", "group", "id", "manufacturer".
* Tạo một DataFrame từ danh sách db, chỉ lấy các trường được liệt kê trong info\_keys.
* Hiển thị 5 dòng đầu tiên của bảng info để xem nhanh nội dung.

info.info()

Out:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 6636 entries, 0 to 6635

Data columns (total 4 columns):

 #   Column        Non-Null Count  Dtype

---  ------        --------------  -----

 0   description   6636 non-null   object

 1   group         6636 non-null   object

 2   id            6636 non-null   int64

 3   manufacturer  5195 non-null   object

dtypes: int64(1), object(3)

memory usage: 207.5+ KB

**Nhận xét:**

* Tổng số dòng dữ liệu là 6636 dòng
* Có 4 cột:
  + description: tên thực phẩm, không thiếu dữ liệu (6636 non-null)
  + group: nhóm thực phẩm, cũng đầy đủ
  + id: mã số định danh, kiểu số nguyên (int64)
  + manufacturer: tên nhà sản xuất, chỉ có 5,195 giá trị, tức là còn thiếu 1,441 dòng => Điều này thường xảy ra khi một số thực phẩm không ghi rõ nhà sản xuất trong cơ sở dữ liệu.

**Kết luận:** Xử lý dữ liệu bị thiếu

* **Làm phẳng dữ liệu dinh dưỡng**:
  + Lặp qua từng thực phẩm trong danh sách db.
  + Với mỗi thực phẩm, chuyển danh sách nutrients của nó thành một DataFrame fnuts.
  + Thêm cột id của thực phẩm vào fnuts để có thể liên kết lại sau này.
  + Nối tất cả các DataFrame fnuts nhỏ lại thành một DataFrame nutrients lớn bằng pd.concat().

nutrients = []

for rec in db:

    fnuts = pd.DataFrame(rec["nutrients"])

    fnuts["id"] = rec["id"]

    nutrients.append(fnuts)

nutrients = pd.concat(nutrients, ignore\_index=True)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **value** | **units** | **description** | **group** | **id** |
| **0** | 25.180 | g | Protein | Composition | 1008 |
| **1** | 29.200 | g | Total lipid (fat) | Composition | 1008 |
| **2** | 3.060 | g | Carbohydrate, by difference | Composition | 1008 |
| **3** | 3.280 | g | Ash | Other | 1008 |
| **4** | 376.000 | kcal | Energy | Energy | 1008 |
| **...** | ... | ... | ... | ... | ... |
| **389350** | 0.000 | mcg | Vitamin B-12, added | Vitamins | 43546 |
| **389351** | 0.000 | mg | Cholesterol | Other | 43546 |
| **389352** | 0.072 | g | Fatty acids, total saturated | Other | 43546 |
| **389353** | 0.028 | g | Fatty acids, total monounsaturated | Other | 43546 |
| **389354** | 0.041 | g | Fatty acids, total polyunsaturated | Other | 43546 |

389355 rows × 5 columns

**Nhận xét:**

* 389,355 dòng: mỗi dòng là một chất dinh dưỡng cụ thể cho một loại thực phẩm.
* 5 cột: như mô tả ở trên.
* **Làm sạch**:
  + Kiểm tra và loại bỏ các hàng trùng lặp bằng nutrients.duplicated().sum() và nutrients.drop\_duplicates().

nutrients.duplicated().sum()  # number of duplicates

Out: np.int64(14179)

**Nhận xét:**

* Kiểm tra có 14179 dòng dữ liệu bị trùng lặp trong DataFrame nutrients
* Với tổng số dòng là 389,355, thì khoảng 3.6% dữ liệu bị trùng.

nutrients = nutrients.drop\_duplicates() # loại bỏ dữ liệu bị trùng lặp

**Nhận xét:**

* Giảm nhiễu trong phân tích dữ liệu
* Ngăn chặn dữ liệu bị trùng lặp khi tổng hợp
* Tăng tốc độ xử lý và trực quan hóa
* Đổi tên các cột bị trùng tên (description, group) ở cả hai DataFrame info và nutrients để tránh nhầm lẫn.

col\_mapping = {"description" : "food",

       "group"       : "fgroup"} # đổi tên cột tránh trùng tên với các cột cùng tên trong bảng nutrients

info = info.rename(columns=col\_mapping, copy=False) # để đổi tên các cột theo ánh xạ đã tạo

info.info()

Out:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 6636 entries, 0 to 6635

Data columns (total 4 columns):

 #   Column        Non-Null Count  Dtype

---  ------        --------------  -----

 0   food          6636 non-null   object

 1   fgroup        6636 non-null   object

 2   id            6636 non-null   int64

 3   manufacturer  5195 non-null   object

dtypes: int64(1), object(3)

memory usage: 207.5+ KB

**Nhận xét:**

* Hiển thị thông tin tổng quan về DataFrame info sau khi đổi tên.
* Giúp xác nhận rằng các cột đã được đổi tên thành công.

13.4.3.2. Hợp nhất và phân tích

* **Hợp nhất**: Dùng pd.merge() để kết hợp nutrients và info dựa trên cột id chung.

ndata = pd.merge(nutrients, info, on="id")

ndata.info()

Out:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 375176 entries, 0 to 375175

Data columns (total 8 columns):

 #   Column        Non-Null Count   Dtype

---  ------        --------------   -----

 0   value         375176 non-null  float64

 1   units         375176 non-null  object

 2   nutrient      375176 non-null  object

 3   nutgroup      375176 non-null  object

 4   id            375176 non-null  int64

 5   food          375176 non-null  object

 6   fgroup        375176 non-null  object

 7   manufacturer  293054 non-null  object

dtypes: float64(1), int64(1), object(6)

memory usage: 22.9+ MB

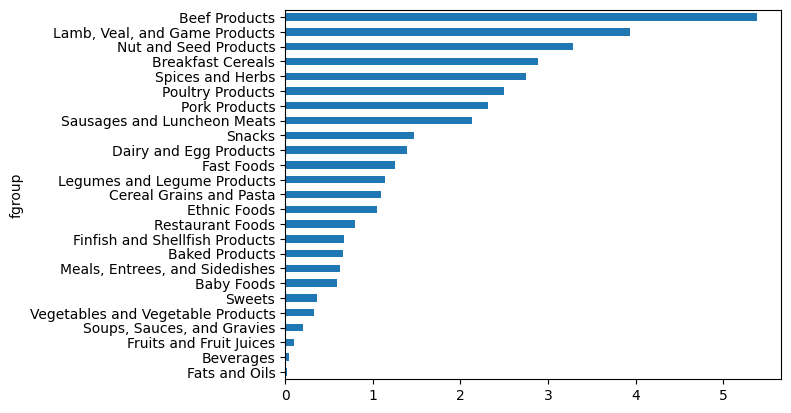
**Nhận xét:**

* Tổng số dùng dữ liệu là 375,176 dòng.
* Có 8 cột: value, units, nutrient, nutgroup, id, food, fgroup, manufacture.
* Cột manufacturer có giá trị thiếu: Chỉ có 293,054 giá trị không rỗng trên tổng số 375,176 → khoảng 22% dữ liệu bị thiếu.
* Bộ dữ liệu khá lớn: 25.8+ MB và hơn 375,000 dòng → cần cân nhắc hiệu suất khi xử lý tiếp theo (lọc, nhóm, trực quan hóa).
* Không có chỉ số thời gian: Nếu bạn định phân tích theo thời gian, cần bổ sung thêm cột thời gian nếu có.
* **Phân tích theo nhóm**:
  + Sử dụng groupby(["nutrient", "fgroup"])["value"].quantile(0.5) để tính giá trị trung vị (median) của mỗi chất dinh dưỡng trong mỗi nhóm thực phẩm.
  + Vẽ biểu đồ cột ngang để so sánh, ví dụ, lượng kẽm (Zinc, Zn) trong các nhóm thực phẩm.

result = ndata.groupby(["nutrient", "fgroup"])["value"].quantile(0.5)  # Vẽ đồ thị các giá trị trung bình theo nhóm thực phẩm và loại chất dinh dưỡng

result["Zinc, Zn"].sort\_values().plot(kind="barh")

Out: <Axes: ylabel='fgroup'>



Hình 13.4.3.2. Biểu đồ So sánh hàm lượng kẽm trung bình giữa các nhóm thực phẩm

**Nhận xét:**

* Nhóm Beef Products có hàm lượng kẽm cao nhất → nguồn cung cấp kẽm tốt.
* Nhóm Fats and Oils có hàm lượng thấp nhất → không phải nguồn kẽm lý tưởng.
* **Tìm thực phẩm có giá trị dinh dưỡng cao nhất**:
  + Nhóm dữ liệu theo nutgroup và nutrient.
  + Viết một hàm get\_maximum tùy chỉnh sử dụng x.loc[x.value.idxmax()] để tìm hàng có giá trị cao nhất trong mỗi nhóm.
  + Sử dụng .apply(get\_maximum) để áp dụng hàm này cho tất cả các nhóm chất dinh dưỡng và tìm ra loại thực phẩm tương ứng.

# Sử dụng idxmax và argmax series methods chạy trên Jupyter cell

by\_nutrient = ndata.groupby(["nutgroup", "nutrient"])

def get\_maximum(x):

    return x.loc[x.value.idxmax()]

max\_foods = by\_nutrient.apply(get\_maximum)[["value", "food"]]

# make the food a little smaller

max\_foods["food"] = max\_foods["food"].str[:50] # Rút gọn tên thực phẩm còn tối đa 50 ký tự để dễ hiển thị hoặc trực quan hóa

Out: /tmp/ipython-input-1825454003.py:6: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the operation. Either pass `include\_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to silence this warning.

  max\_foods = by\_nutrient.apply(get\_maximum)[["value", "food"]]

**Nhận xét:**

* Nhóm dữ liệu theo nhóm chất dinh dưỡng (nutgroup) và tên chất dinh dưỡng (nutrient).
* Mỗi nhóm sẽ chứa nhiều dòng tương ứng với các thực phẩm khác nhau.
* Hàm này tìm dòng có giá trị value lớn nhất trong mỗi nhóm → tức là thực phẩm chứa nhiều nhất chất dinh dưỡng đó.
* Áp dụng hàm get\_maximum cho từng nhóm → kết quả là một bảng gồm:
  + value: hàm lượng cao nhất
  + food: tên thực phẩm tương ứng

13.4.4. Các hàm/phương thức chính được sử dụng

* json.load(): Tải dữ liệu từ file JSON.
* pd.concat(): Nối các DataFrame.
* DataFrame.drop\_duplicates(): Loại bỏ hàng trùng lặp.
* DataFrame.rename(): Đổi tên cột.
* pd.merge(): Hợp nhất các DataFrame.
* DataFrame.groupby().apply(): Áp dụng hàm tùy chỉnh.
* Series.quantile(): Tính phân vị.
* Series.idxmax(): Tìm chỉ mục của giá trị lớn nhất.

13.5. 2012 Federal Election Commission Database

13.5.1. Khái niệm về Data Wrangling và Data Cleaning

* **DataFrames/Series:** Pandas là công cụ trung tâm, cho phép thực hiện các thao tác véc-tơ hóa (vectorized operations) trên toàn bộ cột (Series) hoặc bảng (DataFrame), giúp xử lý dữ liệu nhanh hơn so với vòng lặp Python thuần.
* **Lọc Dữ liệu (Filtering):** Sử dụng chỉ mục Boolean (Boolean Indexing) để trích xuất các hàng thỏa mãn điều kiện (ví dụ: fec.cmte\_id.isin(...) hoặc fec.contb\_receipt\_amt > 0).
* **Ánh xạ (Mapping):** Sử dụng phương thức **.map()** để tạo cột mới dựa trên ánh xạ các giá trị từ một từ điển (ví dụ: ánh xạ cmte\_id sang tên party).

13.5.2. Phân tích Dữ liệu Tổng hợp (Aggregation)

* **pivot\_table():** Là một công cụ mạnh mẽ để tính toán dữ liệu tổng hợp theo nhiều chiều. Trong phân tích FEC, nó được dùng để tính **tổng số tiền đóng góp** (aggfunc='sum') theo hai chiều: occupation (hàng) và party (cột).
* **sum(axis=1):** Được dùng để tính tổng ngang qua các cột (tức là tổng đóng góp của cả hai đảng cho cùng một nghề nghiệp), phục vụ cho việc lọc dữ liệu (chỉ giữ lại các nghề nghiệp đóng góp lớn).

13.5.3. Phân tích Chuỗi Thời gian (Time Series Analysis)

* **Kiểu dữ liệu datetime:** Dữ liệu ngày tháng ban đầu ở dạng chuỗi và cần được chuyển đổi thành kiểu dữ liệu **datetime64** của Pandas bằng hàm **pd.to\_datetime()**. Đây là bước bắt buộc để thực hiện các thao tác phân tích thời gian.
* **resample():** Đây là kỹ thuật cốt lõi trong phân tích chuỗi thời gian của Pandas.
  + Nó cho phép **thay đổi tần suất** (frequency conversion) của dữ liệu.
  + Trong code, **resample('M')** được dùng để nhóm dữ liệu lại theo tần suất **tháng** (M), sau đó thực hiện phép tổng (.sum()) để tính tổng đóng góp hàng tháng.
* **Biểu đồ Đường (Line Plot):** Là kỹ thuật trực quan hóa tiêu chuẩn để thể hiện **xu hướng** hoặc sự thay đổi của một đại lượng theo thời gian.

13.5.4. Ví dụ

{

import pandas as pd

# 1. Chuẩn bị dữ liệu

fec = pd.read\_csv('datasets/fec/P00000001-ALL.csv', low\_memory=False)

cand\_to\_party = {'C00431445': 'Obama', 'C00495828': 'Romney'}

fec\_mr\_contb = fec[fec.cmte\_id.isin(cand\_to\_party.keys())]

fec\_mr\_contb['party'] = fec\_mr\_contb.cmte\_id.map(cand\_to\_party)

fec\_mr\_contb = fec\_mr\_contb[fec\_mr\_contb.contb\_receipt\_amt > 0]

# 2. Phân tích Nghề nghiệp

by\_occupation = fec\_mr\_contb.pivot\_table(

    'contb\_receipt\_amt', index='occupation', columns='party', aggfunc='sum'

)

top\_occupations = by\_occupation[by\_occupation.sum(1) > 2000000] # Lọc > 2 triệu USD

# 3. Phân tích Thời gian

fec\_mr\_contb['contb\_receipt\_dt'] = pd.to\_datetime(fec\_mr\_contb['contb\_receipt\_dt'], format='%d-%b-%y')

monthly\_donations = fec\_mr\_contb.groupby(['contb\_receipt\_dt', 'party'])['contb\_receipt\_amt'].sum()

monthly\_donations = monthly\_donations.unstack(level='party').fillna(0).resample('M').sum()

# monthly\_donations.plot(kind='line') # Trực quan hóa xu hướng

}

13.6. Conclusion

A.1. ndarray Object Internals

1. Tổng quan

* ndarray là **đối tượng trung tâm của thư viện NumPy** — đại diện cho **một khối dữ liệu đồng nhất (homogeneously typed)**, có thể được tổ chức dưới dạng **mảng đa chiều (multidimensional array)**.
* Mỗi phần tử trong mảng đều có **cùng kiểu dữ liệu (dtype)**, ví dụ: float64, int32, bool, v.v.
* Điểm mạnh của ndarray là nó **không chỉ chứa dữ liệu**, mà còn chứa **siêu dữ liệu (metadata)** mô tả cách dữ liệu được lưu và truy cập trong bộ nhớ.

2. Cấu trúc nội bộ của ndarray

* Một đối tượng ndarray gồm 4 thành phần chính:

|  |  |
| --- | --- |
| Thành phần | Ý nghĩa |
| Data pointer | Con trỏ trỏ đến vùng dữ liệu thực sự trong bộ nhớ RAM hoặc file ánh xạ bộ nhớ (memory-mapped file). |
| dtype (data type) | Mô tả kiểu và kích thước cố định của từng phần tử, ví dụ mỗi phần tử float64 chiếm 8 byte. |
| shape (hình dạng) | Bộ tuple biểu diễn kích thước theo từng chiều, ví dụ (10, 5) cho mảng 10 hàng, 5 cột. |
| strides (bước nhảy) | Số byte cần di chuyển trong bộ nhớ để sang phần tử tiếp theo trên mỗi trục (dimension). |

***Hình A-1.*** *Minh họa rằng ndarray giống như một “lớp bọc” (view) thông minh quanh một khối dữ liệu gốc.*

3. Ví dụ minh họa

* Tạo mảng 10×5 toàn số 1:

A white background with black and white clouds

AI-generated content may be incorrect.

* Tạo mảng 3×4×5 kiểu float64 (mỗi phần tử 8 byte):

A black text with black letters

AI-generated content may be incorrect.

Giải thích:

* Trục 0 (chiều ngoài cùng): phải nhảy **160 byte** để đến phần tử kế tiếp.
* Trục 1: nhảy **40 byte**.
* Trục 2: nhảy **8 byte**.  
   → vì mỗi phần tử chiếm 8 byte, và mỗi hàng/khối chứa 5 hoặc 4 phần tử

**Kết luận:** Strides càng lớn → truy cập dữ liệu trên trục đó càng “xa” trong bộ nhớ → tính toán trên trục đó thường **chậm hơn**.

4. Strides âm và “zero-copy views”

* Điểm đặc biệt của NumPy:
* Các phép cắt mảng (slicing) như arr[::2, ::-1] không sao chép dữ liệu mà chỉ tạo một view — tức là một ndarray mới có cùng con trỏ dữ liệu gốc nhưng strides thay đổi.
* Ví dụ:
* [::-1] → stride âm → mảng được duyệt ngược trong bộ nhớ.
* ::2 → stride gấp đôi → chỉ lấy các phần tử cách nhau 2 vị trí.

→ Đây là cơ chế giúp NumPy rất hiệu quả về bộ nhớ và tốc độ.

5. Phân cấp kiểu dữ liệu trong NumPy (Data Type Hierarchy)

NumPy có nhiều kiểu dữ liệu khác nhau (float16, float32, float64, int8, int32, v.v.).

Để kiểm tra xem một mảng thuộc nhóm kiểu nào, ta dùng hàm:



Ví dụ:

A screenshot of a computer code

AI-generated content may be incorrect.

Trong đó:

* np.integer là **lớp cha (superclass)** cho tất cả kiểu số nguyên.
* np.floating là **lớp cha** cho các kiểu số thực.

Lưu ý: Một số kiểu dữ liệu có dấu gạch dưới (như np.int\_, np.float\_) để tránh trùng với kiểu mặc định của Python.

A.2. Advanced Array Manipulation

1. Giới thiệu chung

* NumPy cung cấp nhiều cách xử lý mảng hơn hẳn so với việc chỉ dùng **chỉ số, cắt mảng (slicing)** hay **lọc Boolean**.
* Những kỹ thuật này rất hữu ích khi bạn cần **tự viết thuật toán xử lý dữ liệu tùy chỉnh**, vượt ra ngoài những gì có sẵn trong thư viện như **pandas**.

2. Thay đổi hình dạng mảng – reshape

* Cho phép chuyển đổi hình dạng (shape) của mảng mà không sao chép dữ liệu (zero-copy).
* Dữ liệu gốc trong bộ nhớ vẫn giữ nguyên — chỉ thay đổi cách “nhìn” (view) vào nó.

**A screenshot of a computer

AI-generated content may be incorrect.**

Kết quả:

A screenshot of a computer

AI-generated content may be incorrect.

* Lưu ý:
* Có thể dùng -1 để **tự động suy luận kích thước**:

A close-up of a computer screen

AI-generated content may be incorrect.

Kết quả:

A black and yellow text

AI-generated content may be incorrect.

3. Làm phẳng mảng – ravel() và flatten()

Hai hàm này giúp chuyển mảng nhiều chiều thành mảng 1 chiều.

|  |  |
| --- | --- |
| **Hàm** | **Đặc điểm** |
| ravel() | Trả về view nếu dữ liệu liên tục trong bộ nhớ (zero-copy) |
| flatten() | Luôn tạo bản sao (copy) dữ liệu mới |

4. Trật tự bộ nhớ: C Order vs FORTRAN Order

**C (Row-major order)**

* Mặc định trong NumPy.
* Dữ liệu của mỗi hàng được lưu liên tiếp trong bộ nhớ.

(phù hợp với ngôn ngữ C)

**FORTRAN (Column-major order)**

* Dữ liệu của mỗi cột lưu liên tiếp (phù hợp với Fortran, MATLAB).

Ví dụ:

A screenshot of a computer

AI-generated content may be incorrect.

**Khác biệt:**

* C order: duyệt **theo hàng trước, cột sau**.
* F order: duyệt **theo cột trước, hàng sau**.

5. Nối và chia mảng – Concatenation & Splitting

* **Nối mảng (Concatenate)**

A screenshot of a computer

AI-generated content may be incorrect.

* **Hàm tiện lợi:**

|  |  |
| --- | --- |
| **Hàm** | **Mô tả** |
| vstack() | ghép theo hàng (axis=0) |
| hstack() | ghép theo cột (axis=1) |
| column\_stack() | giống hstack nhưng chuyển mảng 1D thành cột 2D |
| dstack() | ghép theo “chiều sâu” (axis=2) |

* **Chia mảng – split()**

**A black and yellow text

AI-generated content may be incorrect.**

[1, 3] là chỉ số cắt mảng gốc thành 3 phần:

* từ 0–1
* từ 1–3
* từ 3–hết

6. Công cụ r\_ và c\_

* Hai đối tượng đặc biệt giúp nối mảng nhanh hơn:

A screenshot of a computer

AI-generated content may be incorrect.

* Cũng có thể dùng để tạo mảng từ lát cắt (slice):

A white background with black text

AI-generated content may be incorrect.

7. Lặp phần tử – repeat() và tile()

* **repeat()**
* Lặp từng phần tử trong mảng.

A screenshot of a computer

AI-generated content may be incorrect.

* Lặp theo trục:

**A screenshot of a computer

AI-generated content may be incorrect.**

* **tile()**
* Lặp cả mảng nhiều lần (như lát gạch):

A white background with a black border

AI-generated content may be incorrect.

⟹ mảng được nhân bản **3 lần theo hàng**, **2 lần theo cột**.

8. Chỉ số nâng cao – take() và put()

* **take()**
* Chọn phần tử theo chỉ số (tương đương indexing):

A screenshot of a computer

AI-generated content may be incorrect.

* Có thể chỉ định **axis**:

**A black and yellow text with black letters

AI-generated content may be incorrect.**

* **put()**
* Gán giá trị vào các chỉ số (theo mảng phẳng 1D):

A number and text on a white background

AI-generated content may be incorrect.

**Nhận xét:** put() không có tham số axis, luôn hoạt động trên phiên bản **1D phẳng (C order)** của mảng.

A.3. Broadcasting

1. Khái niệm cơ bản

* Broadcasting là cơ chế cho phép NumPy thực hiện phép toán giữa các mảng có kích thước khác nhau, bằng cách tự động mở rộng (broadcast) mảng nhỏ hơn để khớp kích thước với mảng lớn hơn.
* Nói cách khác: Nếu hai mảng không cùng shape, NumPy sẽ cố “kéo giãn” mảng nhỏ để có thể thực hiện phép toán phần tử tương ứng.

2. Ví dụ cơ bản: Nhân mảng với số vô hướng (scalar)

A black and yellow text

AI-generated content may be incorrect.

Kết quả:

A number on a white background

AI-generated content may be incorrect.

**Nhận xét:** Ở đây, **số 4** được **broadcast** lên toàn bộ 5 phần tử của arr.

3. Ví dụ thực tế: Chuẩn hóa dữ liệu (demeaning)

* Giả sử ta có ma trận 4×3:

A black text on a white background

AI-generated content may be incorrect.

* Muốn **trừ đi giá trị trung bình của từng cột** (tức là “demean theo cột”):

A close-up of a smiley face

AI-generated content may be incorrect.

**Nhận xét:**

* arr.mean(0) trả về mảng kích thước (3,) (trung bình theo từng cột).
* NumPy **broadcast mảng (3,)** theo **axis 0** để trừ được cho mảng (4,3).
* Kết quả:

A close-up of a computer screen

AI-generated content may be incorrect.

A white and black rectangular box with white text

AI-generated content may be incorrect.

Hình A.3.1. Broadcasting qua trục 0 với mảng 1D.

4. Quy tắc Broadcasting

**Quy tắc cơ bản:**

* Hai mảng tương thích để broadcast nếu:
* Bắt đầu so sánh từ các chiều cuối cùng trở về đầu.
* Ở mỗi chiều, hai kích thước phải bằng nhau, hoặc một trong hai bằng 1.

➡ Khi đó, NumPy sẽ “kéo giãn” (mở rộng) mảng có kích thước 1 theo chiều tương ứng.

5. Ví dụ: Demean theo hàng (axis 1)

**A screenshot of a computer

AI-generated content may be incorrect.**

row\_means có shape (4,)  
**Nhận xét:** Sau khi reshape((4, 1)), mảng trở thành (4, 1)  
 → tương thích để broadcast theo axis 1 của arr (4,3).

**Kết quả:**

A number and numbers on a white background

AI-generated content may be incorrect.

A white sheet with black squares and numbers

AI-generated content may be incorrect.

*Hình A.3.2. Broadcasting over axis 1 of a 2D array.*

6. Broadcasting với mảng 3 chiều

Ví dụ: muốn demean theo trục 2 (depth) trong mảng 3D

A screenshot of a computer code

AI-generated content may be incorrect.

**Nhận xét:**

* np.newaxis thêm chiều mới để biến (3,4) → (3,4,1)
* Giúp broadcasting thực hiện **trừ trên trục 2 (depth)**

**Kết quả:**

**A white background with black text

AI-generated content may be incorrect.**

**A diagram of different colored cubes

AI-generated content may be incorrect.**

*Hình A.3.3. Compatible 2D array shapes for broadcasting over a 3D array.*

8.  Dùng np.newaxis để thêm chiều phục vụ broadcasting

A screenshot of a computer

AI-generated content may be incorrect.

Hoặc:

A screenshot of a computer code

AI-generated content may be incorrect.

**Nhận xét:** np.newaxis giúp **tạo mảng 1D thành mảng 2D** nhanh chóng, không cần reshape() phức tạp.

9. Viết hàm tổng quát cho demeaning trên mọi axis

A screenshot of a computer code

AI-generated content may be incorrect.

**Nhận xét:**

* Hàm này **tự động tạo indexer** phù hợp với số chiều của mảng (arr.ndim).
* Có thể dùng để **demean theo bất kỳ trục nào** trong mảng N chiều.

10.  Broadcasting khi gán giá trị (assignment)

Broadcasting không chỉ áp dụng trong phép toán, mà còn khi gán giá trị:

A close-up of a number

AI-generated content may be incorrect.

➡ Toàn bộ phần tử đều trở thành 5.

Hoặc broadcast 1D sang 2D:

A number of numbers and letters

AI-generated content may be incorrect.

Kết quả:

A group of orange and black squares

AI-generated content may be incorrect.

Có thể gán từng hàng bằng mảng con:

A number and text on a white background

AI-generated content may be incorrect.

**Quy tắc broadcasting**

|  |  |  |
| --- | --- | --- |
| **Tình huống** | **Shape mảng nhỏ cần có** | **Giải thích** |
| Demean theo cột | (1, n) hoặc (n,) | broadcast theo axis 0 |
| Demean theo hàng | (m, 1) | broadcast theo axis 1 |
| Demean theo depth (3D) | (a, b, 1) | broadcast theo axis 2 |
| Gán 1D vào 2D | (m, 1) hoặc (1, n) | tự động lặp giá trị theo chiều còn lại |

A.4. Advanced ufunc Usage

1. Giới thiệu và nạp dữ liệu

* Phần này tập trung việc sử dụng các hàm ufunc nâng cao trong Numpy để xử lý dữ liệu. Các thao tác bao gồm nạp dữ liệu, thống kê số births, và phân tích dữ liệu theo giới tính.
* Mục tiêu:
* Làm quen với thao tác trên mảng NumPy và ufunc.
* Kết hợp dữ liệu từ nhiều năm.
* Xử lý, lọc và tính toán tổng births theo giới tính.

2. Tiền xử lý và tổ chức dữ liệu

* Các tệp dữ liệu hàng năm có định dạng yobYYYY.txt được đọc tuần tự bằng **pandas.read\_csv()**, sau đó gộp lại bằng **pd.concat()** để tạo thành một DataFrame thống nhất chứa bốn cột: name, sex, births, và year.
* Mục tiêu của bước này là tạo nền tảng dữ liệu liên tục theo thời gian, phục vụ cho các phép tổng hợp và phân tích sau đó.
* Cách tiếp cận này tương đương với quy trình *data ingestion pipeline* trong thực tế.

3. Tổng hợp và trực quan hóa số lượng sinh

* Hàm **pivot\_table()** được sử dụng để nhóm và tổng hợp số lượng sinh (births) theo từng năm và giới tính. Kết quả trả về một bảng tổng hợp, trong đó chỉ số dòng là year và cột là sex.
* Tiếp đó, dữ liệu được trực quan hóa bằng **matplotlib.pyplot.plot()** giúp thể hiện xu hướng sinh theo thời gian. Quan sát biểu đồ cho thấy số lượng sinh tăng mạnh trong thế kỷ XX và thường có nhiều bé trai hơn bé gái.

4. Chuẩn hóa và phân tích dữ liệu

4.1. Tính tỷ lệ phần trăm từng tên (prop)

* Để so sánh tên trong cùng nhóm (năm, giới tính), ta cần tính tỷ lệ phần trăm (prop) cho từng tên. Hàm **apply()** được kết hợp với một hàm tự định nghĩa add\_prop() có nội dung như sau:
* Chuyển cột births thành mảng NumPy (to\_numpy()).
* Tính tỷ lệ bằng phép chia vector hóa births / births.sum() (ứng dụng ufunc np.divide()).
* Công thức:

***prop = births của từng tên / tổng births của nhóm (năm, giới tính)***

* prop = births của từng têntổng births của nhóm (năm, giới tính)
* Tổng các tỷ lệ trong mỗi nhóm (year, sex) bằng 1, đảm bảo tính chuẩn hóa.

4.2. Ý nghĩa học thuật

* Thay vì sử dụng vòng for, NumPy xử lý toàn bộ cột dữ liệu cùng lúc, giảm thời gian tính toán hàng trăm lần.
* Đây là đặc trưng cốt lõi của *ufunc*.

5. Phân tích và lọc dữ liệu

* Hàm get\_top1000() được xây dựng để sắp xếp (sort\_values()) các tên theo births và chọn ra 1000 tên đầu tiên. Hàm này được áp dụng trên từng nhóm (year, sex) bằng **groupby().apply()**.
* **Ý nghĩa:** Tập Top 1000 là cơ sở để nghiên cứu xu hướng tên gọi, vì nó chiếm phần lớn tổng số trẻ sinh mà vẫn đảm bảo tính đại diện.
* **Kiểm tra tính đúng đắn của tỷ lệ:**
* Để xác nhận phép chuẩn hóa đúng, hàm **groupby().sum()** được dùng để cộng tất cả prop trong từng nhóm. Kết quả mong đợi là tổng bằng 1 (hoặc rất gần 1).
* Nếu sai lệch xuất hiện, nguyên nhân thường do lỗi dữ liệu hoặc trùng lặp khi hợp nhất file.
* **Đo độ đa dạng bằng np.cumsum() và np.searchsorted()**
* Độ đa dạng tên được đo bằng cách xác định số lượng tên tối thiểu chiếm 50% tổng số trẻ sinh.
* Quy trình:
* Sắp xếp prop giảm dần.
* Dùng **np.cumsum()** để tính tổng tích lũy (ufunc cộng dồn).
* Dùng **np.searchsorted()** để tìm vị trí đầu tiên có tổng ≥ 0.5.
* Giá trị thu được thể hiện mức độ tập trung tên:
* Nhỏ → tên bị “thống trị” bởi vài lựa chọn phổ biến.
* Lớn → tên phân tán, xã hội đa dạng hơn.

6. Kiểm soát giá trị cực trị bằng np.clip()

* Một số tên có số lượng sinh vượt ngưỡng 10.000, khiến biểu đồ phân bố bị lệch. Hàm **np.clip()** được dùng để giới hạn cột births trong khoảng [0, 10000], đồng thời tạo thêm cột births\_clipped để so sánh.
* Sau đó, biểu đồ **plt.hist()** được vẽ để so sánh phân bố trước và sau khi giới hạn. Kết quả cho thấy giá trị ngoại lệ được kiểm soát, giúp biểu đồ cân đối hơn mà không làm mất thông tin chính.

6.1. Biến đổi và phân loại dữ liệu

* Biến đổi dữ liệu với np.where():
* Cột is\_female được tạo ra bằng **np.where(names\_clean["sex"] == "F", 1, 0)**, giúp mã hóa giới tính thành dạng số nhị phân. Đây là bước quan trọng khi muốn thực hiện phân tích định lượng hoặc huấn luyện mô hình thống kê sau này.
* Tỷ lệ tên nữ trong top 1000 được tính bằng **groupby()** kết hợp với **lambda**.  
  Biểu đồ xu hướng cho thấy sự biến động của tỷ lệ tên nữ qua thời gian.

6.2. Phân loại mức độ phổ biến với np.select():

* Để đánh giá mức độ phổ biến của tên, ba điều kiện được thiết lập dựa trên giá trị prop:
* prop > 0.02 → *Popular*
* 0.005 < prop ≤ 0.02 → *Normal*
* prop ≤ 0.005 → *Rare*
* Hàm **np.select()** được sử dụng để gán nhãn cho toàn bộ mảng dữ liệu cùng lúc (thay vì duyệt từng dòng). Sau đó, **value\_counts()** và **plot(kind="bar")** được dùng để thống kê và biểu diễn tỉ lệ các nhóm.

7. Hiệu năng và tối ưu hóa

* Một thử nghiệm nhỏ được thực hiện:
* Cộng dồn 1 triệu phần tử bằng vòng for.
* So sánh với kết quả từ **np.sum()**.
* Kết quả cho thấy np.sum() nhanh hơn hàng chục lần do được tối ưu bằng C và SIMD vectorization.

A.5. Structured and Record Arrays

5.1. Structured Array

5.1.1. Mục tiêu

* Mục tiêu của phần này là tìm hiểu và ứng dụng **Structured Array** trong NumPy — một cấu trúc dữ liệu mạnh mẽ cho phép lưu trữ nhiều loại dữ liệu (chuỗi, ký tự, số nguyên, số thực, v.v.) trong cùng một mảng.
* Structured Array (mảng có cấu trúc) hoạt động tương tự như **bảng dữ liệu hoặc bản ghi (record)** trong cơ sở dữ liệu, nhưng vẫn giữ được hiệu năng xử lý cao của NumPy.

5.1.2. Định nghĩa kiểu dữ liệu (dtype)

* Structured Array được tạo bằng cách xác định rõ **kiểu dữ liệu (dtype)** của từng trường (field).
* Mỗi phần tử trong mảng là một bản ghi có các trường: name, sex, births, và year.
* Cấu trúc này được mô tả bằng danh sách các bộ dữ liệu (tuple):
* 'name': Chuỗi Unicode tối đa 10–20 ký tự ('U10', 'U20')
* 'sex': Ký tự giới tính ('U1')
* 'births': Số nguyên 4 byte ('i4')
* 'year': Số nguyên 4 byte ('i4')
* Việc định nghĩa dtype rõ ràng giúp NumPy quản lý bộ nhớ chặt chẽ hơn và tăng tốc độ truy cập từng trường dữ liệu.

Hàm được sử dụng: np.array(data, dtype=dtype)

5.1.3. Tạo Structured Array mẫu

* Một mảng structured mẫu được khởi tạo trực tiếp từ danh sách các bản ghi như:  
   ('Mary', 'F', 7065, 1880), ('John', 'M', 9655, 1880),...
* Structured array này đóng vai trò giới thiệu cách NumPy lưu trữ dữ liệu có nhiều loại khác nhau trong cùng một khối bộ nhớ.
* Khi in ra, mỗi bản ghi xuất hiện giống như một dòng dữ liệu trong bảng.
* **Nhận xét:**
* Structured array là cầu nối giữa mảng số học thuần túy và cấu trúc dữ liệu dạng bảng như DataFrame.
* Nhờ khả năng định kiểu, nó đặc biệt hữu ích trong các bài toán xử lý dữ liệu dạng hỗn hợp (heterogeneous data).

5.1.4. Chuyển đổi từ DataFrame sang Structured Array

* Sau khi hiểu cấu trúc cơ bản, dữ liệu thực tế từ **DataFrame names** (đã nạp trong phần A.4) được chọn ngẫu nhiên bằng hàm **sample()** và chuyển đổi sang Structured Array.
* Quy trình:

1. Trích xuất 10 dòng ngẫu nhiên gồm các cột name, sex, births, và year.
2. Chuyển từng dòng thành tuple bằng **itertuples()**.
3. Tạo Structured Array mới bằng **np.array(..., dtype=dtype)**.

* **Ý nghĩa:**
* Bước này cho phép khai thác sức mạnh NumPy để làm việc trực tiếp trên dữ liệu đã được làm sạch trong pandas.
* Structured array cung cấp tốc độ truy cập nhanh và tối ưu bộ nhớ hơn so với DataFrame.

5.1.5. Kiểm tra cấu trúc dữ liệu và kiểu dtype

* Hàm **structured\_data.dtype** cho biết định nghĩa chi tiết của từng trường trong mảng: tên trường, kiểu dữ liệu và kích thước bộ nhớ.
* Việc kiểm tra dtype giúp đảm bảo rằng quá trình chuyển đổi dữ liệu không làm mất thông tin hoặc sai kiểu.
* Hàm liên quan: structured\_data.dtype, structured\_data.shape

5.1.6. Truy cập từng trường (field access)

* Structured array cho phép truy cập trực tiếp tới từng cột dữ liệu bằng cú pháp như:
* structured\_data['name']
* structured\_data['sex']
* structured\_data['births']
* Cách truy cập này tương tự việc chọn cột trong pandas, nhưng được thực hiện ở cấp độ NumPy nên nhanh hơn đáng kể.

5.1.7. Phân tích

* Nhờ khả năng gán kiểu tĩnh, các phép truy cập này diễn ra tức thì mà không cần tra cứu cột như trong DataFrame.
* Điều này giúp Structured Array phù hợp cho xử lý dữ liệu lớn trong khoa học dữ liệu và mô phỏng (simulation).

5.1.8. Lọc dữ liệu có điều kiện

Structured Array hỗ trợ các phép lọc logic tương tự như pandas.  
Ví dụ:

* structured\_data[structured\_data['births'] > 5000] → lọc các tên có hơn 5000 lượt sinh.
* structured\_data[(structured\_data['sex'] == 'F') & (structured\_data['births'] > 3000)] → lọc tên nữ có trên 3000 lượt sinh.

**Hàm được sử dụng:** Các toán tử so sánh (>, <, ==) và phép logic (&, |) được **vector hóa** — xử lý đồng thời toàn bộ phần tử của mảng.

**Nhận xét:** Khả năng lọc vector hóa này là một trong những ưu điểm lớn của NumPy, cho phép xử lý điều kiện hàng loạt mà không cần vòng lặp.

5.1.9. Thực hiện tính toán thống kê

Structured Array cho phép áp dụng các phép toán thống kê trực tiếp trên từng trường.  
Ví dụ, tính trung bình số lượng sinh theo giới tính bằng cách sử dụng:

* structured\_data['births'][structured\_data['sex'] == 'F'].mean()
* structured\_data['births'][structured\_data['sex'] == 'M'].mean()

**Phân tích:**Phép tính này thể hiện sức mạnh của NumPy trong việc thực hiện thao tác thống kê nhanh mà không cần chuyển đổi cấu trúc dữ liệu.  
Các phép như sum(), mean(), std() đều được tối ưu hóa ở mức C, giúp giảm đáng kể thời gian tính toán.

5.1.10. Ý nghĩa học thuật của Structured Array

Structured Array kết hợp linh hoạt giữa:

* Tính hiệu quả của mảng thuần NumPy, và
* Tính mô tả rõ ràng của dữ liệu có cấu trúc.

Nhờ đó, nó trở thành công cụ trung gian giữa **mảng số học** (numeric arrays) và **cấu trúc bảng** (DataFrame).  
 Cấu trúc này thường được dùng trong:

* Xử lý dữ liệu mô phỏng khoa học.
* Đọc/ghi dữ liệu từ file nhị phân (.bin, .dat).
* Truyền dữ liệu hiệu quả giữa Python và ngôn ngữ hệ thống như C/C++.

**Kết luận phần Structured Array**

* Structured Array trong NumPy cung cấp cách lưu trữ và truy cập dữ liệu phức tạp mà vẫn duy trì tốc độ xử lý cao.
* Qua ví dụ với bộ dữ liệu “US Baby Names”, ta thấy rõ rằng:

1. Structured Array giúp quản lý nhiều loại dữ liệu trong cùng mảng.
2. Cho phép thao tác, lọc và tính toán vector hóa trực tiếp mà không cần DataFrame.
3. Là nền tảng cho các cấu trúc phức tạp hơn như **Record Array** — sẽ được trình bày trong phần tiếp theo.

5.2. Record Array

5.2.1. Giới thiệu

* **Record Array** (hay còn gọi là **recarray**) là một biến thể mở rộng của **Structured Array** trong NumPy.
* Điểm khác biệt chính giữa hai loại này là cách truy cập các trường dữ liệu (fields):
* Với **Structured Array**, ta truy cập bằng cú pháp: structured\_data['name'].
* Với **Record Array**, ta có thể truy cập ngắn gọn hơn bằng dấu chấm: record\_data.name.
* Cách truy cập này giúp code trở nên trực quan, dễ đọc và gần gũi với cú pháp của đối tượng trong Python.

**Hàm khởi tạo:** np.rec.array(structured\_data)

* Hàm này chuyển đổi một structured array hiện có thành record array tương ứng, giữ nguyên toàn bộ kiểu dữ liệu (dtype) và nội dung.
  + 1. Khởi tạo Record Array:
* Đầu tiên, dữ liệu có cấu trúc (structured\_data) được chuyển đổi sang dạng record array bằng hàm **np.rec.array()**.
* Bản chất của quá trình này là tạo ra một “view” mới (không nhân đôi dữ liệu trong bộ nhớ), giúp người dùng truy cập dữ liệu một cách thuận tiện hơn.
* **Nhận xét:** Record Array thừa hưởng toàn bộ thông tin về dtype, cấu trúc, và dữ liệu gốc của Structured Array, do đó ta có thể áp dụng tất cả các phép toán NumPy tương tự mà không cần chuyển đổi lại.

5.2.3. Kiểm tra kiểu dữ liệu

* Sau khi chuyển đổi, hàm **type(record\_data)** trả về <class 'numpy.recarray'>, xác nhận rằng dữ liệu đã được chuyển sang dạng record array.
* Khi in ra, các trường dữ liệu (name, sex, births, year) được hiển thị đầy đủ.  
   Hàm **record\_data.dtype** cho biết cấu trúc kiểu dữ liệu vẫn giữ nguyên so với structured array ban đầu.
* Truy cập dữ liệu bằng cú pháp thuộc tính
* Điểm khác biệt nổi bật nhất của record array là khả năng **truy cập các trường dữ liệu bằng dấu chấm (.)** thay vì dùng cú pháp ['field'].
* Ví dụ:
* record\_data.name trả về danh sách tất cả tên.
* record\_data.sex trả về giới tính tương ứng.
* record\_data.births.sum() tính tổng số trẻ sinh trong toàn bộ mảng.
* record\_data.year[0] lấy năm của phần tử đầu tiên.
* **Ý nghĩa:**
* Cú pháp này không chỉ ngắn gọn mà còn tăng khả năng đọc hiểu mã nguồn, đặc biệt khi làm việc với nhiều trường dữ liệu khác nhau.
* Nó khiến Record Array trở nên thân thiện với người lập trình Python hơn so với Structured Array truyền thống.

5.2.4. Lọc dữ liệu theo điều kiện

* Record Array hỗ trợ đầy đủ các phép toán logic và lọc dữ liệu như Structured Array, chỉ khác ở cú pháp gọn hơn:
* Lọc tên có births > 5000: record\_data[record\_data.births > 5000]
* Lọc tên nữ có births > 3000: record\_data[(record\_data.sex == 'F') & (record\_data.births > 3000)]
* Các điều kiện này được **vector hóa**, nghĩa là NumPy sẽ xử lý đồng thời toàn bộ mảng dữ liệu, giúp thao tác lọc diễn ra nhanh và hiệu quả hơn nhiều so với vòng lặp truyền thống.
* **Nhận xét:** Khả năng kết hợp các điều kiện logic song song là đặc trưng mạnh mẽ của NumPy, giúp record array thích hợp cho phân tích dữ liệu thống kê và mô phỏng có điều kiện phức tạp.

5.2.5. Tính toán thống kê theo nhóm

* Tương tự Structured Array, Record Array cho phép thực hiện các phép toán thống kê trực tiếp trên từng trường.
* Trong phần này, trung bình số lượng sinh (births) theo giới tính được tính như sau:
* Trung bình nữ: record\_data.births[record\_data.sex == 'F'].mean()
* Trung bình nam: record\_data.births[record\_data.sex == 'M'].mean()
* Sau khi so sánh, kết quả được diễn giải để xác định giới tính nào có số lượng sinh trung bình cao hơn.
* **Phân tích:**
* Do dữ liệu đã được tách trường rõ ràng, NumPy có thể áp dụng các hàm thống kê như sum(), mean(), std() trực tiếp mà không cần ép kiểu hoặc chuyển đổi.
* Tốc độ xử lý của các phép toán này nhanh hơn đáng kể so với pandas DataFrame khi dữ liệu có kích thước lớn.

5.2.6. Truy cập từng phần tử

* Record Array cho phép truy cập nhanh đến từng phần tử hoặc trường cụ thể bằng chỉ số:
* record\_data.name[0] → tên đầu tiên trong mảng.
* record\_data.sex[0] → giới tính tương ứng.
* record\_data.births[0] → số lượng trẻ sinh tương ứng.
* Cách này giúp người dùng dễ dàng kiểm tra hoặc trích xuất dữ liệu tại một vị trí cụ thể mà không cần dùng slicing phức tạp.
* So sánh thống kê tổng hợp
* Kết quả thống kê được phân tích để rút ra kết luận:
* Nếu mean\_m > mean\_f, điều đó cho thấy bé trai có xu hướng được sinh ra nhiều hơn.
* Ngược lại, nếu mean\_f > mean\_m, các tên nữ phổ biến hơn trong năm đó.
* **Ý nghĩa:**
* Việc kết hợp Record Array và các phép toán vector hóa cho phép tạo ra các chỉ số thống kê có ý nghĩa mà không cần sử dụng các thư viện cao hơn.
* Đây là nền tảng cho các bài toán phân tích giới tính, độ phổ biến và xu hướng tên gọi.

5.2.7. Ưu điểm và ứng dụng của Record Array

* Record Array có các ưu điểm nổi bật:

1. **Cú pháp ngắn gọn, thân thiện**: sử dụng dấu chấm thay vì ngoặc vuông.
2. **Hiệu năng cao**: tận dụng toàn bộ cơ chế tối ưu của NumPy ufunc.
3. **Tính linh hoạt**: có thể dùng cho dữ liệu hỗn hợp, tương thích tốt với Structured Array.
4. **Tích hợp với C/C++**: dữ liệu được tổ chức tuyến tính trong bộ nhớ, dễ dàng truyền cho các thư viện ngoài.

* **Ứng dụng:**
* Phân tích dữ liệu thống kê có cấu trúc cố định (như hồ sơ dân cư, dữ liệu thời tiết, y tế).
* Tiền xử lý dữ liệu trước khi đưa vào mô hình học máy.
* Xử lý dữ liệu nhị phân hoặc định dạng cố định (fixed-width binary).

5.2.8. Kết luận phần Record Array

* Record Array là bước phát triển trực quan hơn của Structured Array. Nó cho phép:
* Truy cập dữ liệu bằng cú pháp ngắn gọn (record.field) thay vì record['field'].
* Giữ nguyên hiệu năng của NumPy và hỗ trợ đầy đủ các thao tác vector hóa.
* Dễ dàng tích hợp với các mô hình xử lý dữ liệu phức tạp, đồng thời giúp mã nguồn rõ ràng và dễ bảo trì hơn.
* Từ góc độ học thuật, Record Array thể hiện sự kết hợp giữa **tính mô-đun của hướng đối tượng** và **hiệu năng xử lý mảng của NumPy**,  
  đem lại một mô hình dữ liệu mạnh mẽ, linh hoạt và dễ đọc trong các ứng dụng phân tích dữ liệu quy mô lớn.

A.6. More About Sorting

6.1. Giới thiệu về Sorting trong NumPy

Trong NumPy, sorting (sắp xếp) là một thao tác quan trọng giúp tổ chức, phân tích và trích xuất thông tin từ dữ liệu có cấu trúc.

Khác với pandas, NumPy làm việc trực tiếp trên mảng và không cần dùng sort\_values(). Thay vào đó, các hàm np.sort() và np.argsort() được tối ưu ở cấp độ C, giúp việc sắp xếp dữ liệu có cấu trúc như **structured array** hoặc **record array** trở nên cực kỳ nhanh và hiệu quả.

Các hàm chính được sử dụng gồm:

* **np.sort()**: Sắp xếp dữ liệu dựa trên giá trị của một hoặc nhiều trường (field).
* **np.argsort()**: Trả về **chỉ số (index)** thể hiện thứ tự sắp xếp, thay vì giá trị.
* **[::-1]**: Đảo ngược mảng, giúp dễ dàng thực hiện sắp xếp giảm dần.

Mục tiêu của phần này là hiểu cách NumPy thực hiện sắp xếp trên dữ liệu nhiều chiều và cách kết hợp linh hoạt các phương pháp để phân tích dữ liệu thực tế.

6.2. Sắp xếp cơ bản theo từng trường

* Ở bước đầu tiên, dữ liệu được sắp xếp **theo số lượng sinh (births)** bằng np.sort() với tham số order='births'.
* Phương thức này trả về một bản sao của mảng đã được sắp xếp theo thứ tự tăng dần, giúp quan sát phân bố tự nhiên của dữ liệu.
* Tương tự, việc sắp xếp **theo tên (name)** dựa trên thứ tự bảng chữ cái (alphabetical order) cho phép tổ chức lại danh sách tên một cách có hệ thống, hữu ích khi cần tra cứu hoặc xuất báo cáo thống kê.

**Ý nghĩa:**

* Việc sắp xếp theo một field giúp nhận diện nhanh các phần tử cực trị (ví dụ: tên phổ biến nhất hoặc ít gặp nhất).
* NumPy xử lý thao tác này bằng cơ chế vector hóa cấp thấp, đảm bảo tốc độ vượt trội so với Python thông thường.

6.3. Sắp xếp đa tiêu chí (multi-field sorting)

* Để phản ánh cấu trúc dữ liệu thực tế (vừa có giới tính, vừa có số sinh), NumPy cho phép sắp xếp theo nhiều trường cùng lúc bằng cú pháp:

***np.sort(arr, order=['sex', 'births'])***

* Ở đây, dữ liệu được sắp xếp trước theo **giới tính (sex)**, sau đó trong mỗi nhóm giới tính lại sắp xếp theo **số lượng sinh (births)**.

**Ý nghĩa:**

Cách sắp xếp này mô phỏng đúng các tình huống thực tế, ví dụ phân tích “Top tên phổ biến nhất trong từng giới tính”.

Cho phép xử lý **dữ liệu phân cấp (hierarchical data)** trực tiếp trong NumPy, không cần chuyển sang pandas.

6.4. Sắp xếp sử dụng chỉ số (np.argsort)

* Hàm np.argsort() không trả về dữ liệu đã sắp xếp mà trả về chỉ số vị trí của các phần tử theo thứ tự sắp xếp.
* Ví dụ: khi sắp xếp theo births, ta có thể dùng chỉ số này để truy cập lại dữ liệu gốc theo thứ tự tăng dần mà không tạo bản sao mới của mảng.

**Ứng dụng:**

* Giảm chi phí bộ nhớ khi cần sắp xếp mảng lớn.
* Dễ dàng tái sử dụng thứ tự sắp xếp cho các thao tác liên quan (ví dụ, đồng bộ hai mảng theo cùng thứ tự).

**Ý nghĩa:** Đây là ví dụ điển hình của indirect sorting (sắp xếp gián tiếp) – một kỹ thuật quan trọng trong xử lý dữ liệu lớn, giúp tối ưu hiệu năng mà vẫn bảo toàn cấu trúc gốc.

6.5. Sắp xếp giảm dần và lọc nhóm con

# Sắp xếp giảm dần theo births

sorted\_desc = np.sort(record\_data, order='births')[::-1]

print("Sắp xếp theo births (giảm dần):")

print(sorted\_desc[:5])

Để thực hiện sắp xếp giảm dần, ta chỉ cần đảo ngược kết quả tăng dần bằng cú pháp [::-1].

Ví dụ, khi sắp xếp giảm dần theo births, ta nhanh chóng xác định được **tên phổ biến nhất**.

Tiếp theo, bằng cách lọc dữ liệu theo giới tính (record\_data.sex == 'F' hoặc 'M') và áp dụng np.sort() riêng từng nhóm, ta có thể lấy ra Top 3 tên nữ và Top 3 tên nam phổ biến nhất.

**Ý nghĩa**

* Kết hợp giữa lọc điều kiện và sắp xếp vector hóa giúp khai thác dữ liệu sâu hơn mà không cần cấu trúc vòng lặp.
* Đây là kỹ thuật phổ biến trong phân tích thống kê xã hội học và nghiên cứu nhân khẩu học (demographic analysis).

6.6. Tổng kết

Phần này minh họa khả năng mạnh mẽ của NumPy trong việc sắp xếp và thao tác trên mảng có cấu trúc.

Các kỹ thuật chính gồm:

* Sắp xếp đơn tiêu chí và đa tiêu chí với np.sort().
* Sắp xếp gián tiếp bằng np.argsort().
* Kết hợp điều kiện lọc và đảo chiều mảng để trích xuất dữ liệu đặc thù.

So với pandas, NumPy cho phép thao tác trực tiếp ở mức mảng, mang lại tốc độ xử lý nhanh hơn và kiểm soát chi tiết hơn về bộ nhớ — một yếu tố quan trọng trong xử lý dữ liệu quy mô lớn.

A.7. Writing Fast NumPy Functions with Numba

Phần này giới thiệu **Numba**, một thư viện mã nguồn mở giúp tạo ra các hàm tính toán nhanh cho dữ liệu kiểu NumPy bằng cách sử dụng CPU, GPU hoặc các phần cứng khác. Numba sử dụng LLVM Project để dịch mã Python thành mã máy đã được biên dịch, giúp tăng tốc độ thực thi đáng kể.

7.1. Cách hoạt động

* **Biên dịch Just-in-Time (JIT)**: Bạn có thể biến một hàm Python thông thường thành một hàm Numba đã được biên dịch bằng cách sử dụng hàm numba.jit hoặc decorator @nb.jit.
* **Hiệu suất**: Một hàm đã được Numba biên dịch có thể chạy nhanh hơn đáng kể so với phiên bản Python thuần túy và thậm chí nhanh hơn cả phiên bản NumPy đã được vector hóa.
* **Chế độ nopython**: Để đảm bảo hiệu suất cao nhất, bạn có thể sử dụng tùy chọn nopython=True (hoặc decorator @njit) để yêu cầu Numba biên dịch hoàn toàn hàm mà không cần gọi đến API CPython. Điều này buộc bạn phải viết mã trong một tập hợp con của Python mà Numba hỗ "trợ".

import numba as nb

numba\_mean\_distance = nb.jit(mean\_distance)

@nb.jit

def numba\_mean\_distance(x, y):

  nx = len(x)

  result = 0.0

  count = 0

  for i in range(nx):

    result += x[i] - y[i]

    count += 1

    return result / count

%timeit numba\_mean\_distance(x, y)

Out: The slowest run took 28.73 times longer than the fastest. This could mean that an intermediate result is being cached.

4.9 µs ± 9.23 µs per loop (mean ± std. dev. of 7 runs, 1 loop each)

**Nhận xét:**

* Kết quả chí mất khoảng 387ns mỗi lần chạy.
* Numba phiên bản tối ưu nhanh hơn gần 100 lần bản trong trường hợp NumPy (x - y).mean().

# Dùng @njit để biên dịch hàm

from numba import float64, njit

@njit(float64(float64[:], float64[:]))

def mean\_distance(x, y):

  return (x - y).mean()

**Nhận xét:**

* Đoạn code này tạo ra một hàm mean\_distance được tối ưu hóa cao độ bằng Numba. Hàm này hoạt động trên toàn bộ mảng NumPy và sẽ chạy nhanh hơn nhiều so với phiên bản Python thuần túy.

7.2. Tạo các đối tượng ufunc tùy chỉnh

* Hàm numba.vectorize cho phép bạn tạo ra các ufunc (hàm phổ quát) đã được biên dịch, hoạt động tương tự như các ufunc tích hợp sẵn của NumPy.
* Ví dụ, bạn có thể tạo một ufunc để cộng hai số, và nó sẽ hoạt động trên toàn bộ mảng một cách hiệu quả.

from numba import vectorize

@vectorize

def nb\_add(x, y):

  return x + y

x = np.arange(10)

nb\_add(x, x)

Out: array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])

nb\_add.accumulate(x, 0)

Out: array([ 0,  1,  3,  6, 10, 15, 21, 28, 36, 45])

**Nhận xét:**

* @vectorize: Decorator này biến một hàm Python đơn giản (chỉ làm việc với các số vô hướng - scalar) thành một ufunc hiệu suất cao của NumPy.
* def nb\_add(x, y): return x + y: Lưu ý rằng hàm này rất đơn giản, nó chỉ cộng hai số x và y. Bạn không cần viết vòng lặp hay xử lý mảng ở đây. Numba sẽ tự động làm điều đó.
* nb\_add(x, x): Khi bạn gọi nb\_add với các mảng NumPy, ufunc mới tạo sẽ áp dụng phép toán cộng cho từng cặp phần tử tương ứng trong hai mảng.
  + x là [0, 1, 2, ..., 9]
  + Kết quả là [0+0, 1+1, 2+2, ..., 9+9] -> [0, 2, 4, ..., 18].
* nb\_add.accumulate(x, 0): Vì nb\_add bây giờ là một ufunc, nó có các phương thức đặc biệt như .accumulate(). Phương thức này thực hiện một phép toán tích lũy. Đối với phép cộng, nó chính là tính tổng tích lũy (cumulative sum).
  + Phần tử đầu tiên: 0
  + Phần tử thứ hai: 0 + 1 = 1
  + Phần tử thứ ba: 1 + 2 = 3
  + Phần tử thứ tư: 3 + 3 = 6 ....

A.8. Advanced Array Input and Output

Phần này thảo luận về các tùy chọn phức tạp hơn để lưu và tải dữ liệu mảng, đặc biệt là các tệp ánh xạ bộ nhớ (memory-mapped files), hữu ích cho việc xử lý các bộ dữ liệu không vừa với RAM.

8.1. Tệp ánh xạ bộ nhớ (Memory-Mapped Files)

* **Khái niệm**: Tệp ánh xạ bộ nhớ là một phương pháp tương tác với dữ liệu nhị phân trên đĩa như thể nó đang nằm trong một mảng trong bộ nhớ.
* **Đối tượng memmap của NumPy**: NumPy cung cấp đối tượng memmap, hoạt động giống như một ndarray, cho phép đọc và ghi các phần nhỏ của một tệp lớn mà không cần tải toàn bộ tệp vào bộ nhớ.
* **Cách tạo**: Sử dụng hàm np.memmap và cung cấp đường dẫn tệp, kiểu dữ liệu (dtype), hình dạng (shape), và chế độ tệp (mode).

mmap = np.memmap('mymmap', dtype='float64', mode='w+', shape=(10000, 10000)) # Tạo bản đồ bộ nhớ đến một mảng được lưu trữ trong tệp nhị phân trên đĩa.

mmap

Out: memmap([[0., 0., 0., ..., 0., 0., 0.],

               [0., 0., 0., ..., 0., 0., 0.],

               [0., 0., 0., ..., 0., 0., 0.],

                ...,

               [0., 0., 0., ..., 0., 0., 0.],

               [0., 0., 0., ..., 0., 0., 0.],

               [0., 0., 0., ..., 0., 0., 0.]])

* **Hoạt động**:
  + Việc cắt lát (slicing) một memmap sẽ trả về các "khung nhìn" (views) trên dữ liệu trên đĩa, không sao chép dữ liệu.
  + Khi bạn gán dữ liệu cho một lát cắt, các thay đổi sẽ được lưu vào bộ đệm trong bộ nhớ. Bạn cần gọi phương thức flush() để đồng bộ hóa các thay đổi này xuống đĩa.
  + Khi mở một memmap đã có, bạn vẫn cần chỉ định dtype và shape vì tệp chỉ là một khối dữ liệu nhị phân không có siêu dữ liệu này.

section[:] = rng.standard\_normal((5, 10000))

mmap.flush()

mmap

Out:

memmap([[ 0.52183871,  0.06485206,  0.09331012, ...,  1.63170909,

         -0.94138588, -0.29831689],

        [ 0.10194139,  0.13554111, -0.5722865 , ...,  1.20593654,

         -0.15919804,  0.9670215 ],

        [-0.49704329,  1.44833063,  0.749871  , ..., -0.38287281,

         -1.71018367, -0.41107318],

        ...,

        [ 0.        ,  0.        ,  0.        , ...,  0.        ,

          0.        ,  0.        ],

        [ 0.        ,  0.        ,  0.        , ...,  0.        ,

          0.        ,  0.        ],

        [ 0.        ,  0.        ,  0.        , ...,  0.        ,

          0.        ,  0.        ]])

del mmap

mmap = np.memmap('mymmap', dtype='float64', shape=(10000, 10000))

mmap

Out:

memmap([[ 0.52183871,  0.06485206,  0.09331012, ...,  1.63170909,

         -0.94138588, -0.29831689],

        [ 0.10194139,  0.13554111, -0.5722865 , ...,  1.20593654,

         -0.15919804,  0.9670215 ],

        [-0.49704329,  1.44833063,  0.749871  , ..., -0.38287281,

         -1.71018367, -0.41107318],

        ...,

        [ 0.        ,  0.        ,  0.        , ...,  0.        ,

          0.        ,  0.        ],

        [ 0.        ,  0.        ,  0.        , ...,  0.        ,

          0.        ,  0.        ],

        [ 0.        ,  0.        ,  0.        , ...,  0.        ,

          0.        ,  0.        ]])

**Nhận xét:**

* np.memmap(...) tạo một mảng ánh xạ bộ nhớ tên mymmap có kiểu dữ liệu float64.
* shape=(10000, 10000) → mảng có 100 triệu phần tử.
* Mảng này không nằm hoàn toàn trong RAM mà được ánh xạ trực tiếp tới file mymmap trên đĩa.
* Kết quả hiển thị cho thấy các giá trị ngẫu nhiên được lưu trong một phần mảng, còn phần lớn mảng là số 0 (chưa được ghi dữ liệu).

8.2. Giải phóng bộ nhớ và xóa file

%xdel mmap

!rm mymmap

**Nhận xét:**

* %xdel mmap là xóa hoàn toàn biến mmap khỏi kernel.
* !rm mymmap là xóa file mymmap khỏi ổ đĩa.

**Kết luận:** Đây là cách dọn dẹp tài nguyên sau khi sử dụng memmap.

8.3. HDF5 và các tùy chọn lưu trữ mảng khá

* **PyTables và h5py**: Đây là hai thư viện Python cung cấp giao diện thân thiện với NumPy để lưu trữ dữ liệu mảng ở định dạng HDF5 hiệu quả và có thể nén.
* **Ưu điểm**: HDF5 rất phù hợp để làm việc với các bộ dữ liệu không vừa với bộ nhớ, vì bạn có thể đọc và ghi hiệu quả các phần nhỏ của các mảng lớn hơn nhiều.
* **Lưu ý**: HDF5 không phải là một cơ sở dữ liệu. Nó phù hợp nhất cho các bộ dữ liệu "ghi một lần, đọc nhiều lần" (write-once, read-many). Nếu nhiều tiến trình ghi đồng thời, tệp có thể bị hỏng

A.9. Performance Tips

Phần này cung cấp các lời khuyên để đạt được hiệu suất tốt nhất khi sử dụng NumPy.

9.1. Các mẹo chính

* **Vector hóa**: Chuyển đổi các vòng lặp Python và logic điều kiện thành các phép toán trên mảng và các phép toán trên mảng Boolean.
* **Sử dụng broadcasting**: Tận dụng cơ chế broadcasting (truyền phát) bất cứ khi nào có thể để thực hiện các phép toán trên các mảng có hình dạng khác nhau một cách hiệu quả.
* **Sử dụng "views" (khung nhìn)**: Sử dụng các lát cắt (slicing) để tạo ra các "views" thay vì các bản sao (copies) của dữ liệu để tránh việc sao chép không cần thiết.
* **Tận dụng ufuncs**: Sử dụng các hàm ufunc và các phương thức của chúng (như reduce, accumulate) vì chúng được viết bằng C và rất nhanh.
* **Sử dụng các công cụ bên ngoài**: Nếu hiệu suất vẫn chưa đạt yêu cầu, hãy cân nhắc viết mã bằng C, FORTRAN, hoặc **Cython**. Cython thường được sử dụng để đạt được hiệu suất gần bằng C mà không tốn quá nhiều thời gian phát triển.

9.2. Tầm quan trọng của bộ nhớ liền kề (Contiguous Memory)

* **Bố cục bộ nhớ**: Cách một mảng được sắp xếp trong bộ nhớ có thể ảnh hưởng đáng kể đến tốc độ tính toán.
* **C order vs F order**:
  + Mặc định, các mảng NumPy được tạo theo thứ tự **C** (row major - hàng liền kề), nghĩa là các phần tử trong cùng một hàng được lưu trữ cạnh nhau trong bộ nhớ.
  + Thứ tự **FORTRAN** (column major - cột liền kề) có nghĩa là các giá trị trong mỗi cột được lưu trữ cạnh nhau.
* **Hiệu suất**: Các phép toán truy cập vào các khối bộ nhớ liền kề (ví dụ: tính tổng các hàng của một mảng theo thứ tự C) thường sẽ nhanh nhất vì hệ thống bộ nhớ có thể đệm các khối bộ nhớ thích hợp vào bộ đệm cache L1 hoặc L2 của CPU.
* **Kiểm tra và thay đổi**: Bạn có thể kiểm tra xem một mảng có liền kề hay không thông qua thuộc tính flags của nó. Nếu cần, bạn có thể tạo một bản sao của mảng với thứ tự bộ nhớ mong muốn bằng phương thức copy().

arr\_c = np.ones((100, 10000), order='C')  # Tạo hai mảng arr\_c, arr\_f

arr\_f = np.ones((100, 10000), order='F')

arr\_c.flags

Out:

 C\_CONTIGUOUS : True

  F\_CONTIGUOUS : False

  OWNDATA : True

  WRITEABLE : True

  ALIGNED : True

  WRITEBACKIFCOPY : False

arr\_f.flags

Out:

 C\_CONTIGUOUS : False

  F\_CONTIGUOUS : True

  OWNDATA : True

  WRITEABLE : True

  ALIGNED : True

  WRITEBACKIFCOPY : False