**TRƯỜNG ĐẠI HỌC SÀI GÒN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**\*-\*-\***



**Bài báo cáo môn : Phân tích dữ liệu**

***Lap02 chapter 13: Data Analysis Examples***

**Giảng viên : Đỗ Như Tài**

**Nhóm thực hiện:**

Nguyễn Thị Xuân Mai 3122410230

Trần Lê Đăng Khoa 3122410187

Lê Đức Nguyên Phú 3122410313

Mai Lê Mỹ Linh 3122410210

Thành phố Hồ Chí Minh - Tháng 10 năm 2025

**MỤC LỤC**

[LỜI MỞ ĐẦU 1](#_Toc211512867)

[LỜI CẢM ƠN 2](#_Toc211512868)

[BẢNG PHÂN CÔNG 3](#_Toc211512869)

[I. Giới thiệu chung 4](#_Toc211512870)

[II. Công cụ và phương pháp 4](#_Toc211512871)

[1. Ngôn ngữ và thư viện sử dụng 4](#_Toc211512872)

[2. Quy trình thực hiện 5](#_Toc211512873)

[3. Mục tiêu phương pháp 6](#_Toc211512874)

[III. Phân tích dữ liệu theo từng mục 6](#_Toc211512875)

[3.1. Dataset 1 – Bitly Data from 1.USA.gov 6](#_Toc211512876)

[3.1.1 Mục tiêu: 6](#_Toc211512877)

[3.1.2 Phân tích: 6](#_Toc211512878)

[3.1.3 Kết quả & nhận xét: 9](#_Toc211512879)

[3.2. Dataset 2 - MovieLens 1M Dataset 11](#_Toc211512880)

[3.2.1 Mục tiêu: 11](#_Toc211512881)

[3.2.2 Phân tích 11](#_Toc211512882)

[Kết quả & nhận xét 11](#_Toc211512883)

[3.3. Dataset 3 – US Baby Names 1880–2010 16](#_Toc211512884)

[3.3.1. Mục tiêu 16](#_Toc211512885)

[3.3.2. Quy trình thực hiện 16](#_Toc211512886)

[3.3.3. Phân tích và kết quả 17](#_Toc211512887)

[3.3.4. Nhận xét tổng hợp 22](#_Toc211512888)

[3.3.5. Kết luận 22](#_Toc211512889)

[3.4. Dataset 4 – USDA Food Database 23](#_Toc211512890)

[3.4.1. Mục tiêu 23](#_Toc211512891)

[3.4.2. Cơ sở dữ liệu USDA FoodData Central 23](#_Toc211512892)

[3.4.3. Quy trình thực hiện 23](#_Toc211512893)

[3.4.4. Phân tích và kết quả 26](#_Toc211512894)

[3.4.5. Thực phẩm giàu nhất theo từng chất dinh dưỡng 27](#_Toc211512895)

[3.4.6. Kết luận 28](#_Toc211512896)

[3.4. Dataset 5 – 2012 Federal Election Commission (FEC) Data 29](#_Toc211512897)

[3.5.1 Donation Statistics by Occupation and Employer 33](#_Toc211512898)

[3.5.2 Bucketing Donation Amounts 37](#_Toc211512899)

[IV. Appendix Advanced Numpy 41](#_Toc211512900)

[4.1 ndarray Object Internals 41](#_Toc211512901)

[4.1.1 Mục tiêu 41](#_Toc211512902)

[4.1.2 Nội dung & Giải thích chi tiết 41](#_Toc211512903)

[4.2 Advanced Array Manipulation 44](#_Toc211512904)

[4.2.1 Mục tiêu 44](#_Toc211512905)

[4.2.2 Các bước & Khái niệm chính 44](#_Toc211512906)

[4.2.3 Kết quả & nhận xét: 48](#_Toc211512907)

[4.3 Broadcasting 48](#_Toc211512908)

[4.4 Advanced ufunc Usage 49](#_Toc211512909)

[4.5 Structured and Record Arrays 49](#_Toc211512910)

[4.5.1. Mục tiêu 49](#_Toc211512911)

[4.5.2. Cơ sở lý thuyết 49](#_Toc211512912)

[4.5.3. Quy trình và các đoạn code minh họa 49](#_Toc211512913)

[4.5.4. Ứng dụng – “Why Use Structured Arrays?” 51](#_Toc211512914)

[4.5.5. Kết luận 51](#_Toc211512915)

[4.6 More About Sorting 52](#_Toc211512916)

[4.6.1. Mục tiêu 52](#_Toc211512917)

[4.6.2. Cơ sở lý thuyết 52](#_Toc211512918)

[4.6.3. Quy trình thực hiện và ví dụ minh họa 53](#_Toc211512919)

[4.6.4. Kết quả và nhận xét 56](#_Toc211512920)

[4.6.6. Kết luận 57](#_Toc211512921)

[4.7 Writing Fast NumPy Functions with Numba 57](#_Toc211512922)

[4.7.1 Tính giá trị trung bình của hiệu hai mảng số thực có kích thước rất lớn 57](#_Toc211512923)

[4.7.2 Creating Custom numpy.ufunc Objects with Numba 61](#_Toc211512924)

[4.8 Advanced Array Input and Output 62](#_Toc211512925)

[4.8.1 Memory-Mapped Files 62](#_Toc211512926)

[4.8.2 HDF5 and Other Array Storage Options 63](#_Toc211512927)

[4.9 Performance Tips 64](#_Toc211512928)

[V. Tổng kết và bài học 67](#_Toc211512929)

[VI. Tài liệu tham khảo 68](#_Toc211512930)

# 

# LỜI MỞ ĐẦU

Bài báo cáo này được thực hiện nhằm tổng hợp và vận dụng các kiến thức đã học trong học phần Phân tích dữ liệu (Data Analytics) vào thực tế thông qua nhiều bộ dữ liệu khác nhau.

Thông qua bài Lab 02 – Data Analytics, sinh viên được rèn luyện kỹ năng xử lý, làm sạch, trực quan hóa và phân tích dữ liệu bằng ngôn ngữ Python và các thư viện phổ biến như pandas, numpy, matplotlib và seaborn.

Báo cáo gồm nhiều phần, mỗi phần tập trung vào một bộ dữ liệu riêng biệt, từ đó sinh viên có thể: hiểu và vận dụng quy trình phân tích dữ liệu thực tế, phát hiện và diễn giải các xu hướng, mối quan hệ trong dữ liệu và củng cố kỹ năng lập trình và tư duy phân tích logic.

Bài báo cáo không chỉ nhằm kiểm tra kiến thức lý thuyết, mà còn giúp sinh viên hình thành tư duy dữ liệu (data-driven thinking) – một kỹ năng quan trọng trong học tập và công việc sau này.

# LỜI CẢM ƠN

Em xin gửi lời cảm ơn chân thành đến thầy/cô giảng viên hướng dẫn đã tận tình giảng dạy, hướng dẫn và hỗ trợ em trong suốt quá trình học tập và thực hiện bài Lab 02 – Phân tích dữ liệu (Data Analytics). Những kiến thức về Python, pandas, matplotlib, seaborn cùng với phương pháp tư duy phân tích dữ liệu mà thầy/cô đã truyền đạt là nền tảng quý giá giúp em hoàn thành tốt bài thực hành này. Em cũng xin cảm ơn các bạn cùng lớp đã chia sẻ tài liệu, kinh nghiệm và hỗ trợ lẫn nhau trong quá trình học tập, giúp việc thực hiện bài Lab trở nên dễ dàng và hiệu quả hơn.

Cuối cùng, em xin cảm ơn nhà trường đã tạo điều kiện thuận lợi về cơ sở vật chất, môi trường học tập và công cụ hỗ trợ để em có thể hoàn thành bài báo cáo này một cách tốt nhất.

Xin chân thành cảm ơn!

# BẢNG PHÂN CÔNG

|  |  |  |
| --- | --- | --- |
| **STT** | **Họ và tên** | **Nội dung công việc** |
| 1 | Mai Lê Mỹ Linh | Tìm hiểu phần 13.3, A.3, A.4, viết báo cáo và bài thuyết trình |
| 2 | Trần Lê Đăng Khoa | Tìm hiểu phần 13.1, 13.2 , A.3, A.4, viết báo cáo và bài thuyết trình |
| 3 | Lê Đức Nguyên Phú | Tìm hiểu phần 13.4, A.5, A.6, viết báo cáo và bài thuyết trình |
| 4 | Nguyễn Thị Xuân Mai | Tìm hiểu phần 13.5, 13.6 , A.7, A.8, A.9 viết báo cáo và bài thuyết trình |

# I. Giới thiệu chung

Trong thời đại dữ liệu bùng nổ hiện nay, việc hiểu và khai thác giá trị từ dữ liệu trở thành kỹ năng thiết yếu trong mọi lĩnh vực. Bài Lab 02 – Phân tích dữ liệu (Data Analytics) được thực hiện nhằm củng cố và mở rộng các kiến thức đã học về xử lý, trực quan hóa và phân tích dữ liệu bằng ngôn ngữ Python và các thư viện phổ biến như NumPy, pandas, matplotlib và seaborn.

Tài liệu Lab02\_Data\_Analytics.pdf tổng hợp nhiều ví dụ và bài tập thực hành dựa trên các bộ dữ liệu thực tế như: dữ liệu rút gọn liên kết từ Bitly (1.USA.gov), dữ liệu tên trẻ em ở Mỹ, cơ sở dữ liệu dinh dưỡng của USDA, dữ liệu tài chính bầu cử Hoa Kỳ, dữ liệu đánh giá phim (MovieLens), cùng các nguồn khác. Thông qua việc phân tích những tập dữ liệu đa dạng này, người học được rèn luyện khả năng:

* Hiểu cấu trúc và đặc điểm của từng loại dữ liệu;
* Tiền xử lý, làm sạch và chuẩn hóa dữ liệu để đảm bảo độ chính xác;
* Sử dụng các công cụ phân tích thống kê và trực quan hóa để tìm ra xu hướng, mẫu hình và mối quan hệ trong dữ liệu.

Ngoài ra, phần Advanced NumPy trong phụ lục của tài liệu còn giúp người học hiểu sâu hơn về cơ chế bên trong của thư viện NumPy — nền tảng cốt lõi của phân tích dữ liệu trong Python. Qua đó, sinh viên có thể tối ưu hiệu năng, kiểm soát bộ nhớ và xây dựng các giải pháp tính toán hiệu quả hơn cho các bài toán dữ liệu phức tạp.

Thông qua bài báo cáo này, mục tiêu không chỉ là vận dụng các công cụ đã học mà còn là phát triển tư duy phân tích dữ liệu (data-driven thinking) – tư duy dựa trên bằng chứng và con số, nhằm đưa ra kết luận chính xác và có cơ sở. Đây chính là kỹ năng trọng yếu đối với người học và người làm việc trong lĩnh vực khoa học dữ liệu, trí tuệ nhân tạo và phân tích kinh doanh hiện nay.

# II. Công cụ và phương pháp

## 1. Ngôn ngữ và thư viện sử dụng

Trong quá trình thực hiện bài *Lab 02 – Phân tích dữ liệu*, ngôn ngữ lập trình được sử dụng chính là Python – ngôn ngữ mạnh mẽ, phổ biến trong lĩnh vực phân tích dữ liệu và khoa học dữ liệu (Data Science).

Các thư viện hỗ trợ bao gồm:

* *pandas:* Dùng để xử lý, biến đổi và tổng hợp dữ liệu dưới dạng bảng (DataFrame).
* *NumPy***:** Cung cấp công cụ tính toán khoa học, hỗ trợ các thao tác mảng và ma trận tốc độ cao.
* *matplotlib***:** Dùng để trực quan hóa dữ liệu dưới dạng biểu đồ 2D.
* *seaborn***:** Xây dựng trên matplotlib, hỗ trợ tạo biểu đồ thống kê trực quan, sinh động và dễ hiểu.

Môi trường thực hành là Google Colab, cho phép chạy mã Python trực tiếp trên nền tảng đám mây, giúp thuận tiện trong việc nhập, xuất và xử lý dữ liệu mà không cần cài đặt phần mềm cục bộ.

## 2. Quy trình thực hiện

Quy trình phân tích dữ liệu trong bài được thực hiện theo các bước tuần tự sau:

**Đọc dữ liệu (Data Loading):**

* Sử dụng các hàm như pd.read\_csv(), pd.read\_json(), hoặc open() để đọc dữ liệu từ các nguồn như CSV, JSON, hoặc file text.
* Kiểm tra kích thước, kiểu dữ liệu và nội dung tổng quan bằng df.info() và df.head().

**Tiền xử lý dữ liệu (Data Preprocessing):**

* Làm sạch dữ liệu: xử lý giá trị thiếu (NaN), loại bỏ trùng lặp, chuyển đổi kiểu dữ liệu phù hợp.
* Chuẩn hóa dữ liệu: đổi tên cột, định dạng lại dữ liệu thời gian, mã hóa giá trị phân loại nếu cần.
* Lọc dữ liệu theo điều kiện để phục vụ mục tiêu phân tích.

**Phân tích dữ liệu (Data Analysis):**

* Sử dụng các hàm thống kê (mean(), count(), groupby(), value\_counts(), v.v.) để khám phá đặc điểm của dữ liệu.
* Kết hợp pandas và NumPy để tính toán, tổng hợp và tìm mối quan hệ giữa các biến.

**Trực quan hóa dữ liệu (Data Visualization):**

* Dùng **matplotlib** và **seaborn** để biểu diễn dữ liệu dưới dạng biểu đồ: bar chart, line chart, histogram, boxplot, heatmap,…
* Giúp phát hiện xu hướng, phân bố và mối tương quan giữa các yếu tố.

**Nhận xét và kết luận (Interpretation):**

* Từ kết quả trực quan, rút ra nhận xét, xu hướng, và kết luận về dữ liệu.
* Liên hệ các phát hiện với ngữ cảnh thực tế của từng bộ dữ liệu (ví dụ: hành vi người dùng, xu hướng đặt tên, mô hình đóng góp tài chính, v.v.).

## 3. Mục tiêu phương pháp

Phương pháp này giúp đảm bảo quy trình làm việc có hệ thống, từ khâu thu thập dữ liệu đến phân tích và rút ra insight. Bằng cách sử dụng Colab và các thư viện Python, người học có thể tái hiện toàn bộ quy trình phân tích dữ liệu chuyên nghiệp chỉ bằng công cụ mã nguồn mở.

# III. Phân tích dữ liệu theo từng mục

## 3.1. Dataset 1 – Bitly Data from 1.USA.gov

### 3.1.1 Mục tiêu:

Phân tích dữ liệu từ dịch vụ rút gọn link 1.USA.gov (Bitly) để hiểu hành vi người dùng, tập trung vào:

* Phân bố theo múi giờ (tz)
* Quốc gia truy cập
* Đặc điểm trình duyệt/thiết bị (từ user\_agent)
* Nguồn giới thiệu (referrer) tiêu biểu

### 3.1.2 Phân tích:

Đọc dữ liệu JSON.

* Parse từng dòng JSON; bỏ qua dòng “heartbeat”/hỏng.
* Tổng số bản ghi hợp lệ: 3,440.

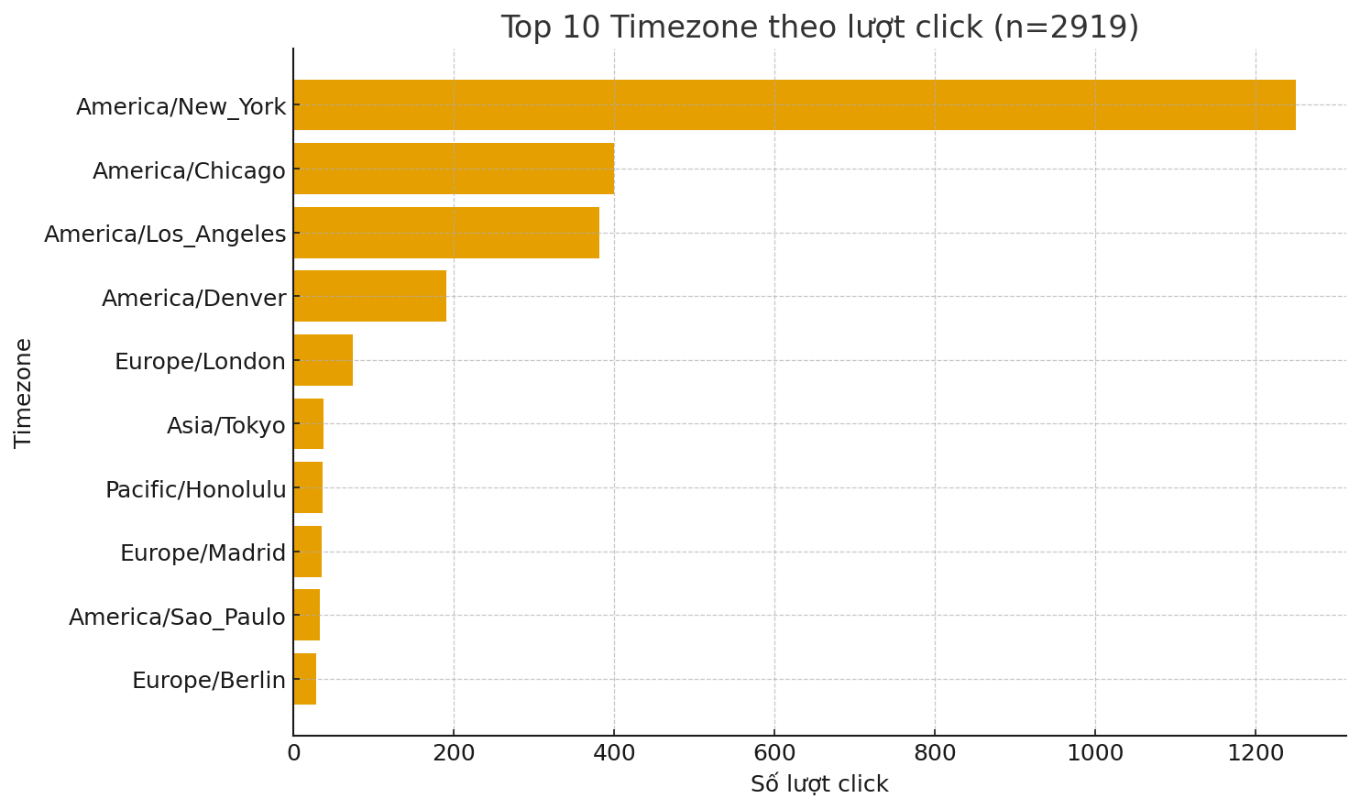
Tách thông tin như múi giờ (tz), quốc gia, trình duyệt.

* Trích: tz, c (country), a (user\_agent), t (timestamp), u (URL), r (referrer)…
* Chuẩn hóa chuỗi, xử lý rỗng/thiếu.
* Bao phủ dữ liệu:
  + Có tz: 2,919 bản ghi; trống tz: 521
  + Có country: 2,919; trống country: 521

Thống kê tần suất & tỷ lệ:

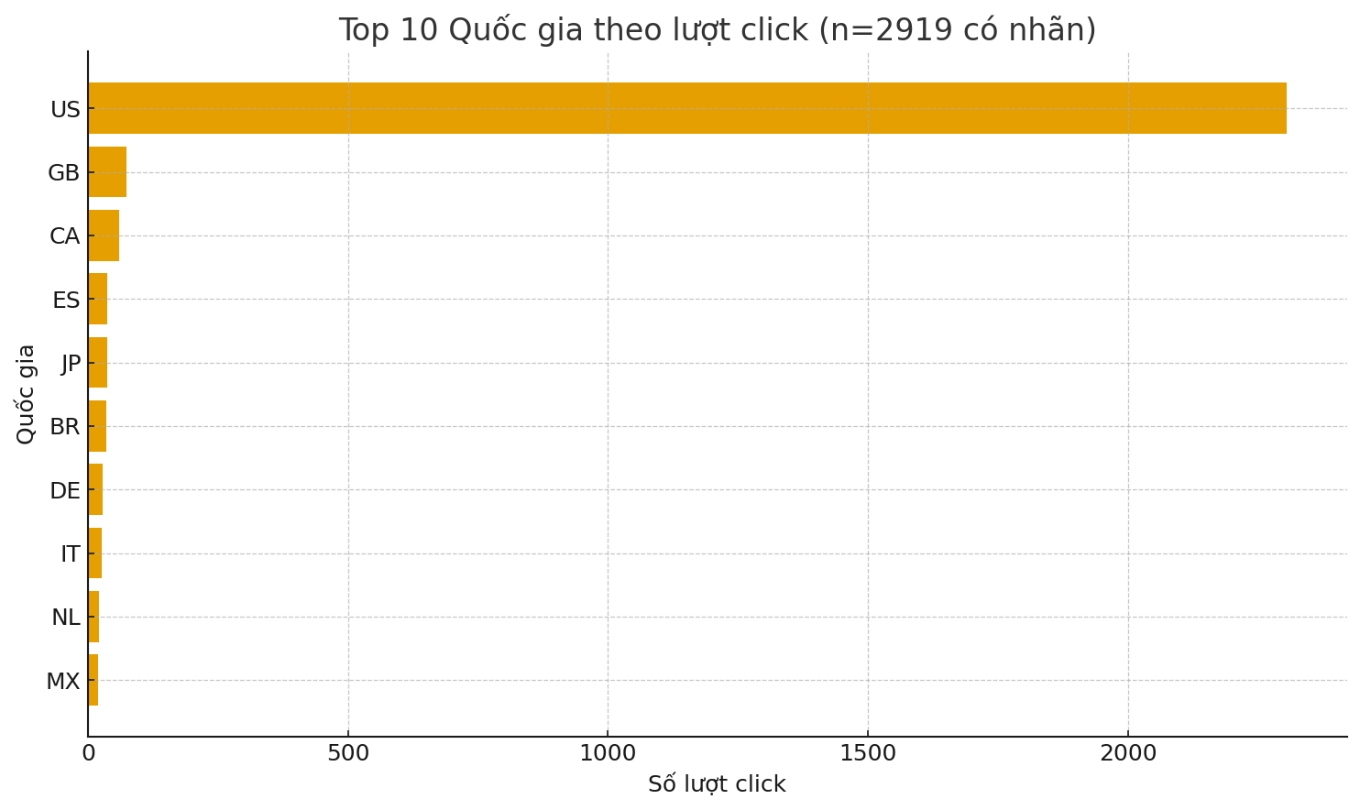
* Đếm tần suất theo tz, country, nhóm browser family & device type từ user\_agent.
* Tính % trên toàn bộ 3,440 và (với country) thêm % trên tập có nhãn.

**Trực quan hóa bằng biểu đồ. Top Timezone theo lượt click:**



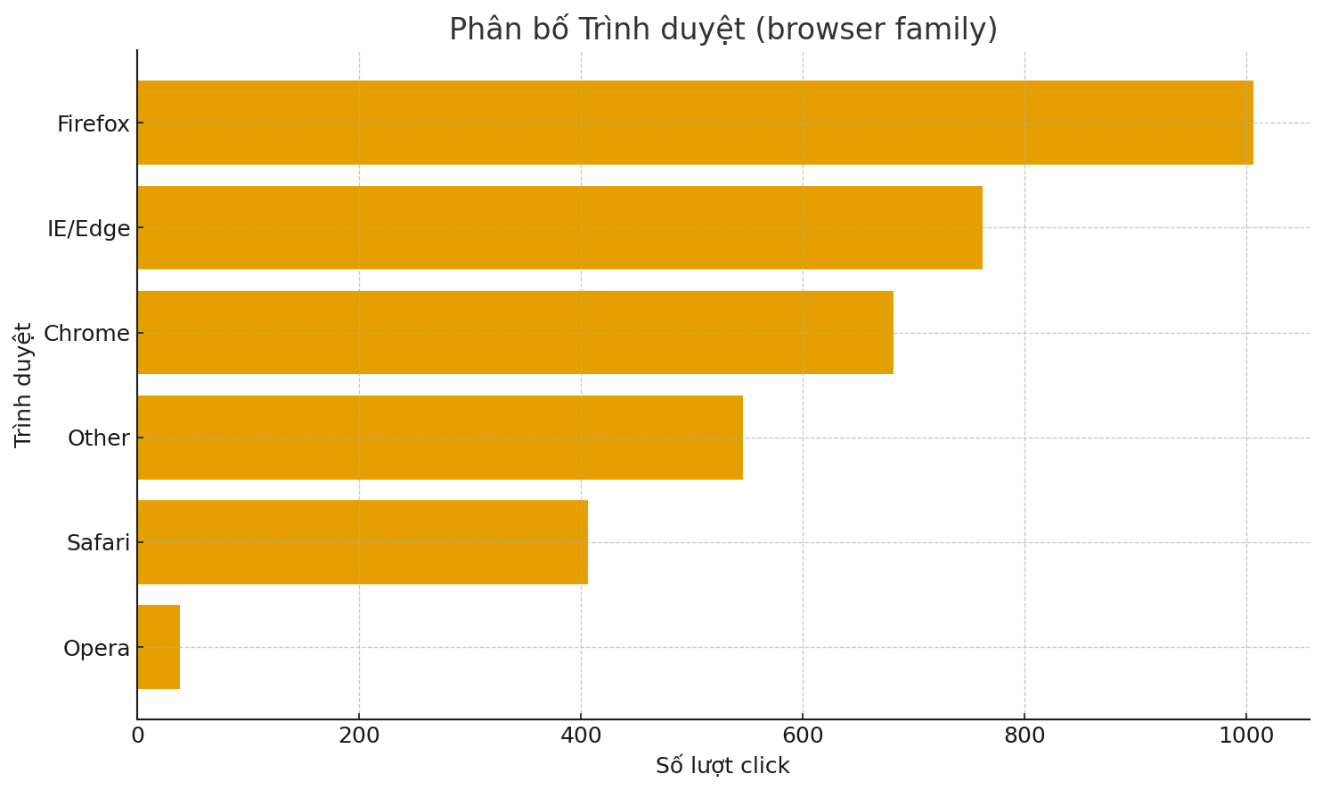
Hình 1.Hiển thị 10 múi giờ có lượng truy cập cao nhất (Cho thấy người dùng tập trung chủ yếu ở múi giờ America/New\_York, America/Chicago và America/Los\_Angeles.).

**Top Quốc gia theo lượt click:**



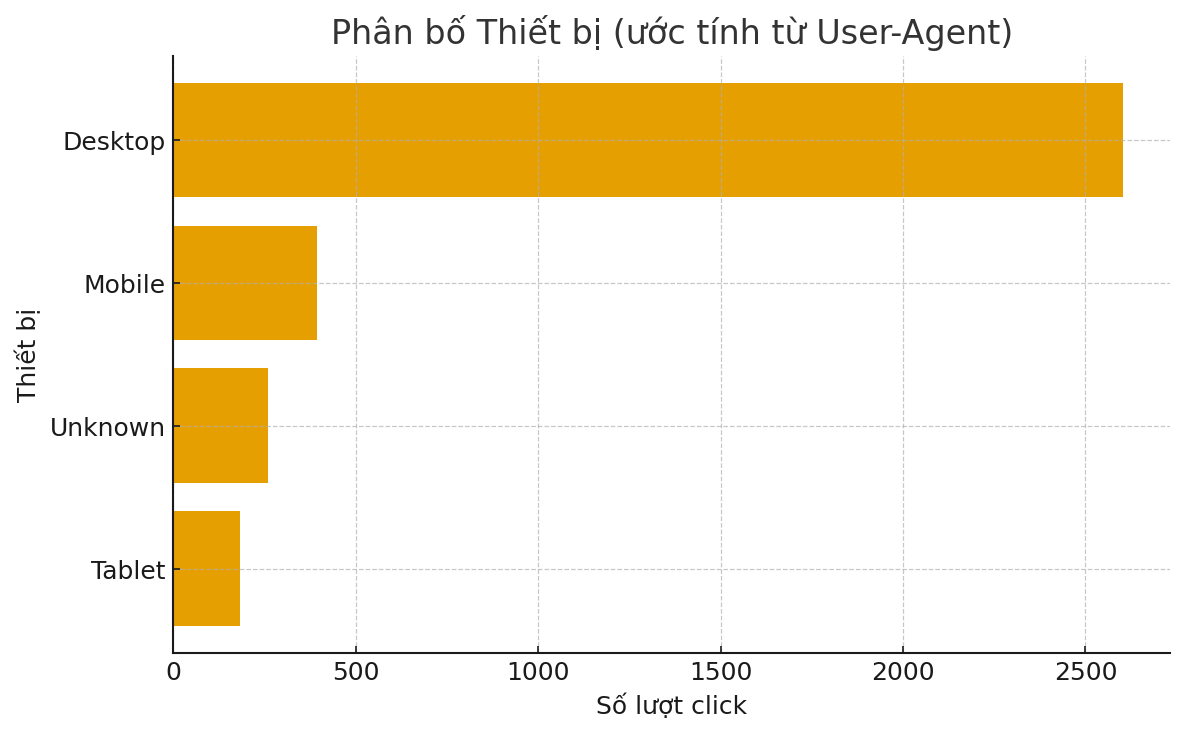
Hình 2. Phân bố lượt click theo quốc gia (Hoa Kỳ chiếm khoảng 67% tổng truy cập, đứng đầu tuyệt đối.)

**Phân bố Trình duyệt (Browser):**



Hình 3. Thể hiện tỷ lệ truy cập theo loại trình duyệt (Firefox (29%), IE/Edge (22%) và Chrome (20%) là ba nhóm chính).

**Phân bố Thiết bị (Device):**



Hình 4. Phân tích loại thiết bị truy cập (Desktop chiếm ~76%, cho thấy người dùng chủ yếu truy cập từ máy tính bàn hoặc laptop).

### 3.1.3 Kết quả & nhận xét:

*Bao phủ dữ liệu:*

* Tổng bản ghi hợp lệ: 3,440
* tz: có nhãn 2,919 (≈ 84.86%), trống 521
* country: có nhãn 2,919 (≈ 84.86%), trống 521

*Timezone (Top 10) — đếm & % trên toàn bộ 3,440:*

* America/New\_York: 1,251 (≈ 36.37%)
* America/Chicago: 400 (≈ 11.63%)
* America/Los\_Angeles: 382 (≈ 11.10%)
* America/Denver: 191 (≈ 5.55%)
* Europe/London: 74 (≈ 2.15%)
* Asia/Tokyo: 37 (≈ 1.08%), Pacific/Honolulu: 36 (≈ 1.05%), Europe/Madrid: 35 (≈ 1.02%), America/Sao\_Paulo: 33 (≈ 0.96%), Europe/Berlin: 28 (≈ 0.81%)

*Country:*

* US: 2,305 → 67.01% (toàn bộ 3,440) | 78.97% (trên tập có nhãn = 2,919)
* GB: 74 → 2.15% | 2.54%
* CA: 60 → 1.74% | 2.06%
* ES: 37 → 1.08% | 1.27%
* JP: 37 → 1.08% | 1.27%
* BR: 35 → 1.02% | 1.20%
* DE: 28 → 0.81% | 0.96%
* IT: 27 → 0.78% | 0.92%
* NL: 22 → 0.64% | 0.75%
* Thiếu nhãn country: 521 (≈ 15.15%)

Browser family

* Firefox: 1,006 (≈ 29.24%)
* IE/Edge: 762 (≈ 22.15%)
* Chrome: 682 (≈ 19.83%)
* Safari: 406 (≈ 11.80%)
* Opera: 38 (≈ 1.10%)
* Other: 546 (≈ 15.87%

*Device type :*

* Desktop: 2,602 (≈ 75.64%)
* Mobile: 393 (≈ 11.42%)
* Tablet: 184 (≈ 5.35%)
* Unknown: 261 (≈ 7.59%)

**Nhận xét:**

Trọng tâm theo địa lý & thời gian:

* Lưu lượng tập trung ở Hoa Kỳ; các múi giờ America/New\_York, America/Chicago, America/Los\_Angeles chiếm ưu thế.
* Gợi ý vận hành: đăng/đẩy nội dung theo giờ hành chính EST/EDT, sau đó đến CST/PST để tối đa hiệu quả.

Tỷ trọng quốc gia:

* US chiếm ~67% (toàn bộ) và ~79% (trên tập có nhãn) → phản ánh đúng bối cảnh liên kết .gov/.mil.
* Có sự quan tâm quốc tế từ GB, CA, ES, JP,… nhưng ở mức độ khiêm tốn.

Hành vi thiết bị :

* Desktop ~76% → người dùng chủ yếu truy cập từ máy tính (phù hợp hoạt động giờ làm việc/tra cứu dịch vụ công).
* Dù Mobile/Tablet ~17%, vẫn nên đảm bảo tối ưu giao diện di động cơ bản.

Tương thích trình duyệt :

* Firefox & IE/Edge chiếm tỷ trọng đáng kể (đặc trưng lịch sử của bộ dữ liệu này), tiếp đến Chrome và Safari.
* Khuyến nghị: kiểm thử đa trình duyệt, đặc biệt Firefox và IE/Edge/Edge cho các trang/cổng thông tin công.

Chất lượng dữ liệu & hạn chế:

* Tỷ lệ thiếu country ~15% và một phần thiếu tz có thể làm sai lệch tỉ lệ theo khu vực.
* Khi cần báo cáo chính xác hơn theo vùng/khu vực, nên ghi chú hạn chế này hoặc bổ sung nguồn đối chiếu.

## 3.2. Dataset 2 - MovieLens 1M Dataset

### 3.2.1 Mục tiêu:

Phân tích bộ dữ liệu MovieLens 1M nhằm:

* Nắm tổng quan: quy mô người dùng, phim, lượt đánh giá; độ thưa (sparsity) của ma trận user–movie.
* Khám phá phân phối điểm đánh giá, thể loại (genre), phim được đánh giá nhiều nhất, và xu hướng theo năm phát hành.
* Rút ra nhận xét phục vụ các tác vụ gợi ý/phân tích hành vi.

### 3.2.2 Phân tích

* Đọc dữ liệu: Định dạng :: trong .dat; xử lý mã hóa latin-1 để tránh lỗi ký tự
* Tiền xử lý:
  + Trích năm phát hành từ tiêu đề phim (ví dụ: (1995) trong Title).
  + Tách cột Genres thành danh sách & “nổ” (explode) để tính theo từng thể loại.
  + Gộp bảng ratings–movies–users để có khung dữ liệu đầy đủ.
* Thống kê chính:
  + Phân phối điểm đánh giá (1–5); tính điểm trung bình toàn bộ.
  + Top thể loại theo số lượt đánh giá và điểm trung bình.
  + Top phim theo số lượt đánh giá (kèm điểm trung bình).
  + Số lượt đánh giá theo năm phát hành để thấy xu hướng theo thời gian.
  + Tính sparsity = 1 – (#ratings / (#users × #movies)).

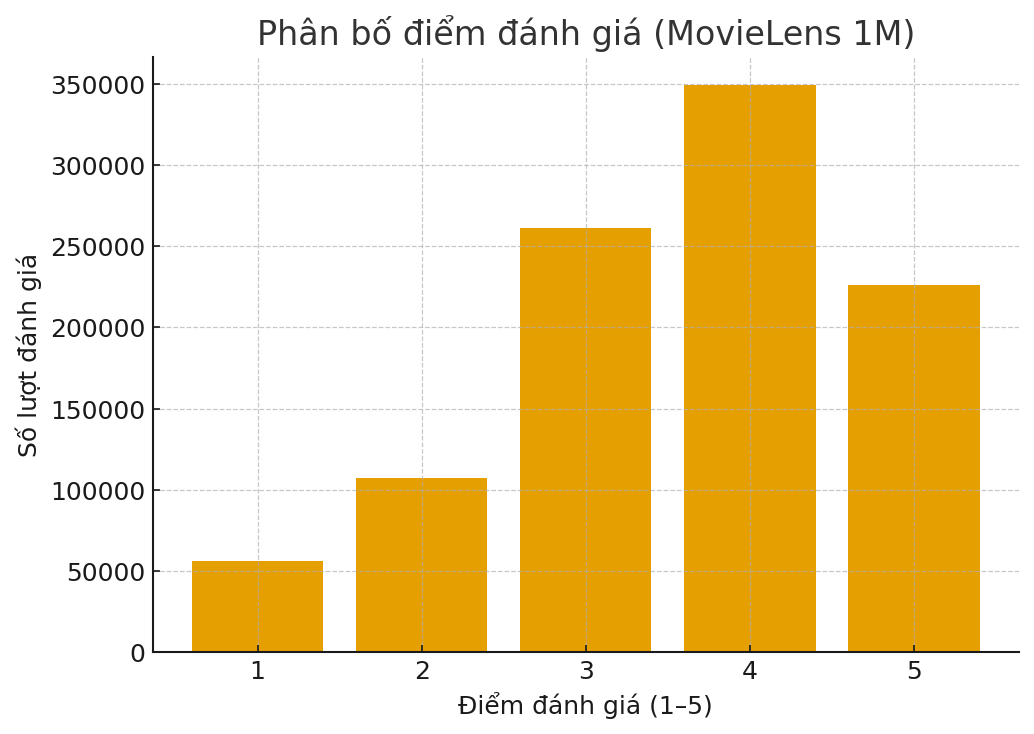
### Kết quả & nhận xét

**a. Tổng quan dữ liệu:**

* Người dùng: 6,040
* Phim: 3,883
* Lượt đánh giá: 1,000,209
* Sparsity (độ thưa ma trận user–movie): 95.7353% → ma trận rất thưa (phù hợp bối cảnh recommenders thực tế).
* Điểm trung bình toàn bộ: ~3.58/5.

**b. Phân phối điểm đánh giá (1–5)**

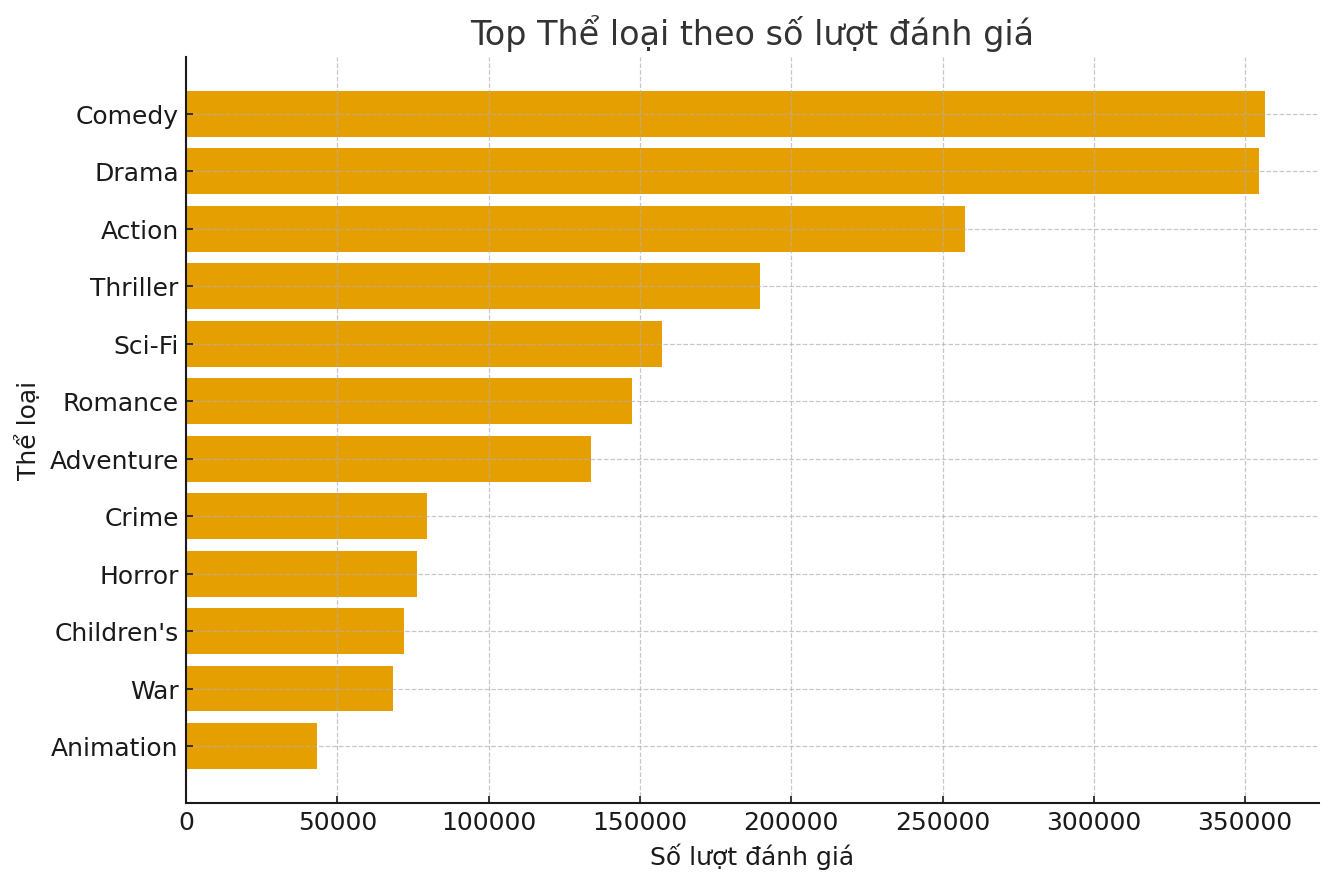
* Mức 3–4–5 chiếm đa số; trung bình ~3.58, phản ánh thiên hướng đánh giá tích cực.



Hình 5. Biểu đồ thể hiện tần suất các mức điểm từ 1 đến 5. Phần lớn người dùng đánh giá ở mức 3–4–5, thể hiện xu hướng chấm điểm tích cực.

**c. Thể loại (genre): Top thể loại theo số lượt đánh giá (kèm điểm TB):**

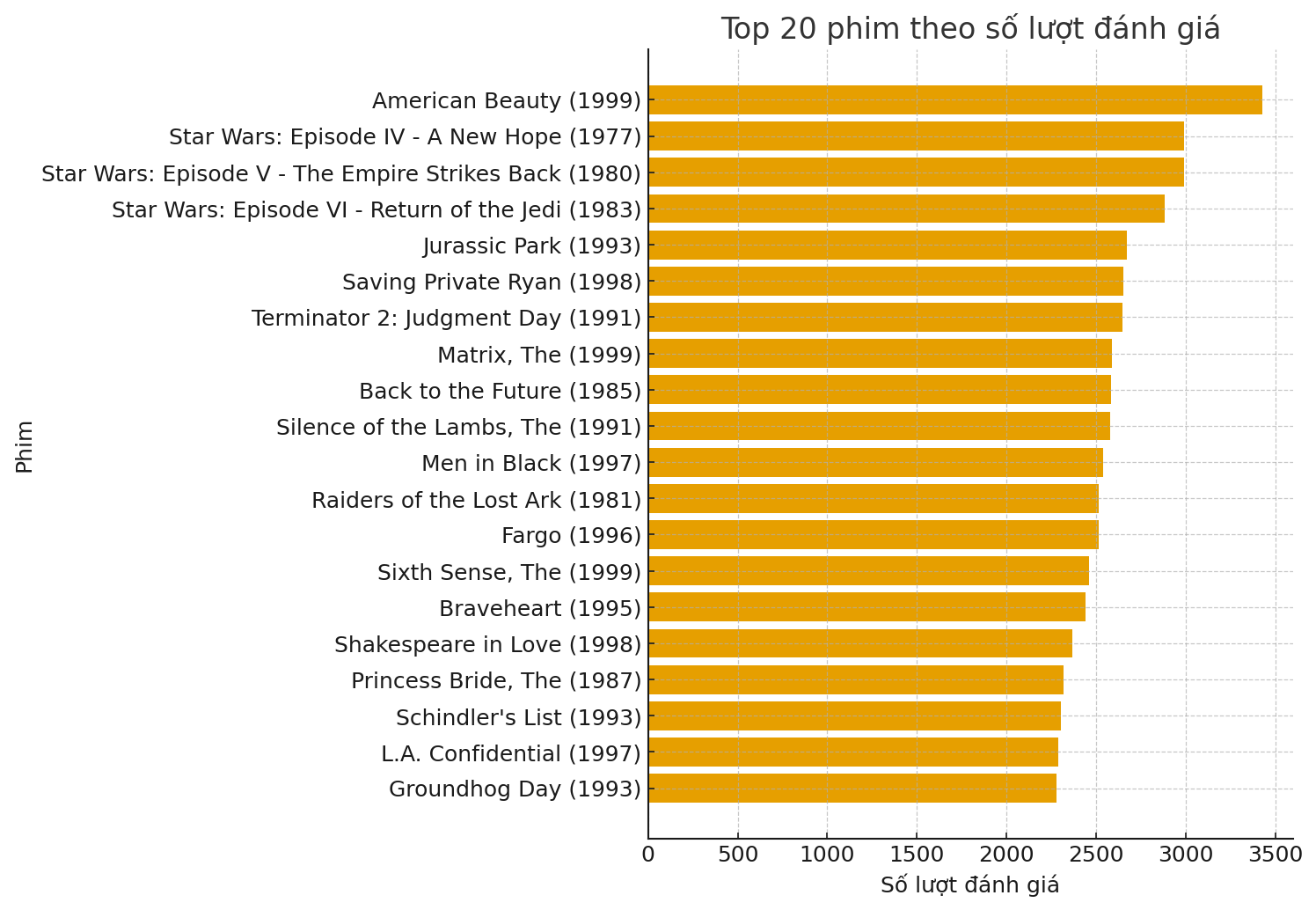
* Comedy: 356,580 (TB 3.52)
* Drama: 354,529 (TB 3.77)
* Action: 257,457 (TB 3.49)
* Thriller: 189,680 (TB 3.57)
* Sci-Fi: 157,294 (TB 3.47)
* Romance: 147,523 (TB 3.61)



Hình 6. Biểu đồ cột ngang thể hiện các thể loại có nhiều lượt đánh giá nhất. “Comedy”, “Drama” và “Action” chiếm tỷ lệ cao nhất, cho thấy độ phổ biến của các thể loại này.

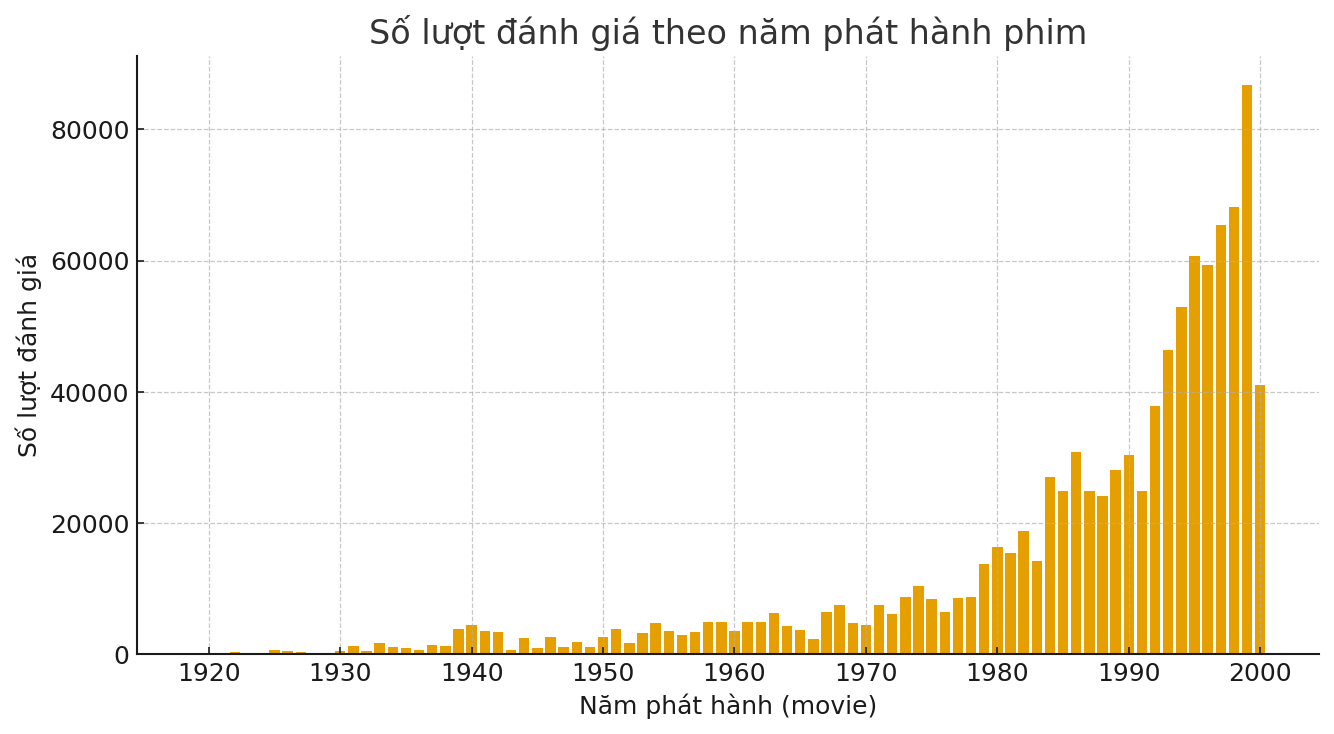
**d. Top 20 phim theo số lượt đánh giá**

* American Beauty (1999) — 3,428 lượt, TB 4.32
* Star Wars: Episode IV - A New Hope (1977) — 2,991 lượt, TB 4.45
* Star Wars: Episode V - The Empire Strikes Back (1980) — 2,990 lượt, TB 4.29
* Star Wars: Episode VI - Return of the Jedi (1983) — 2,883 lượt, TB 4.02
* Jurassic Park (1993) — 2,672 lượt, TB 3.76
* Saving Private Ryan (1998) — 2,653 lượt, TB 4.34
* Terminator 2: Judgment Day (1991) — 2,649 lượt, TB 4.06
* The Matrix (1999) — 2,590 lượt, TB 4.32
* Back to the Future (1985) — 2,583 lượt, TB 3.99
* The Silence of the Lambs (1991) — 2,578 lượt, TB 4.35
* Men in Black (1997) — 2,538 lượt, TB 3.74
* Raiders of the Lost Ark (1981) — 2,514 lượt, TB 4.48
* Fargo (1996) — 2,513 lượt, TB 4.25
* The Sixth Sense (1999) — 2,459 lượt, TB 4.41
* Braveheart (1995) — 2,443 lượt, TB 4.23
* Shakespeare in Love (1998) — 2,369 lượt, TB 4.13
* The Princess Bride (1987) — 2,318 lượt, TB 4.30
* Schindler’s List (1993) — 2,304 lượt, TB 4.51
* L.A. Confidential (1997) — 2,288 lượt, TB 4.22
* Groundhog Day (1993) — 2,278 lượt, TB 3.95



Hình 7. Biểu đồ thể hiện 20 phim được đánh giá nhiều nhất. Các phim như American Beauty (1999) và Star Wars (1977) đứng đầu danh sách, vừa phổ biến vừa có điểm trung bình cao.

**e. Lượt đánh giá theo năm phát hành phim**



Hình 8. Biểu đồ thể hiện tổng số lượt đánh giá của các phim phát hành theo từng năm. Giai đoạn 1995–1999 có lượng đánh giá cao nhất.

**Nhận xét:**

*Phân bố điểm đánh giá:*

* Phần lớn người dùng chấm 4 điểm (≈35%) và 3 điểm (≈26%), cho thấy xu hướng đánh giá tích cực, ít chấm điểm thấp.
* Các điểm 1–2 chiếm tỷ lệ nhỏ → dữ liệu có thiên hướng lạc quan, phù hợp với mô hình gợi ý phim.

*Số lượt đánh giá theo năm phát hành:*

* Phim phát hành từ thập niên 1980–2000 có lượng đánh giá tăng mạnh, đỉnh điểm là các năm 1995–2000.
* Các phim cũ (trước 1960) gần như không được đánh giá nhiều → người dùng tập trung vào phim hiện đại và phổ biến.

*Thể loại phim:*

* Comedy và Drama là hai thể loại được đánh giá nhiều nhất (>350.000 lượt), kế đến là Action, Thriller, Sci-Fi.
* Điều này phản ánh thị hiếu đa dạng nhưng nghiêng về giải trí và kịch tính.

*Top 20 phim được đánh giá nhiều nhất:*

* Dẫn đầu là American Beauty (1999), Star Wars (1977–1983), Saving Private Ryan (1998), The Matrix (1999),…
* Hầu hết đều là phim kinh điển cuối thế kỷ 20, vừa được đánh giá cao vừa phổ biến rộng → minh chứng tập người dùng có xu hướng yêu thích phim nổi tiếng, chất lượng cao.

## 3.3. Dataset 3 – US Baby Names 1880–2010

### 3.3.1. Mục tiêu

Phân tích bộ dữ liệu US Baby Names (1880–2010) nhằm:

* Nắm tổng quan: quy mô dữ liệu, số năm, tổng lượt sinh, phân bố giới tính
* Phân tích xu hướng đặt tên theo thời gian: tên phổ biến, tỷ lệ giới tính, độ đa dạng.
* Khám phá các đặc điểm ngôn ngữ như chữ cái cuối của tên (last letter), sự thay đổi theo thập kỷ.
* Phát hiện xu hướng văn hóa – xã hội trong cách đặt tên qua hơn 130 năm, phục vụ các bài toán phân tích hành vi và dự đoán tên.

### 3.3.2. Quy trình thực hiện

***Bước 1 – Nạp dữ liệu***: Đọc dữ liệu từ file CSV babynames.csv (được tổng hợp từ các file gốc yob1880.txt–yob2010.txt của SSA).

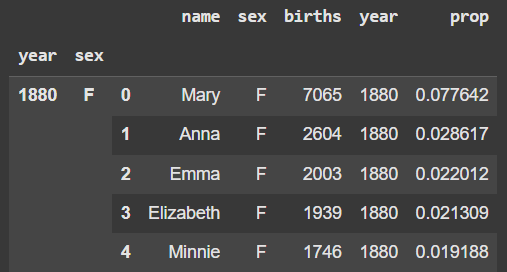
Dữ liệu gồm các cột:

* name: tên trẻ
* sex: giới tính (F hoặc M)
* births: số lượng trẻ được đặt tên đó
* year: năm sinh

Kích thước dữ liệu: khoảng 1.69 triệu dòng, bao phủ giai đoạn 1880–2010.

***Bước 2 – Tiền xử lý:***

* Chuẩn hóa dữ liệu: loại bỏ dòng trống, viết hoa chữ cái đầu của tên (name).
* Thêm cột prop = births / tổng số sinh cùng năm & giới → biểu thị tỷ lệ trẻ mang tên đó.
* Kiểm tra tổng tỷ lệ theo từng nhóm (year, sex) để đảm bảo dữ liệu hợp lệ (tổng = 1).



***Bước 3 – Lọc Top 1000 tên phổ biến nhất mỗi năm***

* Nhóm dữ liệu theo (year, sex) và chọn 1000 tên có nhiều lượt sinh nhất trong từng nhóm.
* Bảng kết quả top1000 gồm ~130 năm × 2 giới × 1000 tên → khoảng 262,000 dòng.

***Bước 4 – Trực quan hóa xu hướng tổng quan***

* Tổng số trẻ sinh theo năm & giới tính (pivot table và biểu đồ line).
* Xu hướng đặt tên theo từng cá nhân (John, Mary, Harry, Marilyn).
* Tỷ lệ tổng của Top 1000 tên theo thời gian (độ đa dạng).
* Phân tích chữ cái cuối cùng của tên (last letter revolution).
* So sánh nhóm tên unisex (ví dụ “Leslie”, “Lesley”).

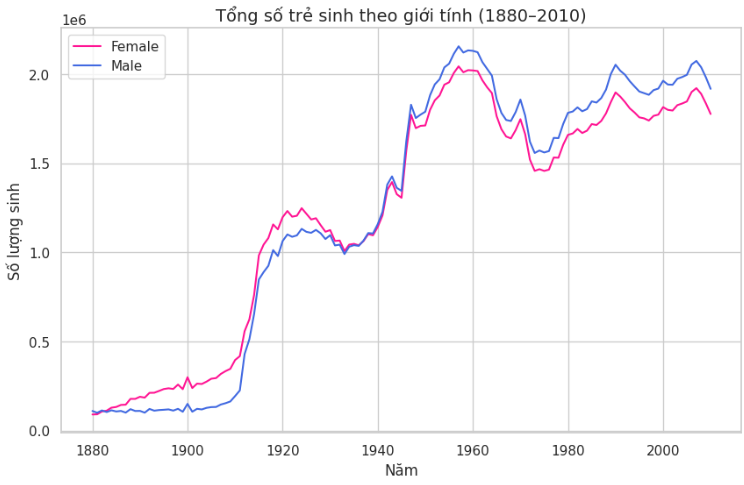
***Bước 5 – Phân tích nâng cao***

* Tính toán số lượng tên chiếm 50% tổng số sinh để đo lường mức độ tập trung.
* So sánh theo giới tính qua các năm → xác định giai đoạn đa dạng hóa mạnh nhất.
* Lọc và phân tích riêng các chữ cái cuối cùng phổ biến (d, n, y).

### 3.3.3. Phân tích và kết quả

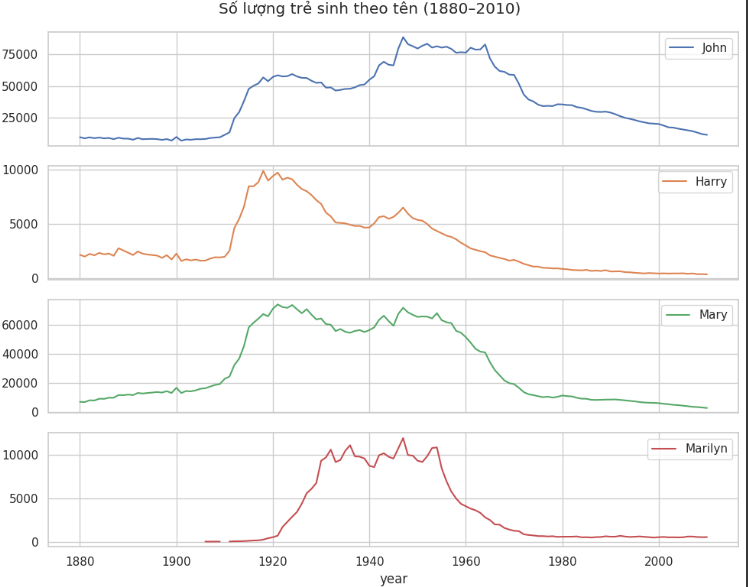
**a. Tổng quan dữ liệu**

* Số dòng dữ liệu: 1,690,784
* Số năm: 131 (1880–2010)
* Số tên khác nhau: ≈96,400
* Bé trai luôn có số lượng sinh cao hơn bé gái.
* Số lượng sinh đạt đỉnh vào khoảng 1955–1965, sau đó giảm dần.



**b. Xu hướng tên phổ biến**

* Các tên truyền thống như Mary, John, William phổ biến mạnh trong thế kỷ 19–20, nhưng giảm dần sau thập niên 1960.
* Ngược lại, các tên hiện đại và cá nhân hóa hơn (Ashley, Madison, Joshua, Jacob…) tăng nhanh sau năm 1980.



**Nhận xét:** Xu hướng đặt tên thay đổi theo thế hệ, phản ánh sự dịch chuyển văn hóa: từ truyền thống sang hiện đại, từ phổ biến sang độc đáo.

**c. Độ đa dạng trong đặt tên (Diversity Trend)**

Tính tổng tỷ lệ của Top 1000 tên mỗi năm:

* Giai đoạn đầu (1880–1920): Top 1000 chiếm >90% tổng số sinh → đặt tên ít đa dạng.
* Sau 1980: tỷ lệ giảm xuống <75% → độ đa dạng tăng mạnh.
* Bé gái luôn có độ đa dạng cao hơn bé trai.

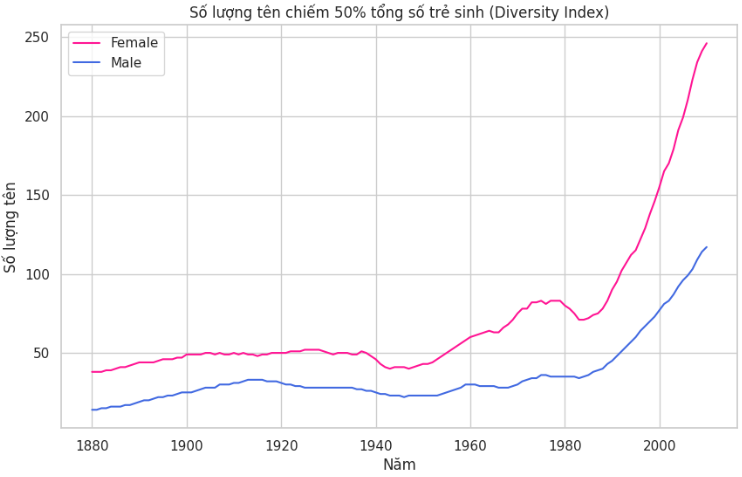


**Nhận xét**:  Độ đa dạng đặt tên tăng phản ánh xu hướng cá nhân hóa, đặc biệt sau năm 1980.

**d. Số lượng tên chiếm 50% tổng số sinh**

* Năm 1880: chỉ cần 38 tên nữ và 14 tên nam chiếm 50% tổng số sinh.
* Năm 2010: cần 250+ tên nữ và 130+ tên nam để đạt cùng tỷ lệ.

Xã hội ngày càng đa dạng trong đặt tên, đặc biệt ở nữ giới.

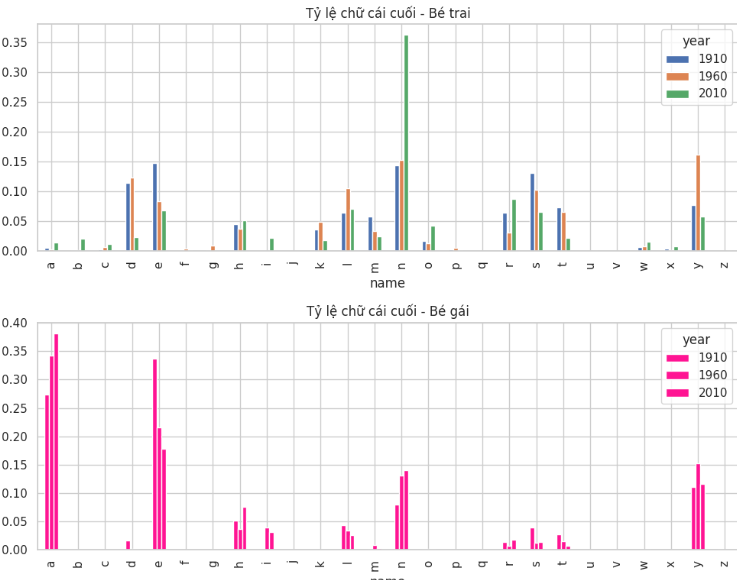


**e. Phân tích chữ cái cuối cùng (Last Letter Revolution)**

Tổng hợp tỷ lệ tên kết thúc bằng các chữ cái.

Kết quả cho thấy:

* Tên nữ kết thúc bằng **“a”** chiếm tỷ lệ cao nhất (≈38% năm 2010).
* Tên nam kết thúc bằng **“n”** tăng nhanh sau 1980, vượt “y” và “d”.



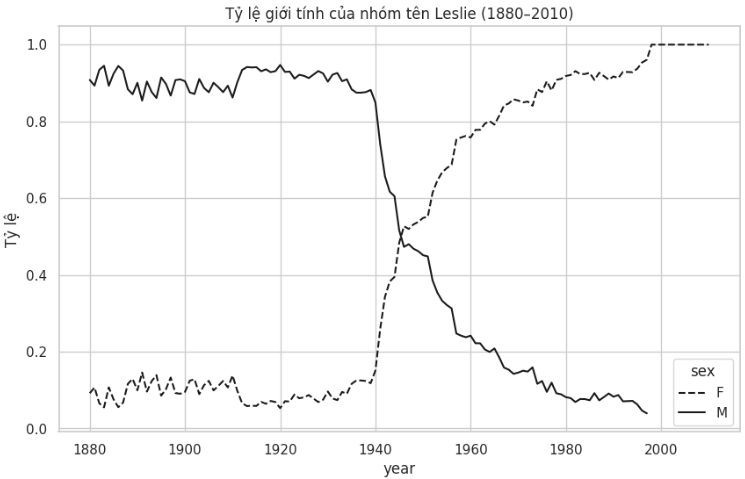
**Nhận xét:**

* Bé gái: “a” gợi cảm giác mềm mại, nữ tính → phổ biến.
* Bé trai: “n” (như Ethan, Aiden, Ryan) thể hiện âm điệu hiện đại, mạnh mẽ.

**f. Phân tích tên unisex (Leslie/Lesley)**

Các tên **Leslie**, **Lesley** ban đầu phổ biến cho bé trai (trước 1940), nhưng sau 1980 gần như chỉ còn được đặt cho bé gái.

→ Phản ánh thay đổi nhận thức giới tính và ảnh hưởng văn hóa xã hội.



### 3.3.4. Nhận xét tổng hợp

* Độ đa dạng đặt tên tăng rõ rệt qua từng thập niên.
* Bé gái có xu hướng tên phong phú và đổi mới nhanh hơn bé trai.
* Âm tiết cuối (“a” – nữ, “n” – nam) thể hiện xu hướng thẩm mỹ ngôn ngữ hiện đại.
* Một số tên chuyển đổi giới tính theo thời gian (Leslie, Taylor, Jordan…).
* Dữ liệu cho phép phân tích xã hội học và ngôn ngữ học qua hơn 130 năm.

### 3.3.5. Kết luận

Phân tích dữ liệu US Baby Names 1880–2010 minh họa rõ quy trình xử lý – làm sạch – phân tích – trực quan hóa dữ liệu lớn với Python/Pandas.

Kết quả thể hiện:

* Xu hướng đặt tên tại Hoa Kỳ thay đổi mạnh qua thời gian, phản ánh tiến trình cá nhân hóa và bình đẳng giới.
* Bé trai luôn chiếm tỷ lệ sinh cao hơn, nhưng bé gái có mức đa dạng tên lớn hơn.
* Tên truyền thống giảm, tên hiện đại tăng; đặc biệt, những tên kết thúc bằng “a” và “n” trở thành xu hướng phổ biến.
* Một số tên unisex chuyển đổi hoàn toàn về giới tính sử dụng, cho thấy ảnh hưởng văn hóa – xã hội rõ nét.

Bộ dữ liệu này là ví dụ điển hình cho Exploratory Data Analysis (EDA), cho phép rút ra insight văn hóa từ dữ liệu dân số thực tế, đồng thời ứng dụng tốt trong các mô hình thống kê và dự báo xu hướng đặt tên trong tương lai.

## 3.4. Dataset 4 – USDA Food Database

### 3.4.1. Mục tiêu

Phần thực hành này nhằm mục đích:

* Làm quen với cấu trúc dữ liệu JSON phức tạp (nested JSON).
* Thực hành xử lý, làm phẳng và trích xuất dữ liệu từ tệp JSON của USDA Food Database.
* Gộp dữ liệu thực phẩm với dữ liệu dinh dưỡng, phân tích thống kê, và tìm ra các thực phẩm giàu nhất ở từng nhóm chất.
* Rèn luyện kỹ năng sử dụng thư viện pandas trong Python để xử lý dữ liệu thực tế.

### 3.4.2. Cơ sở dữ liệu USDA FoodData Central

Nguồn dữ liệu**:** [Bộ Nông nghiệp Hoa Kỳ (USDA)](https://fdc.nal.usda.gov/download-datasets/) công bố công khai cơ sở dữ liệu FoodData Central, chứa thông tin chi tiết về thành phần dinh dưỡng của hàng nghìn loại thực phẩm.

Định dạng dữ liệu: JSON, trong đó mỗi thực phẩm là một bản ghi gồm:

* Thông tin định danh: mô tả, mã số (fdcId), nhóm thực phẩm, nhà sản xuất.
* Danh sách “foodNutrients”: chứa tên chất, đơn vị và giá trị.
* Danh sách “foodPortions”: mô tả khẩu phần tương ứng (grams, units…).

### 3.4.3. Quy trình thực hiện

***Bước 1: Nạp dữ liệu JSON***

Sử dụng thư viện json để đọc file vào Python.

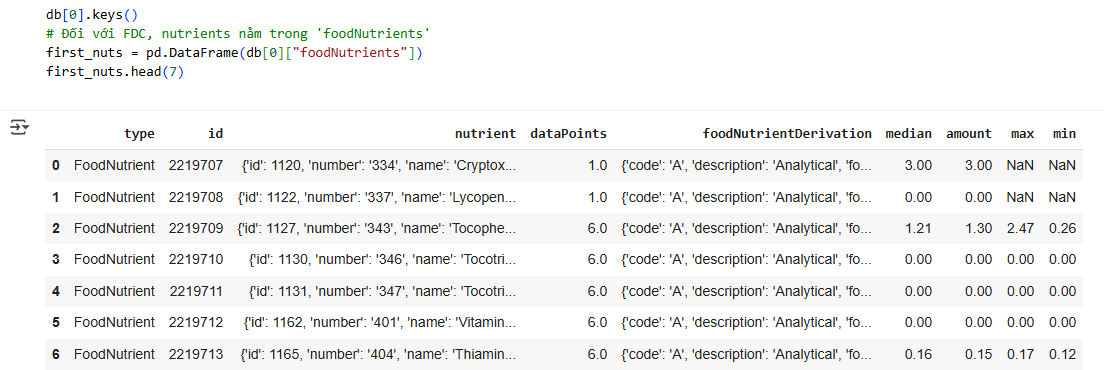
Kết quả là một danh sách gồm hàng nghìn bản ghi (mỗi bản ghi tương ứng một loại thực phẩm).  
Việc kiểm tra cấu trúc (.keys()) giúp hiểu được các trường cần thiết cho việc phân tích.

***Bước 2: Tạo bảng thông tin thực phẩm***

Từ mỗi bản ghi, trích xuất các trường:

* description: tên món ăn
* group: nhóm thực phẩm (nằm trong foodCategory.description)
* id: mã fdcId
* manufacturer: nhà sản xuất (nếu có)

Bảng info là bảng tóm tắt (metadata) về tất cả thực phẩm.



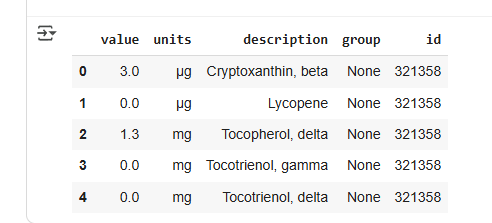
Hình 1. Kết quả đọc bảng foodNutrients bằng Pandas DataFrame

**Bước 3 – Tạo bảng dữ liệu dinh dưỡng (nutrients)**

Duyệt qua từng bản ghi, trích xuất danh sách foodNutrients.  
Mỗi chất dinh dưỡng được đưa vào DataFrame riêng, có các cột:

* value: giá trị dinh dưỡng
* units: đơn vị đo (g, mg, kcal, v.v.)
* nutrient: tên chất dinh dưỡng
* id: mã thực phẩm để liên kết với bảng info

Sau đó, nối tất cả các bảng nhỏ này lại thành một bảng dinh dưỡng tổng hợp.  
Loại bỏ các dòng trùng lặp để tránh dư thừa.



Hình 2. Bảng dữ liệu dinh dưỡng sau khi trích xuất từ các bản ghi foodNutrients

***Bước 4: Chuẩn hóa dữ liệu***

Đổi tên các cột để dễ hiểu và tránh trùng lặp:

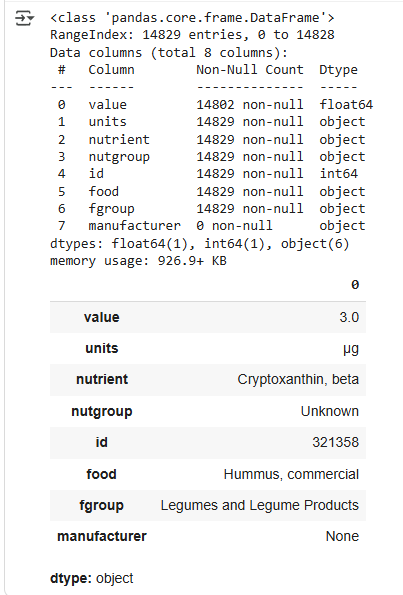
* Trong bảng info: description → food, group → fgroup
* Trong bảng nutrients: description → nutrient, group → nutgroup

Điều này giúp dễ dàng khi gộp hai bảng và thao tác sau này.

***Bước 5: Gộp dữ liệu***

Thực hiện phép merge theo cột id để gộp dữ liệu thực phẩm (info) với dữ liệu dinh dưỡng (nutrients).  
Bảng kết quả (ndata) chứa đầy đủ thông tin:

* Tên thực phẩm, nhóm thực phẩm
* Tên chất dinh dưỡng, đơn vị, giá trị
* Mã định danh và nhà sản xuất



Hình 3. Bảng dữ liệu tổng hợp sau khi gộp thông tin thực phẩm và dữ liệu dinh dưỡng

Đây là bảng hoàn chỉnh dùng cho phân tích và trực quan hóa.

### 3.4.4. Phân tích và kết quả

**a. Phân bố nhóm thực phẩm**

Thống kê bằng value\_counts() cho thấy các nhóm có số lượng thực phẩm nhiều nhất là:

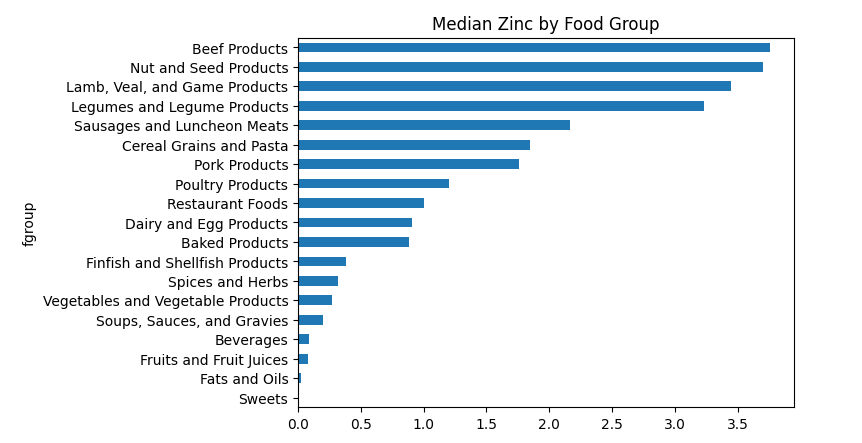
* Vegetables and Vegetable Products
* Beef Products
* Baked Products
* Breakfast
* Cereals

Điều này phản ánh USDA có độ bao phủ tốt với nhóm rau, thịt và ngũ cốc.

**b. Trung vị (median) giá trị dinh dưỡng theo nhóm**

Sử dụng groupby(["nutrient", "fgroup"])["value"].quantile(0.5)cho phép so sánh mức dinh dưỡng trung vị giữa các nhóm thực phẩm.

Ví dụ, hàm lượng **Zinc (kẽm)** trung vị cao nhất thuộc về nhóm **Beef Products**, thể hiện đúng thực tế dinh dưỡng.



Hình 4. Biểu đồ so sánh hàm lượng kẽm trung vị giữa các nhóm thực phẩm

**Nhận xét :**

* Phân tích này cho phép so sánh mức dinh dưỡng giữa các nhóm thực phẩm.
* Median được chọn thay vì mean để tránh ảnh hưởng bởi giá trị ngoại lai.

### 3.4.5. Thực phẩm giàu nhất theo từng chất dinh dưỡng

Nhóm dữ liệu theo (nutgroup, nutrient) rồi chọn hàng có value lớn nhất (idxmax).  
Kết quả cho nhóm **“Amino Acids”** như sau:

| **nutrient** | **food** |
| --- | --- |
| Alanine | Gelatins, dry powder, unsweetened |
| Arginine | Seeds, sesame flour, low-fat |
| Aspartic acid | Soy protein isolate |
| Cystine | Seeds, cottonseed flour, low fat (glandless) |
| Glutamic acid | Soy protein isolate |
| Glycine | Gelatins, dry powder, unsweetened |
| Histidine | Whale, beluga, meat, dried (Alaska Native) |
| Hydroxyproline | KENTUCKY FRIED CHICKEN, Fried Chicken, ORIGINAL RE |
| Isoleucine | Soy protein isolate |
| Leucine | Soy protein isolate |
| Lysine | Seal, bearded (Oogruk), meat, dried (Alaska Native) |
| Methionine | Fish, cod, Atlantic, dried and salted |
| Phenylalanine | Soy protein isolate |
| Proline | Gelatins, dry powder, unsweetened |
| Serine | Soy protein isolate |
| Threonine | Soy protein isolate |
| Tryptophan | Sea lion, Steller, meat with fat (Alaska Native) |
| Tyrosine | Soy protein isolate |
| Valine | Soy protein isolate |

**Nhận xét:**

* Gelatins là nguồn giàu nhất của Alanine, Glycine, Proline.
* Soy protein isolate (đạm đậu nành tinh khiết) là nguồn giàu hầu hết axit amin còn lại.
* Một số loại thịt cá biển hoặc động vật hoang dã xuất hiện (Whale, Seal, Sea lion) do dữ liệu của USDA bao gồm cả thực phẩm vùng Alaska.

### 3.4.6. Kết luận

Phần 13.4 cho thấy quy trình đầy đủ của xử lý dữ liệu JSON phức tạp trong thực tế:

* + Nạp và hiểu cấu trúc dữ liệu.
  + Trích xuất và làm phẳng dữ liệu lồng nhau.
  + Chuẩn hóa và gộp dữ liệu.
  + Thực hiện phân tích mô tả, thống kê, và trực quan hóa.

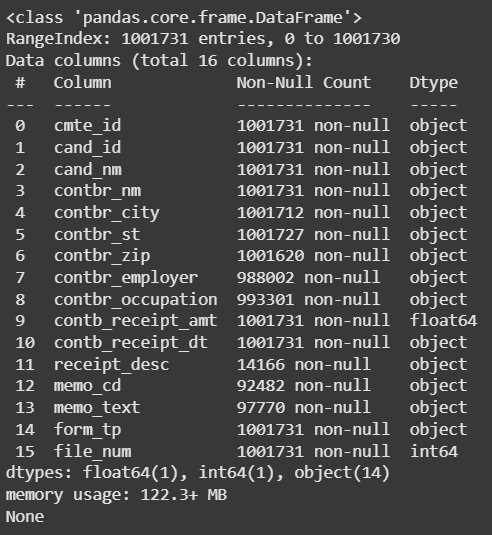
Kết quả cuối cùng giúp rút ra được những insight thực tế về thành phần dinh dưỡng:

* + Thực phẩm nào giàu protein hoặc axit amin nhất.
  + Nhóm thực phẩm nào có giá trị dinh dưỡng trung bình cao nhất.

Bài thực hành này là ví dụ rất điển hình về data wrangling & exploratory data analysis (EDA) trong khoa học dữ liệu.

## 3.4. Dataset 5 – 2012 Federal Election Commission (FEC) Data

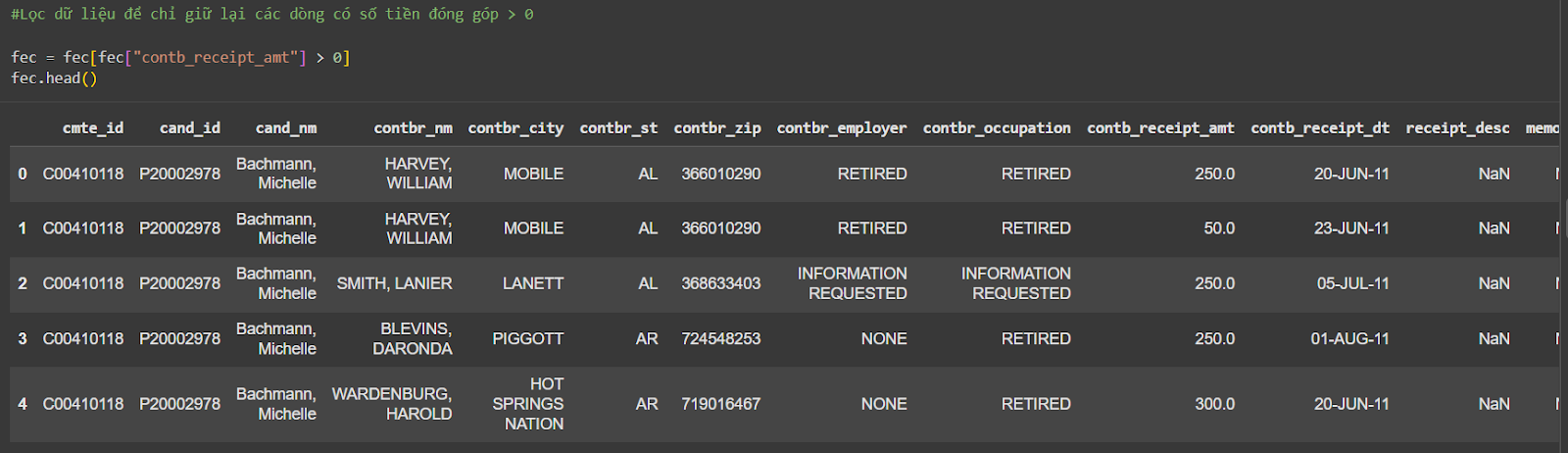
Đọc dữ liệu data từ nguồn Internet:



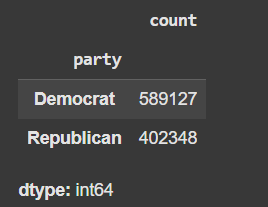
**Nhận xét:**

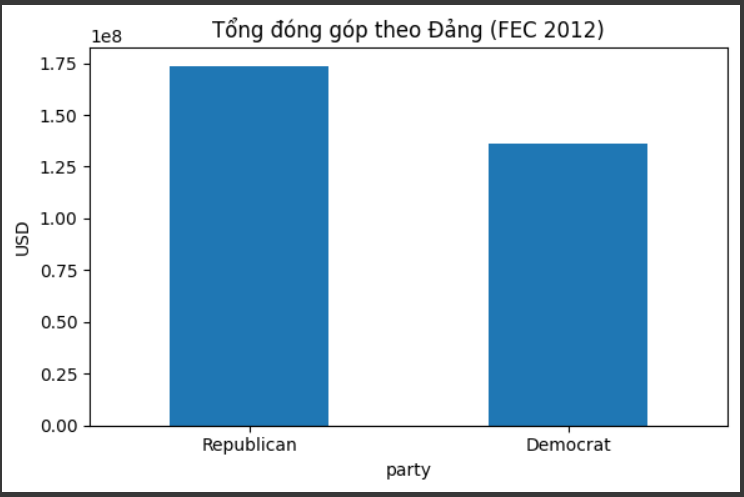
* Dữ liệu có 16 cột, gồm: ID ủy ban, tên ứng viên, người đóng góp, nghề nghiệp, nơi làm việc, thành phố, bang, số tiền, ngày, v.v.
* Có cột contb\_receipt\_amt (số tiền) dạng float64, file\_num dạng int64.
* Quy mô dữ liệu lớn: hơn 1 triệu dòng, thể hiện rõ quy mô của hệ thống tài trợ chính trị Mỹ.
* Không có thông tin đảng phái ban đầu, cần tự bổ sung thủ công bằng ánh xạ ứng viên.
* Dữ liệu thô chứa hoàn tiền (âm) → cần lọc trước khi phân tích.
* Dữ liệu cốt lõi **đầy đủ**: cand\_nm, contb\_receipt\_amt, contb\_receipt\_dt, form\_tp, cmte\_id, cand\_id, file\_num không thiếu — tốt cho phân tích chính.
* Thiếu nhiều ở các cột mô tả tự do: receipt\_desc ~**98.6%**, memo\_cd ~**90.9%**, memo\_text ~**90.2%** → thường **bỏ qua** trong phân tích định lượng.

**Lọc dữ liệu giữ lại các dòng có contb\_receipt\_amt hợp lệ:**



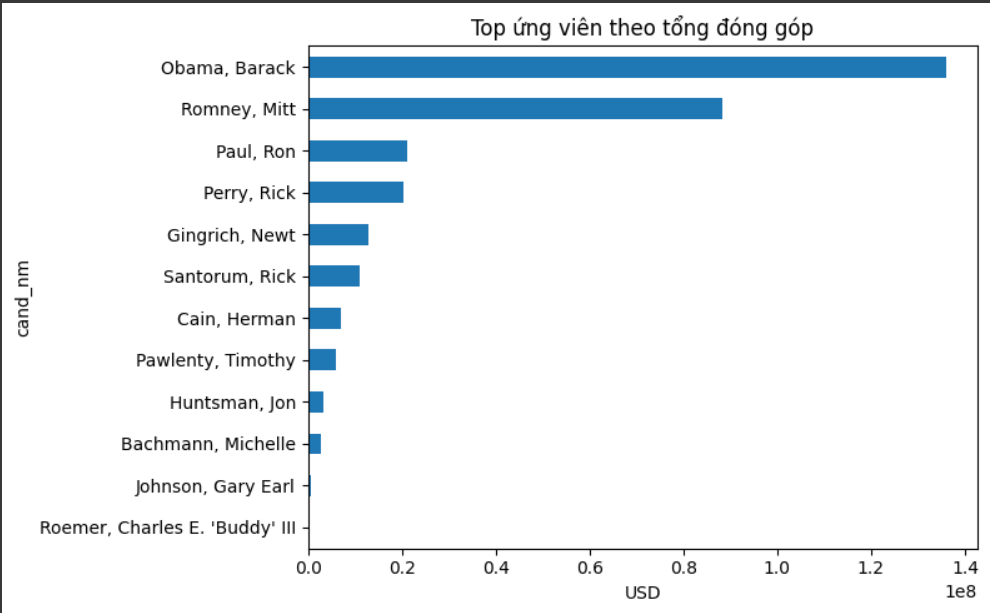
**Số lượng giao dịch đóng góp (số dòng) được gán nhãn đảng qua cột party (map từ tên ứng viên):**





**Nhận xét:**

* Tổng đóng góp của Republican cao hơn Democrat khá rõ (chênh khoảng 30–35%).
* Đây là tổng USD của mọi giao dịch dương, gộp tất cả ứng viên mỗi đảng; không cho biết số người ủng hộ.
* Kết quả phụ thuộc quy tắc gán đảng và vài khoản hiến tặng lớn; nên soi thêm phân phối mức tiền và theo thời gian/bang.



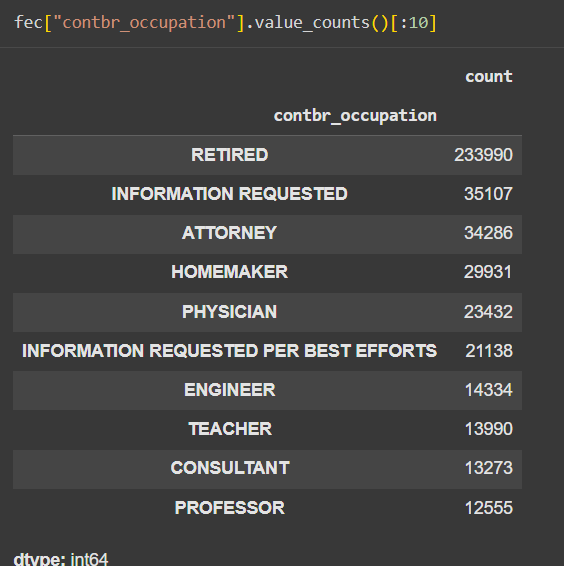
**Nhận xét:**

* Hai ứng viên chính chiếm phần lớn đóng góp: Obama (Democrat) và Romney (Republican).
* Obama dẫn đầu rõ rệt về tổng USD, cao hơn Romney khoảng ~50% theo biểu đồ.
* Sau hai ứng viên top, các ứng viên còn lại đóng góp rất nhỏ (đuôi dài), cho thấy huy động tập trung vào Obama và Romney.
* Nên soi thêm phân phối mức tiền và diễn biến theo thời gian/bang (hoặc theo donor duy nhất) để tách hiệu ứng nhiều giao dịch nhỏ vs. vài khoản lớn.

**Kết luận:**

* Dữ liệu FEC là kho vàng cho phân tích hành vi chính trị – ai đóng góp, bao nhiêu, ở đâu.
* Tự thêm đảng phái giúp biến dữ liệu rời rạc thành dữ liệu “ngữ nghĩa”, phục vụ phân tích ý thức hệ.
* Obama vs Romney đại diện hai mô hình tài trợ khác nhau:
  + Obama (Democrat): nhiều khoản nhỏ, từ nhiều cá nhân → phản ánh mạng lưới tài trợ đại chúng.
  + Romney (Republican): ít khoản hơn nhưng giá trị trung bình cao hơn → phản ánh nhóm tài trợ giàu có, tổ chức.
* Làm sạch dữ liệu là bắt buộc: loại hoàn tiền, chuẩn hóa text (tên nghề nghiệp, công ty) trước khi thống kê.
* Bộ dữ liệu này là ví dụ điển hình cho việc dùng pandas trong phân tích dữ liệu thật: đọc, lọc, gắn nhãn, nhóm và khám phá mẫu hình.

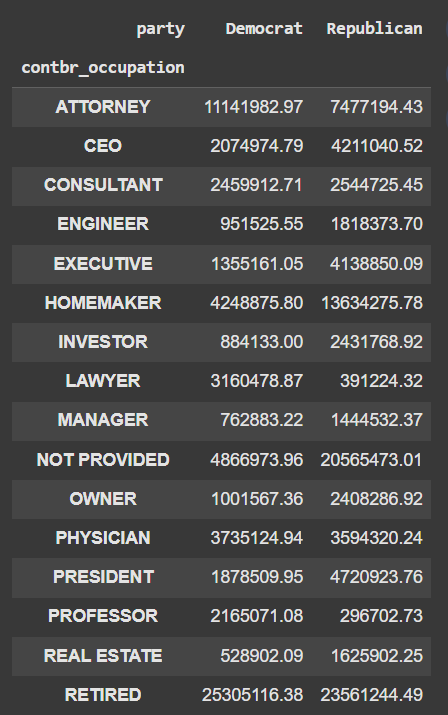
### 3.5.1 Donation Statistics by Occupation and Employer

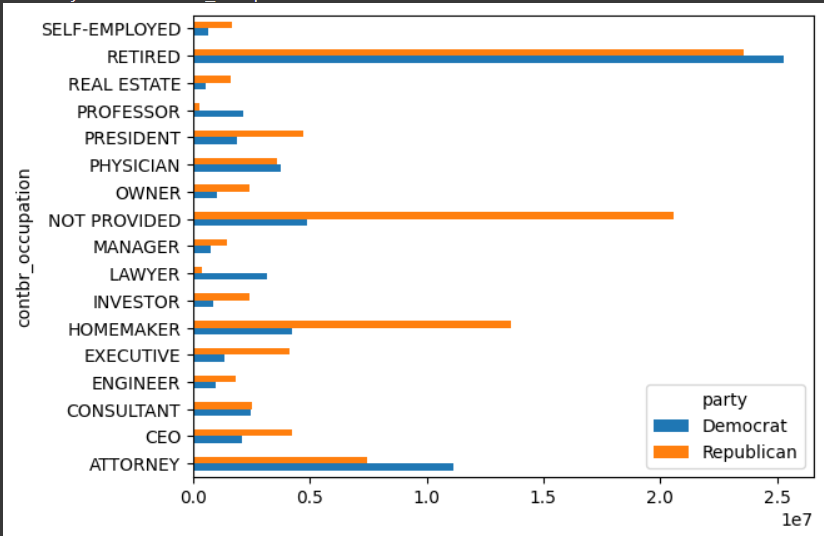


**Nhận xét:**

* Đây là top nghề nghiệp theo số giao dịch (không phải số người, cũng không phải tổng tiền). RETIRED áp đảo ⇒ nhóm về hưu đóng góp rất nhiều giao dịch.
* INFORMATION REQUESTED / …PER BEST EFFORTS là giá trị placeholder (thiếu/không cung cấp). Nên gộp về một nhãn như NOT PROVIDED để tránh méo thống kê.
* Các nghề thu nhập cao/chuyên môn (ATTORNEY, PHYSICIAN, ENGINEER, CONSULTANT, PROFESSOR) xuất hiện dày, phù hợp kỳ vọng về khả năng hiến tặng.
* HOMEMAKER cho thấy đóng góp hộ gia đình/không đi làm cũng đáng kể.

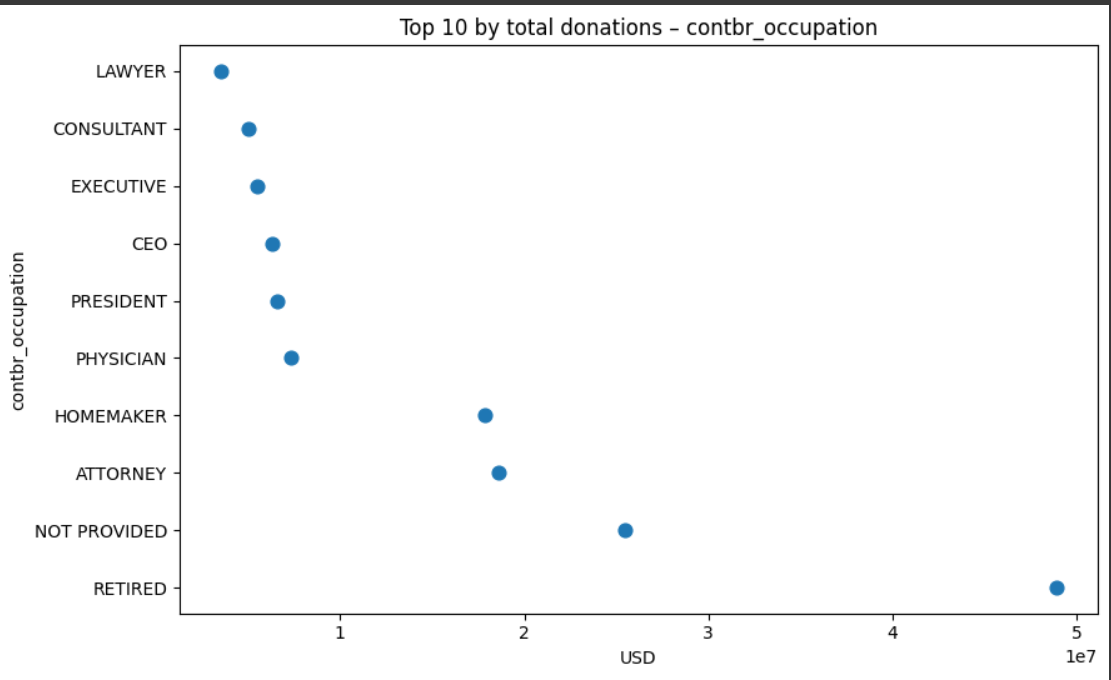
Lọc ra các nghề nghiệp có tổng số tiền quyên góp trên 2 triệu USD, phân theo đảng phái:





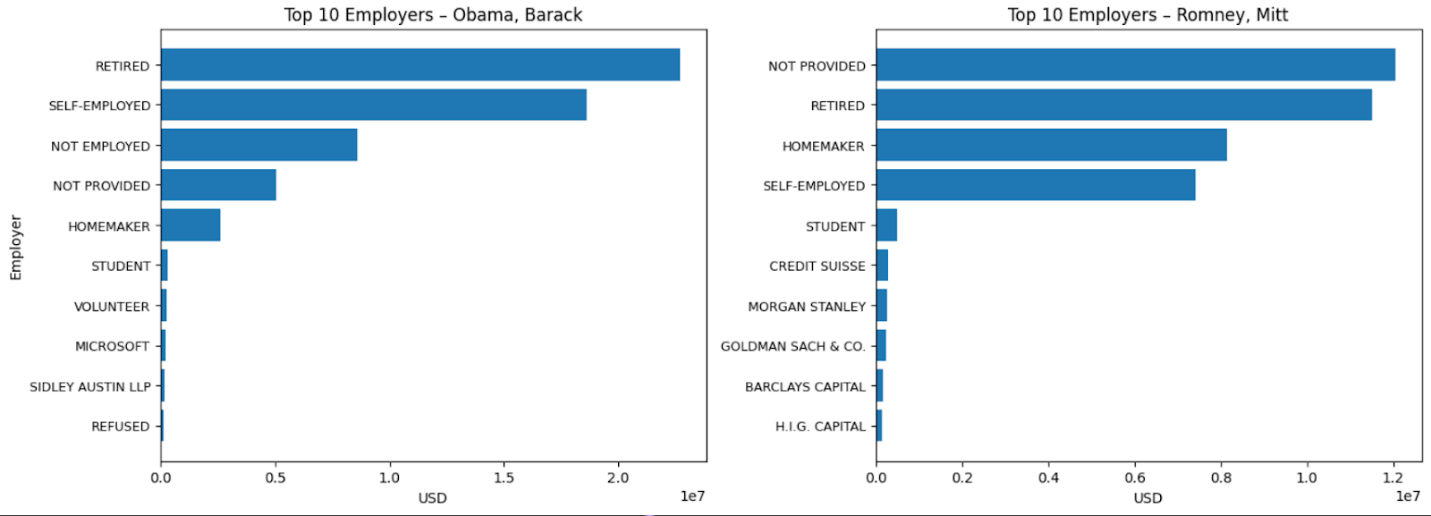
**Nhận xét:**

* RETIRED đóng góp lớn nhất cho cả hai đảng, hơi nghiêng về Democrat.
* ATTORNEY/Lawyer và PROFESSOR nghiêng rõ về Democrat → phù hợp “giới chuyên môn/pháp lý – học thuật” thiên D.
* CEO/EXECUTIVE/PRESIDENT/OWNER/REAL ESTATE thiên mạnh về Republican → nhóm điều hành/kinh doanh ủng hộ R nhiều hơn.
* HOMEMAKER nổi bật về Republican (rất cao), là khác biệt lớn giữa hai đảng.  
  ENGINEER và INVESTOR/MANAGER cũng lệch Republican, còn PHYSICIAN tương đối cân bằng (nhẹ về D).
* Nhãn NOT PROVIDED nghiêng Republican cho thấy nhiễu dữ liệu (không khai báo nghề); nên giữ cảnh giác khi diễn giải.



**Nhận xét:**

* RETIRED vượt trội (~5×10⁷ USD) → nhóm nhà tài trợ lớn nhất; kế đến là NOT PROVIDED (~2.6×10⁷) cho thấy nhiễu do thiếu nghề (nên gộp/loại khi phân tích).
* ATTORNEY & HOMEMAKER cũng rất lớn (~1.8×10⁷), còn CEO/EXECUTIVE/PRESIDENT ở mức trung bình (~5–9×10⁶) và LAWYER/CONSULTANT thấp hơn.
* Phân phối lệch phải mạnh (một vài nhóm chi phối). Để đọc dễ hơn, bạn có thể: dùng log-scale trục X, thêm nhãn giá trị, hoặc tách theo đảng để thấy xu hướng nghề ↔ đảng.



**Nhận xét:**

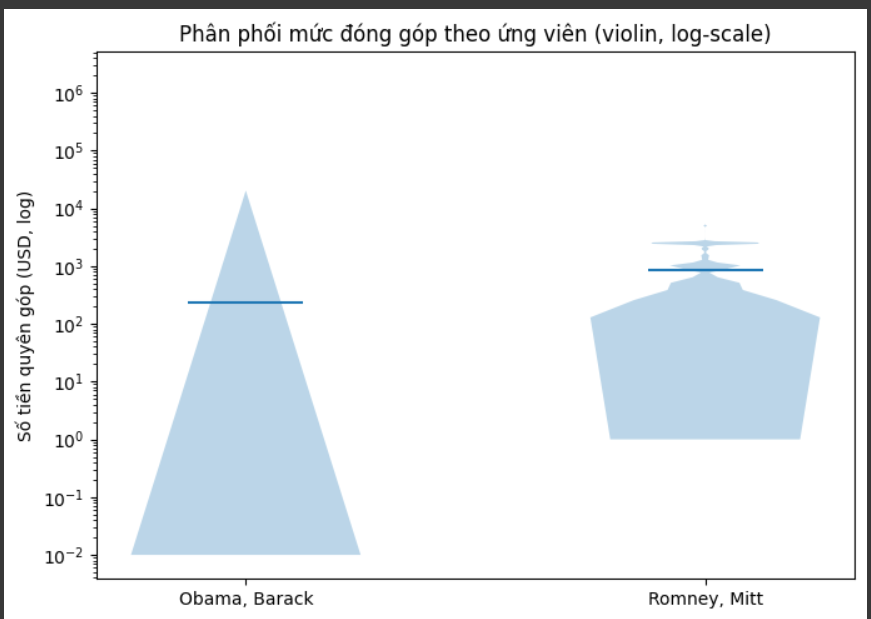
* Cả hai ứng viên đều dựa nhiều vào cá nhân: nhóm Retired và Self-Employed/Not Employed/Homemaker chiếm phần lớn tổng USD. Đây là đóng góp của cá nhân khai báo nơi làm việc, không phải công ty quyên góp trực tiếp.
* Romney có “NOT PROVIDED” rất lớn → dữ liệu thiếu/không khai báo employer gây méo so sánh; nên gộp/loại nhãn này khi phân tích.
* Dấu ấn ngành tài chính ở Romney (Credit Suisse, Morgan Stanley, Goldman Sachs, Barclays, H.I.G. Capital) xuất hiện trong top; trong khi Obama có vài tên gắn với công nghệ/pháp lý (Microsoft, Sidley Austin LLP), phù hợp trực giác về cơ cấu donor hai bên.
* Khuyến nghị: chuẩn hoá employer (SELF ↔ SELF-EMPLOYED, typo), thử log-scale trục X để bớt lệch phải, và phân tích thêm theo đảng/ban thời gian hoặc theo donor duy nhất để tránh một employer bị chi phối bởi vài khoản rất lớn.

**Kết luận:**

Hai liên minh tài trợ khác nhau hiện ra rõ nét. Phía Dân chủ huy động mạnh từ giới chuyên môn (luật sư, giảng dạy, học thuật) và nhóm retired lớn; số tiền cân hơn ở các nghề có kỹ năng/giáo dục cao. Phía Cộng hòa nổi bật ở khối quản trị/doanh nhân (executive, CEO, president), engineer, và đặc biệt homemaker. Cả hai đều dựa nhiều vào retired, nhưng Republicans có khối executive/homemaker lớn và tỷ trọng bản ghi occupation/employer “không cung cấp/thông tin yêu cầu” cao hơn, phản ánh khác biệt trong cách thu thập/kê khai của chiến dịch.

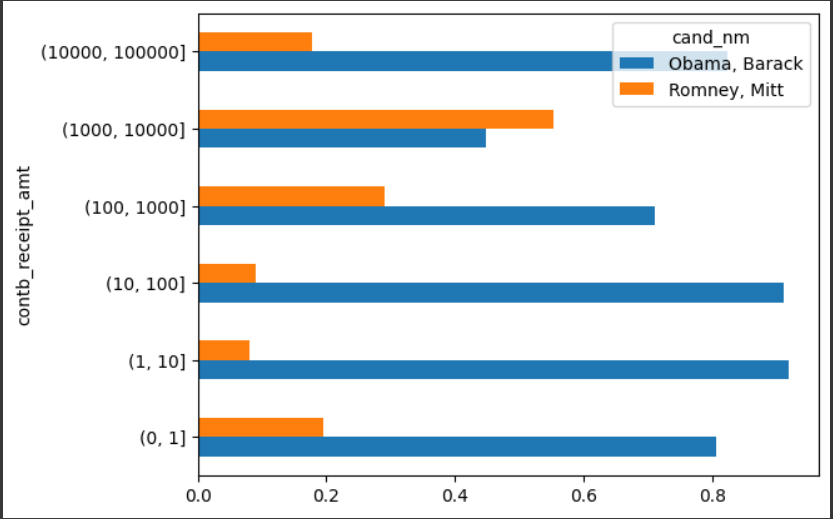
Tóm lại, cơ cấu nghề nghiệp của donor không đồng nhất giữa hai đảng: Democrat nghiêng về chuyên môn–học thuật, Republican nghiêng về quản trị–kinh doanh và homemaker, còn retired là “xương sống” ở cả hai bên.

### 3.5.2 Bucketing Donation Amounts



**Nhận xét:**

* Biểu đồ violin ở thang log cho thấy phân phối rất lệch phải: cả hai bên có “đuôi” khoản rất lớn, nhưng Obama có phạm vi rộng hơn (xuống tận ~10⁻² USD và lên >10⁴), còn Romney tập trung chặt hơn quanh vài trăm–vài nghìn USD.
* Đường ngang là giá trị trung bình: có vẻ Romney có mức trung bình cao hơn, nghĩa là khoản quyên góp điển hình lớn hơn dù độ phân tán nhỏ hơn.  
  Giá trị cực nhỏ ở phía Obama (≈ 0.01 USD) có thể là nhiễu (làm tròn, lệ phí, giao dịch đặc biệt). Nếu bạn đã lọc contb\_receipt\_amt > 0, cân nhắc lọc tối thiểu (ví dụ >= $1) hoặc winsorize/cắt theo bách phân vị (1–99%) để biểu đồ bớt méo.



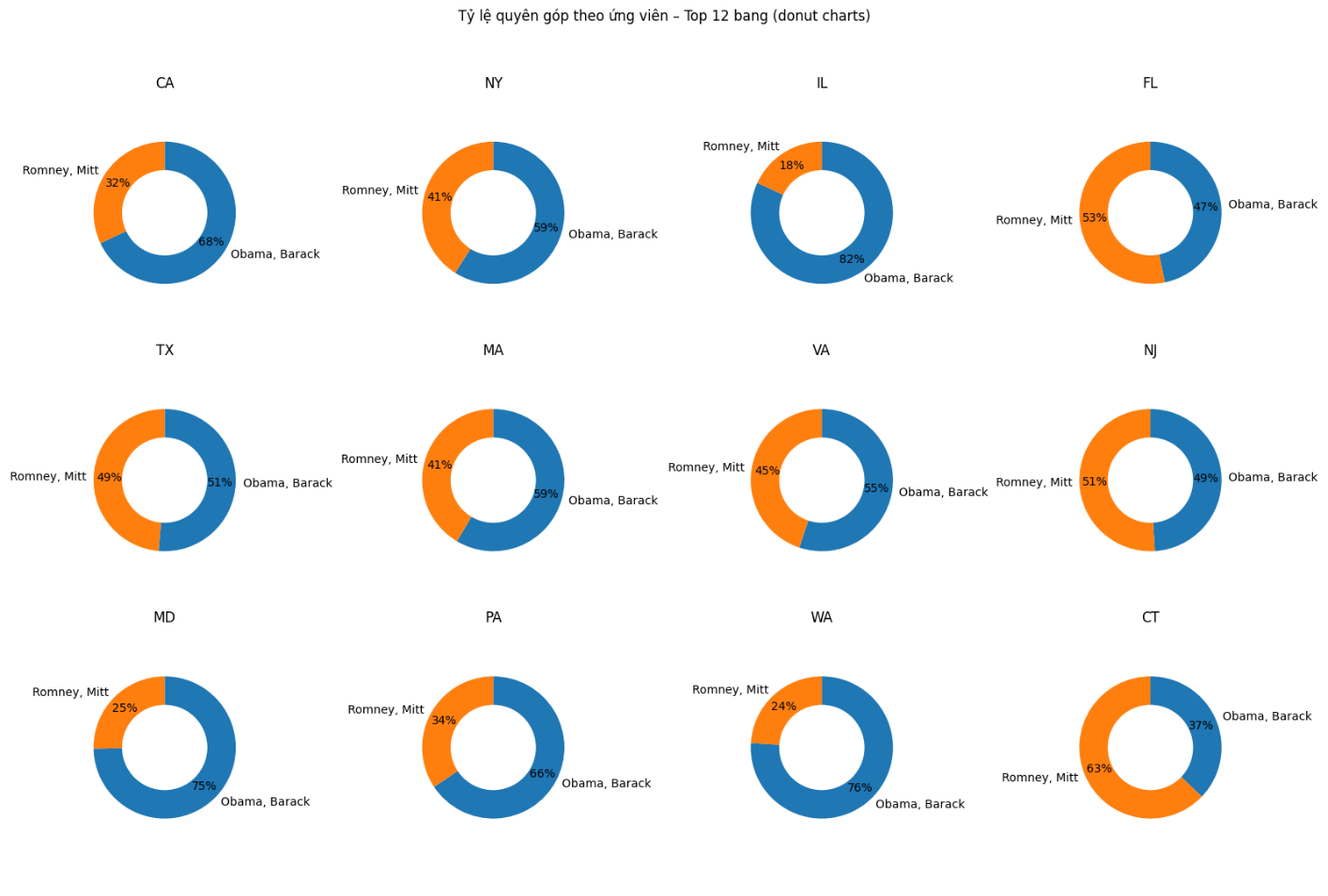
**Nhận xét:**

* Nhìn chung Obama áp đảo ở các bin nhỏ: (0,1], (1,10], (10,100], và đặc biệt (100,1000] → dấu hiệu nền tảng small-dollar mạnh.
* Romney nổi bật hơn ở (1000, 10000] (donations cỡ 1–10 nghìn USD) → tỷ trọng khoản lớn trung bình cao hơn. Ở (10000, 100000] Obama lại trội.
* Cách nhìn này nhất quán với trực giác: Obama nhiều donation nhỏ/đều, Romney tập trung hơn ở mid-to-high dollar.

**Kết luận:**

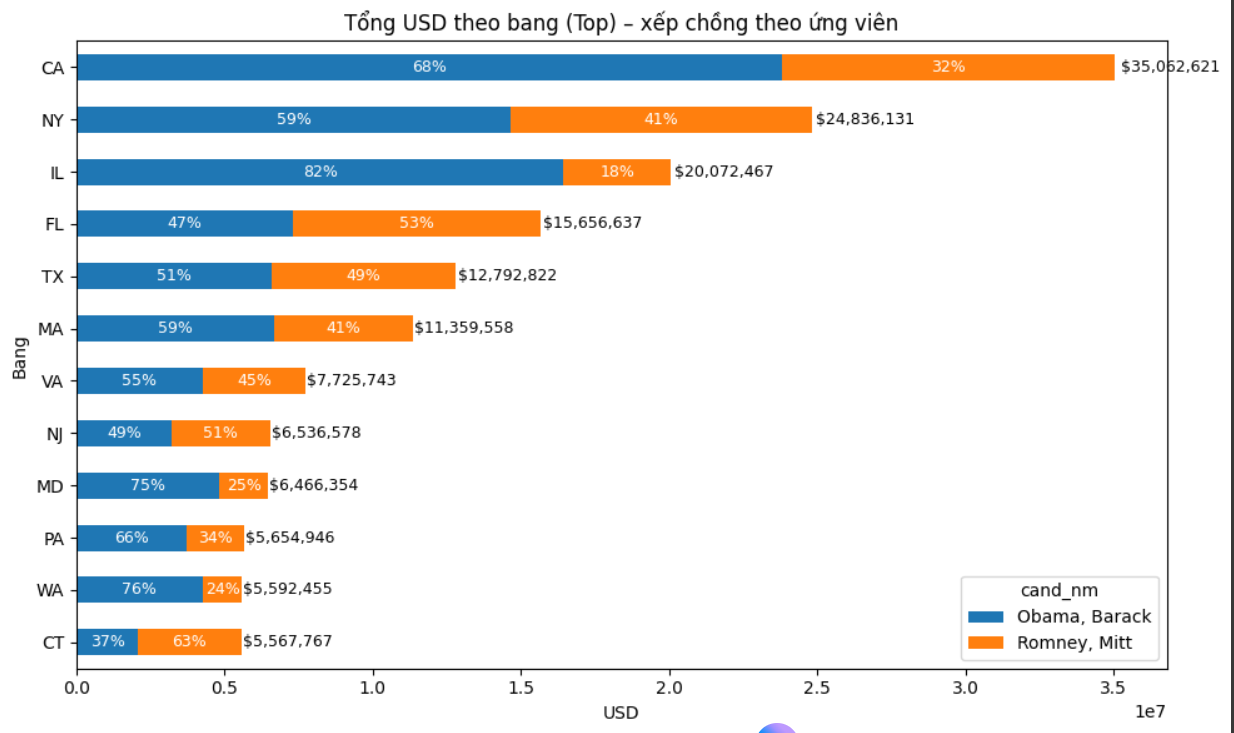
Chiến dịch 2012 cho thấy hai chiến lược huy động khác nhau: Obama xây nền tảng grassroots với vô số khoản nhỏ, tạo độ phủ và tính bền vững; Romney dựa nhiều hơn vào nhà tài trợ giá trị cao. Việc “bucket hóa” làm lộ rõ cấu trúc nguồn tiền, gợi ý các chiến dịch nên tối ưu kênh và thông điệp theo phân khúc mức đóng góp. (Bước nâng cao: gộp theo donor để phân biệt “nhiều khoản nhỏ lặp” vs “một khoản lớn”.)

**3.5.3 Donation Statistics by State**



**Nhận xét:**

* Xanh đậm ở bờ Tây/Đông: CA (~68%), NY (~59%), IL (~82%), WA (~76%), MD (~75%), PA (~66%) nghiêng mạnh Obama. Nghiêng về Romney: FL (~53%), NJ (~51%), CT (~63%) – đáng chú ý CT (trung tâm tài chính) lệch R khá rõ.
* Sát nút: TX (~51% Obama vs 49% Romney) và VA (~55% Obama) cho thấy bang lớn/swing vẫn cạnh tranh.
* Đây là tỷ trọng USD đóng góp, không phải số phiếu; phản ánh cơ cấu donor (thu nhập/ngành nghề/đô thị) hơn là cục diện bầu cử.



**Nhận xét:**

* Quy mô chênh lệch lớn: CA dẫn tuyệt đối (~$35M), tiếp theo NY, IL; các bang còn lại nhỏ hơn đáng kể → khi đọc phần trăm nên nhớ khác biệt về tổng USD.
* Nghiêng Obama mạnh: IL (82%), WA (76%), MD (75%), CA (68%), PA (66%).
* Nghiêng Romney: CT (63%), FL (53%), NJ (51%); TX gần cân bằng (51%–49%).  
  Nhãn % trên thanh + tổng USD ngoài cùng giúp thấy cả cơ cấu và quy mô—rất ổn để so sánh giữa bang.

**Kết luận:**

Bản đồ gây quỹ cho thấy hai hệ sinh thái ủng hộ khác nhau: Obama huy động mạnh ở các trung tâm đô thị, công nghệ, chính phủ và học thuật (DC, California…), còn Romney nổi bật ở các vùng tài chính/giới doanh nhân và cộng đồng hưu trí (Connecticut, Arizona, Florida). Dòng tiền phản ánh mạng lưới xã hội–kinh tế hơn là kết quả bầu cử trực tiếp; vì thế cần nhớ rằng donations ≠ votes và chịu ảnh hưởng bởi mật độ nhà tài trợ, thu nhập, chi phí sống và văn hóa quyên góp của từng bang.

# IV. Appendix Advanced Numpy

## 4.1 ndarray Object Internals

### 4.1.1 Mục tiêu

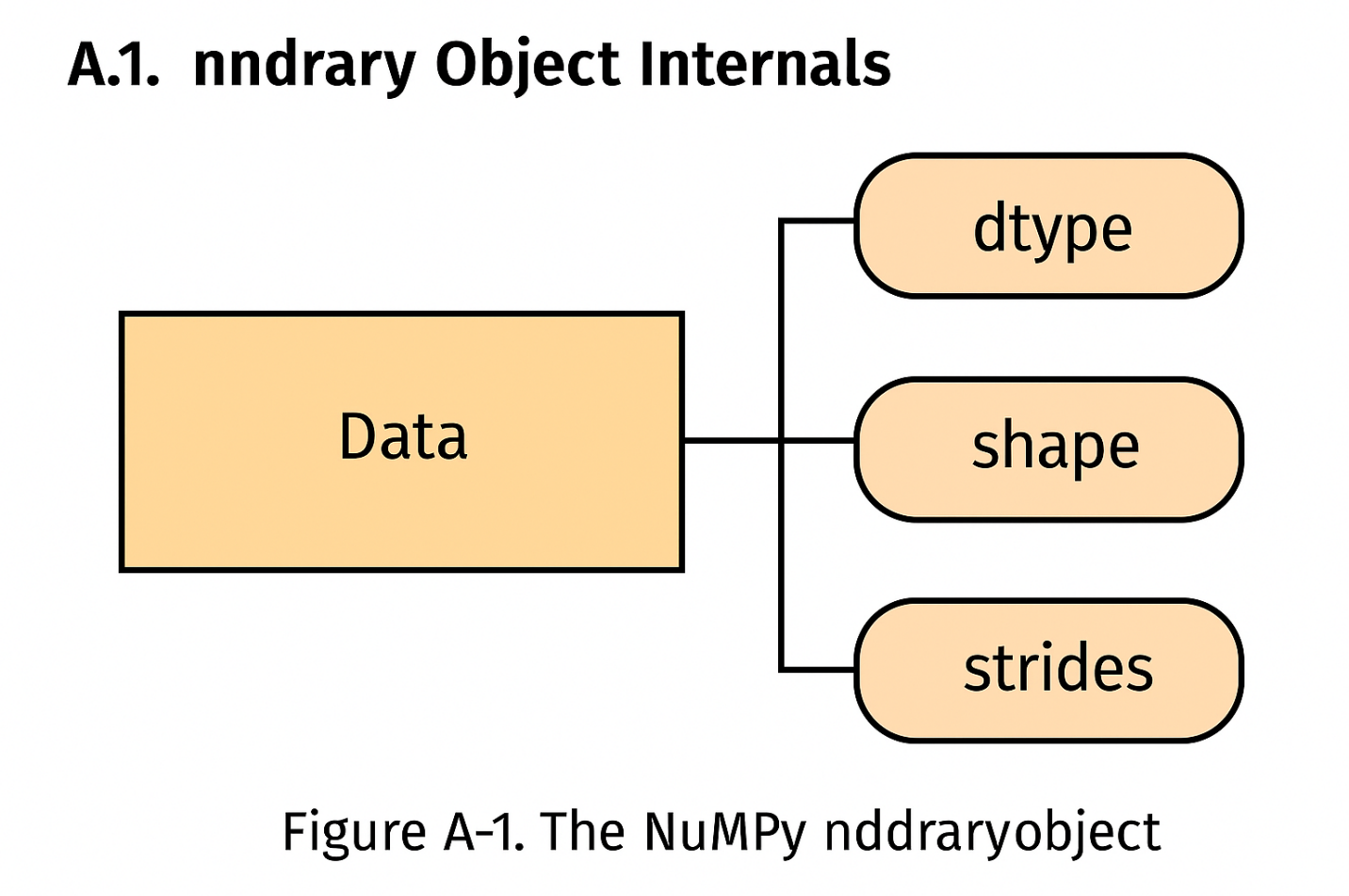
Tìm hiểu cấu trúc nội bộ của đối tượng mảng (ndarray) trong NumPy — cách NumPy quản lý bộ nhớ và định nghĩa dữ liệu.

Mục tiêu chính:

* Hiểu cơ chế lưu trữ và truy cập dữ liệu của ndarray.
* Nắm được khái niệm dtype, shape, strides, và data pointer.
* Phân tích cách NumPy có thể tạo các view (không sao chép dữ liệu) chỉ bằng cách thay đổi strides.
* Giải thích hệ thống phân cấp kiểu dữ liệu (NumPy Data Type Hierarchy)

### 4.1.2 Nội dung & Giải thích chi tiết

**a. Giới thiệu về ndarray:**



Đối tượng ndarray trong NumPy là một cách diễn giải (interpretation) của một khối dữ liệu đồng nhất (homogeneously typed data), có thể được lưu liên tục (contiguous) hoặc theo bước nhảy (strided) trong bộ nhớ.

Mỗi phần tử của mảng có cùng kiểu dữ liệu, được mô tả bởi dtype (data type).

Cấu trúc ndarray gồm bốn thành phần chính:

|  |  |
| --- | --- |
| Thành phần | Mô tả |
| Data Pointer | Con trỏ đến vùng dữ liệu thực tế trong RAM (hoặc trong tệp memory-mapped). |
| Dtype | Xác định kiểu phần tử – ví dụ float64, int32, bool... |
| Shape | Bộ số nguyên thể hiện kích thước mảng theo từng chiều, ví dụ (10, 5) hoặc (3, 4, 5). |
| Strides | Bộ số byte cần di chuyển để qua 1 phần tử trên mỗi chiều. |

**b. Strided View và Cơ chế “Zero-Copy”**

Một trong những điểm mạnh của ndarray là nó không chỉ là khối dữ liệu, mà là một lớp “view” có bước nhảy (stride) trên dữ liệu gốc.

Điều này cho phép NumPy tạo mảng con (subarray) mà không sao chép dữ liệu thật.

**c. Ví dụ về Shape và Strides:**



**Giải thích:**

* np.ones((10, 5)) tạo mảng 2 chiều gồm 10 hàng và 5 cột, tất cả giá trị đều bằng 1.
* .shape trả về kích thước (shape) của mảng dưới dạng bộ số (10, 5).



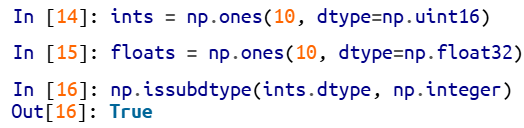
**Giải thích:**

* np.ones((3, 4, 5), dtype=np.float64) tạo mảng 3 chiều có kích thước (3, 4, 5) = 3 khối, mỗi khối có 4 hàng × 5 cột.
* Mỗi phần tử float64 chiếm 8 byte trong bộ nhớ.
* .strides cho biết số byte cần nhảy (step) để di chuyển đến phần tử kế tiếp theo từng chiều.

**d. NumPy Data Type Hierarchy**

Ngoài ndarray, NumPy còn định nghĩa hệ thống phân cấp kiểu dữ liệu (dtype hierarchy). Mỗi kiểu cụ thể (int16, float64, bool\_ …) là một lớp con của nhóm tổng quát hơn như np.integer, np.floating, np.number.

Ví dụ:



Giải thích:

* ints là mảng gồm 10 phần tử, kiểu dữ liệu unsigned 16-bit integer (np.uint16).
* floats là mảng gồm 10 phần tử, kiểu 32-bit float (np.float32).
* Hàm np.issubdtype() dùng để kiểm tra mối quan hệ kiểu dữ liệu (dtype).
* Câu lệnh np.issubdtype(ints.dtype, np.integer)→ kiểm tra xem kiểu của ints (uint16) có phải là một kiểu con của nhóm np.integer không.
* Kết quả True, vì uint16 là một kiểu số nguyên (integer) trong hệ dtype của NumPy.

**4.1.3 Kết quả & Nhận xét**

**Kết quả:**

* ndarray không chỉ là một vùng nhớ mà còn có thông tin strides, dtype, shape, giúp truy cập dữ liệu nhanh, linh hoạt.
* Strides quyết định cách di chuyển qua bộ nhớ, có thể âm hoặc dương.
* NumPy cho phép tạo views (không copy dữ liệu) → tiết kiệm bộ nhớ, tăng hiệu năng.
* Hệ thống dtype hierarchy giúp kiểm tra kiểu dữ liệu nhất quán, hỗ trợ xử lý tổng quát.

**Nhận xét:**

* Cấu trúc nội bộ của ndarray là yếu tố cốt lõi giúp NumPy hiệu quả gấp nhiều lần list Python.
* Việc hiểu rõ shape–strides–dtype giúp tối ưu bộ nhớ và tốc độ tính toán.
* Phân cấp dtype là công cụ mạnh mẽ giúp viết code tổng quát, mở rộng, và kiểm tra dữ liệu nhanh.
* Nhờ cơ chế “zero-copy view”, NumPy trở thành nền tảng cho các thư viện cao cấp như Pandas, TensorFlow, PyTorch.

## 4.2 Advanced Array Manipulation

### 4.2.1 Mục tiêu

Phần này giới thiệu các kỹ thuật thao tác nâng cao trên mảng NumPy (ndarray), vượt ra ngoài các thao tác cơ bản như slicing, indexing và Boolean filtering.

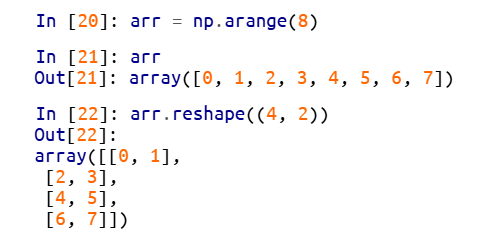
Mục tiêu là giúp hiểu cách:

* Thay đổi hình dạng (reshape) mảng mà không cần sao chép dữ liệu.
* Nối, tách, lặp, hoặc kết hợp các mảng hiệu quả.
* Kiểm soát cách dữ liệu được duyệt trong bộ nhớ (C order và Fortran order).
* Sử dụng các hàm nâng cao như tile, repeat, take, put.

### Các bước & Khái niệm chính

**a. Reshaping Arrays – Thay đổi hình dạng mảng**

* Dùng phương thức .reshape() để chuyển mảng sang kích thước mới mà không sao chép dữ liệu, nếu dữ liệu trong bộ nhớ liên tục (contiguous).
* Ví dụ:



**Giải thích:**

* np.arange(8) → Tạo một mảng 1 chiều gồm 8 phần tử liên tiếp từ 0 đến 7.
* arr.reshape((4, 2)) → Thay đổi hình dạng (shape) của mảng thành ma trận 4 hàng × 2 cột, không sao chép dữ liệu.NumPy chỉ “nhìn” lại vùng dữ liệu gốc theo dạng 2D.

**b. Flattening và Raveling (Làm phẳng mảng)**

* Chuyển mảng nhiều chiều về mảng 1 chiều:
  + ravel() → trả về view (không copy dữ liệu nếu có thể).
  + flatten() → luôn tạo bản sao dữ liệu.
* Ví dụ:

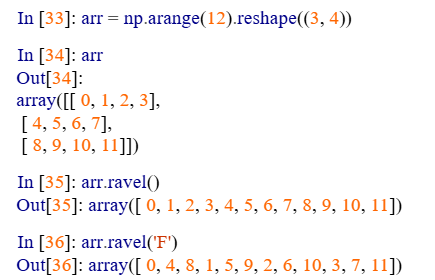


**Giải thích:**

* np.arange(15)→ Tạo mảng gồm các số từ 0 đến 14 (tổng 15 phần tử).
* .reshape((5, 3))→ Chuyển mảng 1D thành ma trận 2D kích thước 5 hàng × 3 cột, không sao chép dữ liệu. → Mỗi hàng chứa 3 phần tử liên tiếp.
* .ravel()→ Làm phẳng (flatten) mảng nhiều chiều thành mảng 1 chiều. → Dữ liệu vẫn cùng vùng nhớ (không copy), chỉ “trải” các phần tử ra liên tục theo hàng (C-order).

**c. C Versus và FORTRAN Order (Thứ tự lưu trữ trong bộ nhớ)**

* NumPy lưu trữ dữ liệu trong bộ nhớ theo:
* C/Row-major order (mặc định): các phần tử trong mỗi hàng nằm liền nhau trong bộ nhớ.
* Fortran/Column-major order: các phần tử trong mỗi cột nằm liền nhau.
* Ví dụ:



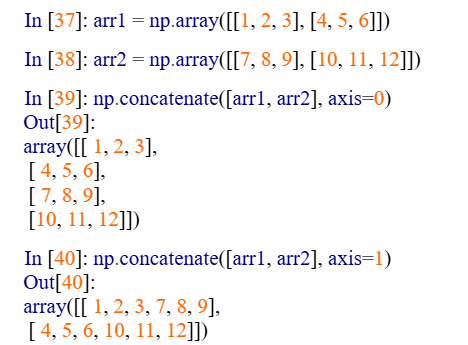
**Giải thích:**

* np.arange(12).reshape((3, 4))→ Tạo mảng 12 phần tử từ 0 → 11, sau đó reshape thành 3 hàng × 4 cột
* arr.ravel()→ Làm phẳng (flatten) mảng theo C-order (row-major order) – tức là đọc theo hàng (row by row).
* arr.ravel('F')→ Làm phẳng mảng theo Fortran order (column-major order) – tức là đọc theo cột (column by column).

**d. Concatenating và Splitting Arrays**

Ghép mảng (concatenate)

* Ví dụ:



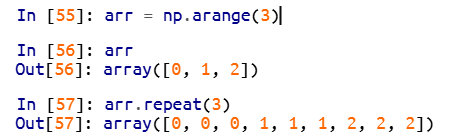
**Giải thích:**

* Tạo 2 mảng ban đầu → Cả hai mảng đều có kích thước (2, 3) (2 hàng, 3 cột).
* np.concatenate([...], axis=0)
* Ghép theo hàng (vertical stacking) → thêm mảng arr2 bên dưới arr1.
* Trục 0 (axis=0) là chiều hàng, nên số cột phải bằng nhau.
* np.concatenate([...], axis=1)
* Ghép theo cột (horizontal stacking) → đặt arr2 bên phải arr1.
* Trục 1 (axis=1) là chiều cột, nên số hàng phải bằng nhau.

**e. Repeating Elements – Hàm repeat và tile**

* repeat: lặp từng phần tử.

Ví dụ:



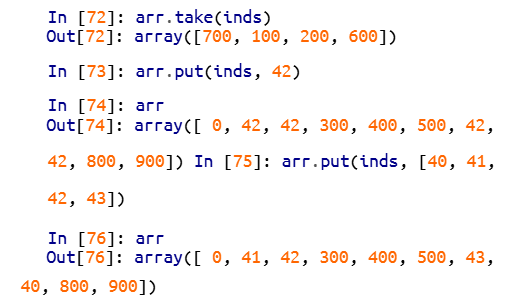
**Giải thích**:

* np.arange(3)→ Tạo mảng [0, 1, 2].
* arr.repeat(3)→ Mỗi phần tử trong arr được lặp lại 3 lần liên tiếp.NumPy sẽ nhân đôi các giá trị này trong mảng đầu ra, nhưng không thay đổi mảng gốc.

**f. Fancy Indexing Equivalents – take() và put()**

* Hai hàm này cho phép lấy hoặc gán giá trị theo vị trí (index) mà không cần dùng [].

Ví dụ:



**Giải thích:**

* arr.take(inds):
* Lấy các phần tử của arr tại vị trí (chỉ số) trong inds.
* Cụ thể: các phần tử tại [7, 1, 2, 6] là [700, 100, 200, 600].
* arr.put(inds, 42):
* Gán giá trị 42 vào tất cả các vị trí chỉ số trong inds.
* Các vị trí 7, 1, 2, 6 đều được thay thành 42.
* arr.put(inds, [40, 41, 42, 43])
* Gán mỗi vị trí một giá trị khác nhau tương ứng trong danh sách [40, 41, 42, 43].
* Cụ thể:
  + arr[7] = 40
  + arr[1] = 41
  + arr[2] = 42
  + arr[6] = 43

### 4.2.3 Kết quả & nhận xét:

**Kết quả:**

* Các thao tác reshape, ravel, và flatten giúp chuyển đổi linh hoạt hình dạng dữ liệu mà không cần sao chép.
* NumPy hỗ trợ nhiều cách lưu trữ và truy cập bộ nhớ (C/F order) để tương thích với các ngôn ngữ khác (Python, Fortran, C).
* Hàm concatenate, split, tile, repeat giúp xây dựng và biến đổi mảng phức tạp dễ dàng.
* Các phương thức take và put cung cấp truy cập nhanh và hiệu quả khi chỉ thao tác trên một trục cụ thể.

**Nhận xét:**

* Đây là phần mở rộng quan trọng, giúp NumPy trở nên mạnh mẽ trong xử lý dữ liệu dạng ma trận.
* Các thao tác reshape, split, tile, repeat rất hữu ích trong xử lý dữ liệu, hình ảnh, hoặc vector hóa.
* Hiểu rõ C vs Fortran order giúp tối ưu hiệu suất khi kết nối NumPy với các thư viện khoa học khác.
* take() và put() mang lại cách thao tác nhanh, an toàn và kiểm soát tốt bộ nhớ.

## 4.3 Broadcasting

### 4.3.1. Mục tiêu:

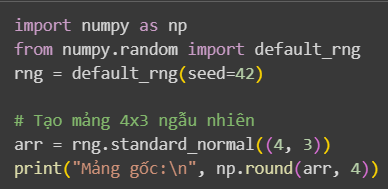
Phân tích và thực hành quy tắc Broadcasting của NumPy để:

* Hiểu cách NumPy mở rộng mảng khi kích thước không khớp.
* Ứng dụng trong xử lý dữ liệu: *demean*, chuẩn hóa, cộng/trừ ma trận nhiều chiều.

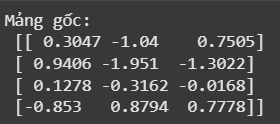
### 4.3.2. Quy trình thực hiện

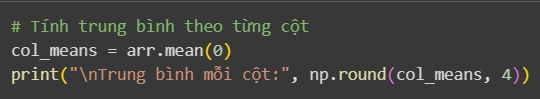
***Bước 1 – Nạp dữ liệu:***

* Tạo mảng 2D ngẫu nhiên arr(4x3) và tính trung bình từng cột.



Kết quả sau khi tạo mảng:



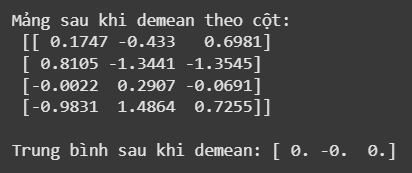


Kết quả sau khi tính trung bình:



***Bước 2 – Demean theo cột:***

Thực hiện arr - arr.mean(0) → broadcasting tự động theo trục 0.



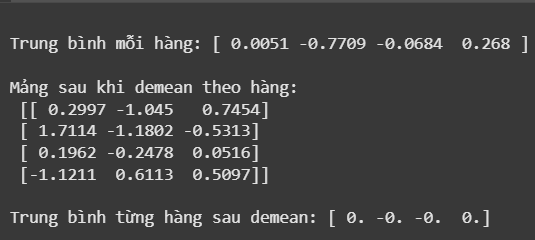
**Kết quả minh họa:**

Trung bình từng cột ≈ [0, 0, 0] sau khi demean.

***Bước 3 – Demean theo hàng:***

Tính arr.mean(1) rồi reshape (4,1) → arr - row\_means.reshape(4,1).

Nếu không reshape → lỗi shapes (4,3) (4,) do không tương thích.



***Bước 4 – Broadcasting mảng 3D:***

Thực hiện arr3d - arr3d.mean(2)[:, :, np.newaxis].

Broadcasting theo axis=2, tự mở rộng chiều còn thiếu để phù hợp (3,4,5).

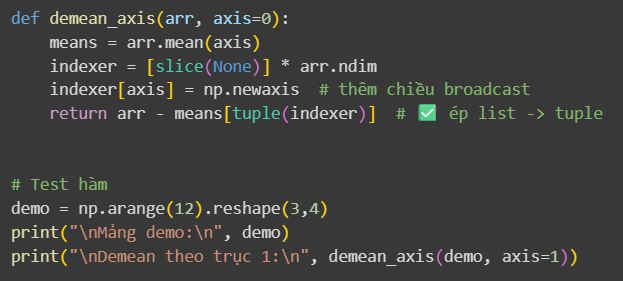


**Kết quả :**

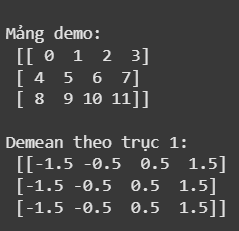


***Bước 5 – Tổng quát hóa:***

Tạo hàm demean\_axis(arr, axis) dùng np.newaxis tự động xác định chiều cần broadcast.



**Kết quả:**



**4.3.3. Nhận xét**

**Quy tắc chính:**

* Hai mảng tương thích để broadcast nếu:
* Kích thước trục cuối cùng bằng nhau, hoặc
* Một trong hai có kích thước 1.

**Lợi ích:**

* Không cần lặp từng phần tử.
* Không tạo mảng trung gian thừa.
* Cực nhanh nhờ vector hóa C.

**Ứng dụng thực tế:**

* Chuẩn hóa dữ liệu nhiều chiều.
* Demean, scale, transform trên ảnh hoặc tensor.

**4.3.4. Kết luận**

Broadcasting là “trái tim” của NumPy giúp:

* Thực hiện phép toán giữa mảng khác kích thước mà không cần vòng lặp Python.
* Tự động mở rộng chiều 1 hoặc thiếu để khớp shape.
* Kết hợp với np.newaxis hoặc reshape() giúp xử lý dữ liệu N chiều dễ dàng.
* Hiểu đúng quy tắc giúp bạn tối ưu hiệu năng và tránh lỗi shape mismatch.

**Tóm lại:**

“Broadcasting” = “tự động nhân bản chiều 1 để khớp phép toán”. Dễ dùng, mạnh mẽ, và là nền tảng của lập trình mảng hiệu quả trong NumPy.

## 4.4 Advanced ufunc Usage

### 4.4.1 Giới thiệu

Phần lớn người dùng NumPy chỉ dùng các phép toán nhanh theo từng phần tử (element-wise operations) mà ufunc (universal functions) cung cấp.

Tuy nhiên, NumPy còn có nhiều tính năng nâng cao giúp bạn viết code ngắn gọn và hiệu quả hơn mà không cần vòng lặp tường minh.

### 4.4.2. Các phương thức của đối tượng ufunc

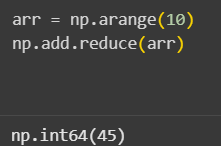
Mỗi ufunc nhị phân (binary ufunc, ví dụ như np.add, np.multiply) đều có các phương thức đặc biệt để thực hiện các phép toán vector hóa đặc thù.

Bảng A-2 dưới đây tóm tắt các phương thức chính, nhưng ta sẽ xem một vài ví dụ cụ thể để hiểu rõ cách hoạt động.

**a. Phương thức reduce**

reduce nhận một mảng đầu vào và tổng hợp giá trị của nó theo chuỗi các phép toán nhị phân (giống như cộng dồn, nhân dồn, v.v.), tùy chọn theo từng trục (axis).

Ví dụ: cộng toàn bộ phần tử trong mảng bằng cách dùng np.add.reduce:

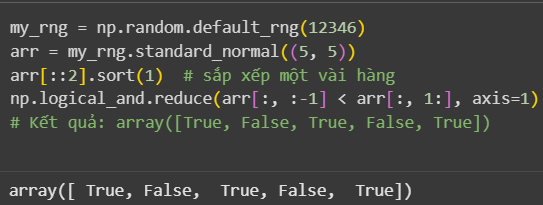


***Tương đương với:***

arr.sum()  # => 45

* Giá trị khởi đầu (ví dụ 0 với phép cộng) phụ thuộc vào từng ufunc.
* Nếu bạn truyền vào tham số axis, phép tổng hợp sẽ được thực hiện dọc theo trục đó.

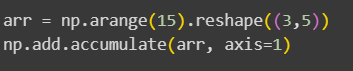
**Ví dụ**: Kiểm tra từng hàng có được sắp xếp hay không :

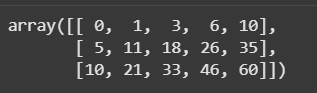


logical\_and.reduce tương đương với np.all(), tức là kiểm tra tất cả các giá trị trong hàng có đúng điều kiện hay không.

**b. Phương thức accumulate**

accumulate có vai trò tương tự reduce, nhưng thay vì trả về một giá trị duy nhất, nó lưu lại toàn bộ giá trị tích lũy trung gian (giống như cumsum so với sum).

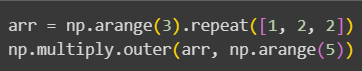


**Kết quả:** 

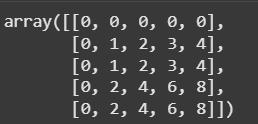
**c. Phương thức outer**

outer thực hiện phép toán chéo giữa từng cặp phần tử của hai mảng (pairwise operation).

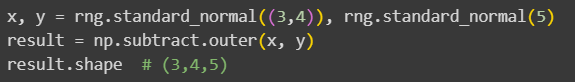
Kết quả là mảng có kích thước bằng tổng (concatenation) của hai shape đầu vào.



**Kết quả:**



**Hoặc dùng:**



**d. Phương thức reduceat**

reduceat thực hiện một phép "giảm cục bộ" (local reduce) — tương tự “group by” — tức là gom nhóm các lát cắt (slice) liên tiếp của mảng và thực hiện phép tổng hợp trên từng nhóm.



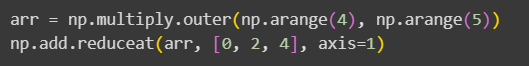
Kết qủa : 

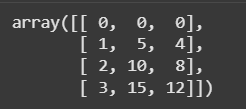
Giải thích: arr[0:5] → tổng = 10

arr[5:8] → tổng = 18

arr[8:] → tổng = 17

Có thể áp dụng theo trục cụ thể (axis):



Kết quả : 

*Tóm tắt các phương thức của ufunc:*

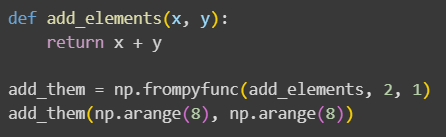
|  |  |
| --- | --- |
| Phương thức | Mô tả |
| accumulate(x) | Tích lũy các giá trị, giữ lại toàn bộ kết quả trung gian. |
| at(x, indices, b=None) | Thực hiện phép toán tại vị trí chỉ định (in-place) trong x. |
| reduce(x) | Tổng hợp các giá trị qua các phép nhị phân liên tiếp. |
| reduceat(x, bins) | Giảm cục bộ ("group by"): tổng hợp các lát dữ liệu liên tiếp. |
| outer(x, y) | Áp dụng phép toán cho mọi cặp phần tử giữa x và y. |

### 4.4.3. Viết ufunc mới trong Python

Có thể tự tạo ufunc tùy chỉnh trong Python mà không cần can thiệp C API.

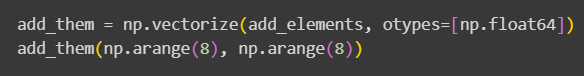
NumPy hỗ trợ điều này qua các hàm tiện ích như np.frompyfunc và np.vectorize

*Cách 1 – Tạo ufunc bằng frompyfunc:*



Kết quả : 

*Cách 2 – Dùng np.vectorize (có thể chọn kiểu dữ liệu output) :*



Kết quả : 

### 4.4.4. Kết luận

* Các phương thức reduce, accumulate, outer, reduceat giúp viết code ngắn gọn hơn, tránh vòng lặp, vẫn đảm bảo hiệu năng cao.
* np.frompyfunc và np.vectorize cho phép tạo ufunc tùy chỉnh trong Python, tuy nhiên tốc độ chậm hơn nhiều so với ufunc C gốc.
* Nếu cần hiệu năng cao, có thể kết hợp Numba (@vectorize, @njit) để tạo ufunc biên dịch nhanh hơn.

## 4.5 Structured and Record Arrays

### 4.5.1. Mục tiêu

Phần này nhằm giúp người học:

* Hiểu và sử dụng structured arrays (mảng có cấu trúc) trong NumPy.
* Phân biệt sự khác nhau giữa mảng đồng nhất (homogeneous ndarray) và mảng dị kiểu (heterogeneous / structured).
* Thực hành khai báo kiểu dữ liệu có nhiều trường, lồng nhau, hoặc đa chiều.
* Biết được ứng dụng thực tế của structured arrays trong việc đọc/ghi dữ liệu nhị phân, hoặc trao đổi dữ liệu với C/C++.

### 4.5.2. Cơ sở lý thuyết

Trong NumPy, một ndarray thông thường chứa các phần tử đồng nhất kiểu dữ liệu.  
Tuy nhiên, trong nhiều trường hợp, ta cần lưu trữ các “bản ghi” có nhiều trường với các kiểu khác nhau – ví dụ: một điểm dữ liệu có x (float), y (int), hoặc phức tạp hơn là nhiều giá trị lồng nhau. Khi đó, structured array là giải pháp.

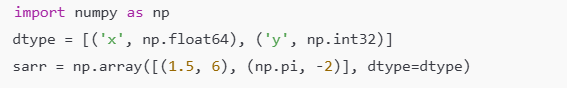
Mỗi phần tử trong mảng được định nghĩa bởi một dtype có nhiều trường (fields), mỗi trường có tên và kiểu dữ liệu riêng.

Cấu trúc này tương tự:

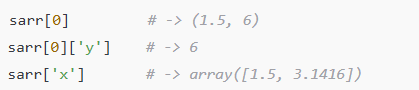
* Một struct trong ngôn ngữ C.
* Một hàng (row) trong bảng SQL, có nhiều cột với tên và kiểu khác nhau.

### 4.5.3. Quy trình và các đoạn code minh họa

**a. Structured dtype cơ bản**



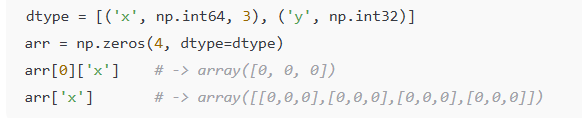
Truy cập:



**Nhận xét:**

* Mỗi bản ghi trong mảng có hai trường: x và y.
* Có thể truy cập bằng chỉ số hoặc tên trường.
* sarr['x'] trả về mảng của tất cả giá trị trường x.

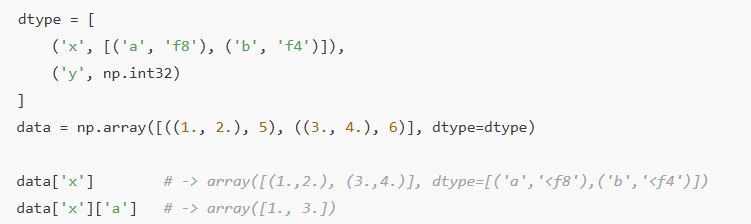
**b. Trường có nhiều phần tử (multidimensional field)**



**Nhận xét:**

* Có thể gán shape cho một trường, ví dụ ('x', np.int64, 3) nghĩa là mỗi bản ghi có mảng x gồm 3 phần tử.
* Khi truy cập arr['x'], ta nhận được mảng 2D (số bản ghi × 3).
* Phù hợp cho dữ liệu vector cố định như tọa độ 3D.

**c. Trường lồng nhau (nested dtype)**



**Nhận xét:**

* Trường x chứa **sub-struct** có hai trường con a và b.
* Có thể truy cập lồng nhau như data['x']['a'].
* Đây là mô hình biểu diễn dữ liệu dạng cây hoặc bản ghi phức tạp, cực kỳ hiệu quả trong biểu diễn dữ liệu nhị phân.

### 4.5.4. Ứng dụng – “Why Use Structured Arrays?”

Structured arrays không phổ biến bằng pandas.DataFrame, nhưng có lợi thế mạnh trong các trường hợp cần hiệu năng bộ nhớ và xử lý nhị phân:

* ***Hiệu suất cao & tiết kiệm bộ nhớ:***
* Các phần tử có kích thước cố định (fixed-size records).
* Dữ liệu lưu liên tục trong bộ nhớ → đọc/ghi nhanh, hỗ trợ memory mapping (mmap).
* ***Tương thích với C/C++:***
* Layout bộ nhớ giống struct trong C → có thể chia sẻ dữ liệu giữa NumPy và code C/C++ mà không cần chuyển đổi.
* ***Xử lý dữ liệu nhị phân (binary I/O):***
* Dễ dàng ghi/đọc file dạng byte stream với np.fromfile() và np.tofile().
* Thường dùng trong hệ thống nhúng, mô phỏng vật lý, hoặc xử lý tín hiệu.

**So sánh nhanh với pandas:**

| **Tiêu chí** | **Structured Array** | **pandas DataFrame** |
| --- | --- | --- |
| Cấp độ | Thấp (low-level) | Cao (high-level) |
| Kiểu dữ liệu | Cố định, định nghĩa trước | Linh hoạt, tự động |
| I/O nhị phân | Tối ưu, tương thích C | Hạn chế |
| Xử lý bảng dữ liệu | Giới hạn | Mạnh mẽ (groupby, join, merge...) |

### 4.5.5. Kết luận

Structured và Record Arrays là cầu nối giữa NumPy và ngôn ngữ C:

Chúng cho phép biểu diễn dữ liệu “giống bảng” trong cùng một block bộ nhớ, hỗ trợ truy cập nhanh, đọc ghi nhị phân và truyền dữ liệu giữa các ngôn ngữ lập trình khác nhau.

**Tóm tắt:**

* Cho phép lưu dữ liệu không đồng nhất trong NumPy array.
* Truy cập dễ dàng bằng tên trường.
* Có thể mở rộng thành nested hoặc multidimensional fields.
* Thích hợp cho các bài toán hiệu năng cao, I/O nhị phân, hoặc xử lý dữ liệu ở mức hệ thống.

## 4.6 More About Sorting

### 4.6.1. Mục tiêu

Phần này nhằm giúp người học:

* Hiểu rõ cách NumPy sắp xếp dữ liệu (sorting) trên mảng một chiều và nhiều chiều.
* Phân biệt sự khác nhau giữa sắp xếp tại chỗ (in-place) và tạo bản sao mới (out-of-place).
* Làm quen với các phương pháp sắp xếp gián tiếp (indirect sorts) như argsort() và lexsort().
* Hiểu thuật toán sắp xếp ổn định (stable), phân loại một phần (partial sorting), và tìm kiếm nhị phân trên mảng đã sắp (searchsorted()).
* Áp dụng sorting để giải quyết các bài toán phân nhóm, chọn phần tử lớn/nhỏ nhất, và xử lý dữ liệu.

### 4.6.2. Cơ sở lý thuyết

**a. ndarray.sort() – Sắp xếp tại chỗ**

* Giống như list.sort() trong Python, phương thức .sort() sắp xếp tại chỗ, không tạo mảng mới.
* Toàn bộ nội dung mảng được sắp xếp theo thứ tự tăng dần (ascending).
* Nếu mảng là “view” của một mảng khác, thay đổi này ảnh hưởng đến mảng gốc.

**b. np.sort() – Sắp xếp trả về bản sao**

* Hàm np.sort() tạo bản sao mới đã sắp xếp, không thay đổi mảng gốc.
* Có thể chỉ định trục (axis) để sắp xếp theo hàng (axis=1) hoặc theo cột (axis=0).

**c. Sắp xếp gián tiếp (Indirect Sort)**

* **argsort()**: trả về chỉ số (index) sắp xếp thay vì giá trị. Dùng khi muốn áp dụng cùng thứ tự sắp xếp cho mảng khác.
* **lexsort()**: cho phép sắp xếp theo nhiều khóa (multi-key sort), tương tự ORDER BY trong SQL.

**d. Thuật toán sắp xếp ổn định**

Một thuật toán “stable sort” đảm bảo các phần tử có giá trị bằng nhau giữ nguyên thứ tự ban đầu.

NumPy hỗ trợ 3 loại:

| **Loại** |  | **Độ phức tạp** | **Tốc độ** | |
| --- | --- | --- | --- | --- |
| quicksort |  | O(n²) | Nhanh nhất |
| mergesort |  | O(n log n) | Trung bình |
| heapsort |  | O(n log n) | Trung bình |

**e. Partial Sorting (Sắp xếp một phần)**

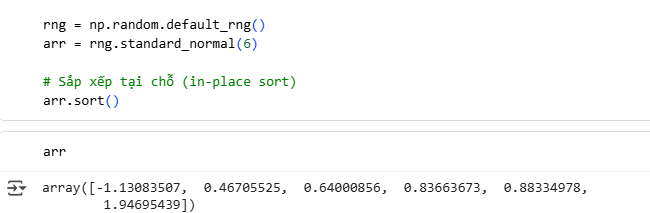
* np.partition() chỉ sắp xếp **một phần** của mảng — ví dụ tìm 3 giá trị nhỏ nhất mà không cần sắp cả mảng.
* np.argpartition() trả về chỉ số tương ứng. Rất hiệu quả cho các bài toán “top-k”.

**f. np.searchsorted() – Tìm vị trí chèn phần tử**

* Thực hiện **tìm kiếm nhị phân** trên mảng đã sắp.
* Trả về chỉ số (index) mà phần tử cần chèn vào để duy trì thứ tự.
* Thường kết hợp với pandas để **phân nhóm dữ liệu theo khoảng giá trị (binning)**.

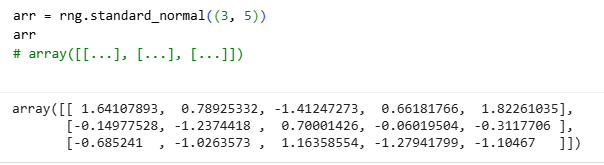
### 4.6.3. Quy trình thực hiện và ví dụ minh họa

**a. Sắp xếp cơ bản**



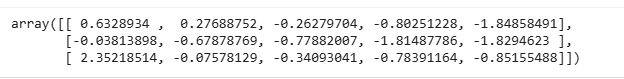
Mảng arr được sắp tăng dần tại chỗ.

**b. Sắp xếp theo hàng hoặc cột**

Mỗi hàng được sắp riêng biệt; axis=0 sẽ sắp theo cột.

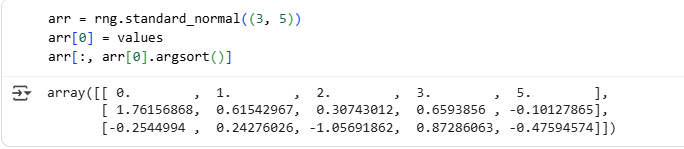
**c. Đảo ngược thứ tự**

arr[:, ::-1]

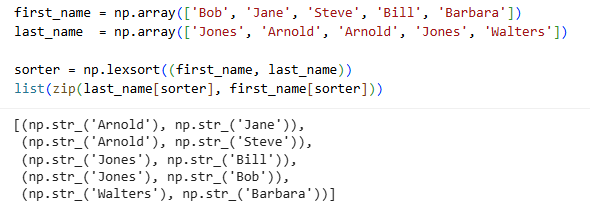


Đảo mảng theo thứ tự giảm dần mà không cần tính toán thêm.

**d. Sắp xếp gián tiếp**

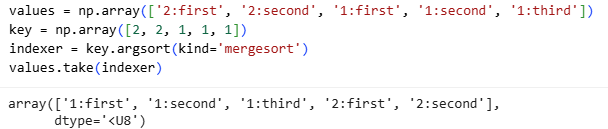
argsort() giúp sắp xếp dựa trên chỉ số — tiện khi cần sắp xếp nhiều mảng theo cùng thứ tự.

**e. Sắp xếp nhiều khóa với lexsort()**



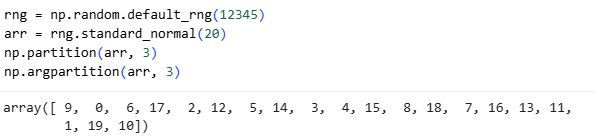
Kết quả sắp xếp theo last\_name trước, first\_name sau (giống ORDER BY last\_name, first\_name).

**f. Sắp xếp ổn định (mergesort)**



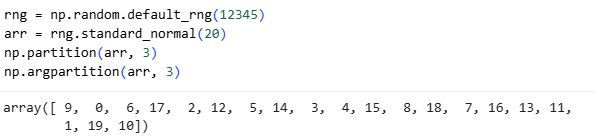
mergesort đảm bảo thứ tự tương đối của các phần tử trùng nhau không bị thay đổi.

**g. Partial sorting**



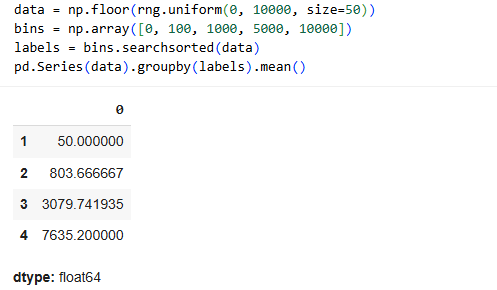
Tìm 3 giá trị nhỏ nhất nhanh chóng mà không sắp toàn mảng.

**h. Tìm kiếm trong mảng đã sắp (searchsorted)**



Xác định vị trí chèn phần tử để duy trì thứ tự tăng.

**i. Ứng dụng phân nhóm dữ liệu**



Kết hợp searchsorted với groupby để phân nhóm dữ liệu theo khoảng (binning).

### 4.6.4. Kết quả và nhận xét

| **Kỹ thuật** | **Kết quả** | **Nhận xét** |
| --- | --- | --- |
| .sort() | Mảng thay đổi trực tiếp | Nhanh, tiết kiệm bộ nhớ |
| np.sort() | Trả về bản sao mới | Không ảnh hưởng mảng gốc |
| argsort() | Trả về chỉ số sắp xếp | Dễ dùng cho sắp xếp gián tiếp |
| lexsort() | Multi-key sorting | Mạnh mẽ, dùng nhiều trong xử lý dữ liệu |
| mergesort | Ổn định, O(n log n) | Giữ thứ tự phần tử bằng nhau |
| partition() | Tìm top-k nhanh | Tối ưu cho bài toán tìm giá trị cực trị |
| searchsorted() | Xác định vị trí chèn | Ứng dụng tốt trong binning và tìm kiếm nhị phân |

**4.6.5. Ứng dụng thực tế**

* *Khoa học dữ liệu***:** chọn top-k giá trị (ví dụ top 5 sản phẩm bán chạy).
* *Phân loại thống kê***:** chia nhóm dữ liệu theo khoảng giá trị.
* *Tiền xử lý học máy***:** sắp xếp và lập chỉ mục dữ liệu lớn trước khi huấn luyện.
* *Xử lý mảng lớn***:** dùng các hàm partition() và searchsorted() để giảm chi phí tính toán.

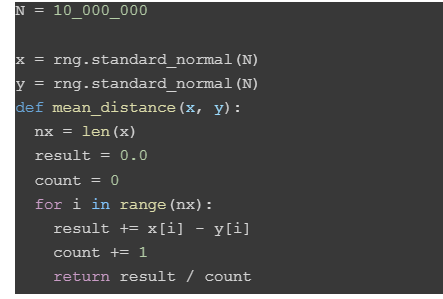
### 4.6.6. Kết luận

Phần A.6 – More About Sorting giúp hiểu sâu hơn về cơ chế sắp xếp dữ liệu trong NumPy, bao gồm cả sắp xếp trực tiếp, gián tiếp và tìm kiếm trên mảng đã sắp xếp.  
Các hàm như argsort, lexsort, partition và searchsorted là công cụ mạnh mẽ cho xử lý dữ liệu lớn, đặc biệt trong các bài toán cần hiệu năng và khả năng truy xuất nhanh.

## 4.7 Writing Fast NumPy Functions with Numba

### 4.7.1 Tính giá trị trung bình của hiệu hai mảng số thực có kích thước rất lớn

Bài toán mẫu:



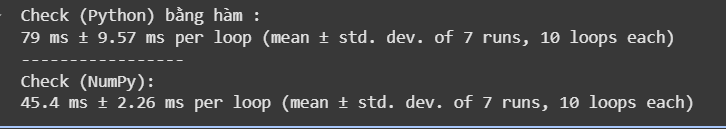
**Cách 1**: Hiệu của hai trung bình

mean(x−y)=mean(x)−mean(y)\mathrm{mean}(x-y)=\mathrm{mean}(x)-\mathrm{mean}(y)mean(x−y)=mean(x)−mean(y)

* Thực hiện: tính x.mean() và y.mean() rồi lấy hiệu.
* Đặc điểm: 2 lượt quét mảng, không tạo mảng (x-y)

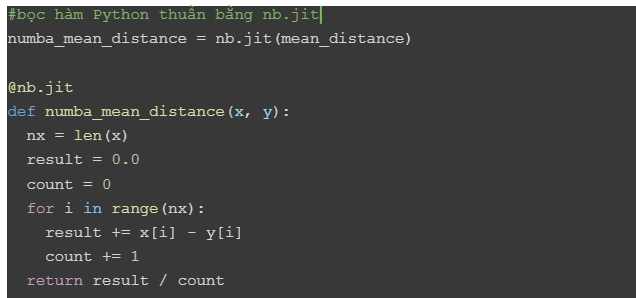
**Cách 2**: “Một bước” bằng bộ đệm (in-place)

* Tạo một buffer (có thể là bản sao của x): buf = x.copy().  
  Thực hiện trừ in-place: buf -= y rồi buf.mean().
* Đặc điểm: 1 lượt quét ghi/đọc, tránh cấp phát mảng (x-y) mới (không phá dữ liệu gốc nếu dùng buf).



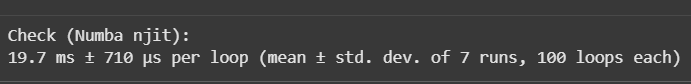
**Nhận xét:**

* NumPy nhanh hơn rõ rệt: (x - y).mean() ~ 45.4 ms so với hàm vòng lặp ~ 79 ms → NumPy ≈ 1.7× nhanh hơn và ổn định hơn (độ lệch chuẩn 2.26 ms vs 9.57 ms).
* Vì sao? Vòng lặp Python chịu overhead thông dịch từng phần tử; NumPy gọi mã C vector hóa/SIMD, tối ưu truy cập bộ nhớ.
* Khác số liệu “kinh điển” (nơi Python chậm ~giây): môi trường của bạn có thể bật tối ưu, I/O bộ nhớ/CPU khác, hoặc hàm Python đã được tối giản. Dù vậy, xu hướng NumPy > Python loop vẫn đúng.

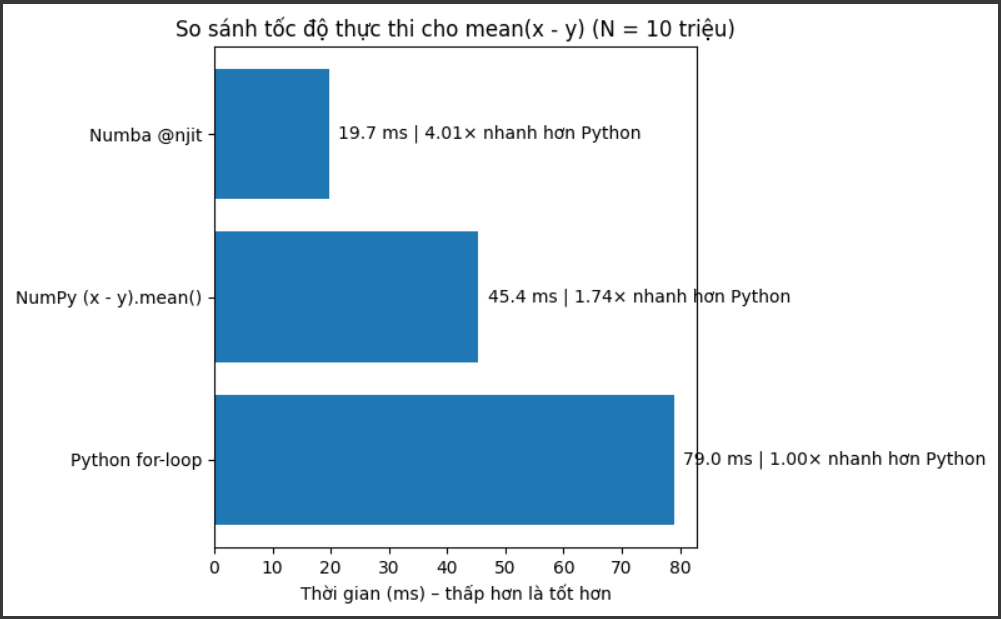


**Cách 3:** Numba JIT (nopython)

* @nb.jit / @njit để biên dịch JIT hàm Python sang machine code (LLVM).
* Có 2 biến thể:
* Vòng lặp thuần trong Numba (for i in range(nx): s += x[i] - y[i]) → chạy một pass, không tạo mảng tạm.
* Cú pháp “giống NumPy” (return (x - y).mean()) nhưng vẫn được Numba biên dịch và (thường) hợp nhất phép tính, nên vẫn rất nhanh.



Kết quả đo được: ≈19.7 ms/lượt (sau warm-up) — nhanh đáng kể.



**Nhận xét:**

* Python for-loop chậm vì vòng lặp cấp độ Python, overhead từng phần tử rất lớn.
* NumPy vector hóa nhanh hơn hàng chục–trăm lần nhờ thực thi bằng C nền tảng và tận dụng SIMD/bộ nhớ liên tục.
* Numba jit/njit có thể đánh bại NumPy trong nhiều ca vì:
  + JIT hóa vòng lặp, bỏ overhead Python.
  + Tận dụng tối ưu hóa LLVM (loop fusion, vectorization).
* Warm-up (lần gọi đầu) tốn thời gian compile; đo benchmark cần loại bỏ warm-up.

**Kết luận:**

NumPy là baseline nhanh nhờ vector hóa, nhưng Numba mang “vận tốc C” đến chính code Python của bạn. Khi bạn có thuật toán tự viết (vòng lặp, logic rẽ nhánh phức tạp, không tiện vector hóa), thêm @njit thường biến nó thành mã máy tối ưu.

Kết quả: bạn giữ nguyên cú pháp Python, mà vẫn đạt hiệu năng gần C/Fortran — đặc biệt hữu dụng khi:

* Vector hóa thuần NumPy khó/không diễn đạt được,
* Bạn cần tối ưu phần lõi tính toán (inner loops),
* Hoặc muốn tránh copy bộ nhớ do tạo mảng tạm khi vector hóa.

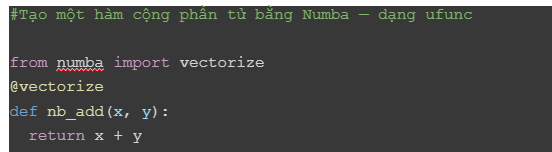
### 4.7.2 Creating Custom numpy.ufunc Objects with Numba

Tạo một hàm phần tử (elementwise) kiểu ufunc để cộng hai mảng và tận dụng các phép toán bậc cao của ufunc (như accumulate).

Cách tính: Dùng @numba.vectorize để biến hàm hai đối số x, y → x + y thành ufunc nb\_add. Ufunc hỗ trợ broadcasting và các phương thức như:

* nb\_add(x, x) → cộng từng phần tử (ở đây tương đương 2\*x).
* nb\_add.accumulate(x, axis=0) → tổng tích lũy (prefix sum) dọc theo axis → với x = [0,1,2,…,9] cho ra [0,1,3,6,10,15,21,28,36,45] (dãy tam giác).

Lần gọi đầu sẽ compile JIT; các lần sau dùng mã đã biên dịch.



**Nhận xét:**

* @numba.vectorize tạo ufunc biên dịch như ufunc gốc của NumPy: broadcasting, dtype dispatch, và các method như .accumulate/.reduce hoạt động y hệt.
* Ví dụ nb\_add(x, x) trả về mảng; nb\_add.accumulate(x, 0) cho tổng lũy tích—đúng hành vi của ufunc chuẩn.
* Ưu điểm: viết hàm scalar “x, y → x+y”, còn Numba lo song song hóa theo phần tử và biên dịch.
* Có thể chỉ định chữ ký kiểu và target (CPU, parallel=True, hoặc CUDA), giúp tối ưu hiệu năng đa nền tảng.

**Lưu ý:** ufunc từ vectorize phù hợp phép toán độc lập theo phần tử; nếu có phụ thuộc giữa phần tử, dùng @guvectorize/@njit với vòng lặp sẽ hợp lý hơn.

**Kết luận:**

@vectorize biến công thức scalar thuần Python thành một khối xây dựng hạ tầng NumPy: bạn giữ code rõ ràng như viết cho một phần tử, nhưng khi chạy, nó mở rộng trên cả mảng với tốc độ của mã biên dịch. Nghĩa là, thay vì cố “nhét” logic vào vector hóa NumPy (dễ tạo mảng tạm và khó đọc), bạn viết tự nhiên, rồi để Numba tạo ufunc hiệu năng cao có đầy đủ “đặc quyền” của ufunc (broadcast/accumulate/reduce). Kết quả: tính đúng kiểu ufunc, chạy nhanh kiểu C, và code vẫn gọn gàng, bảo trì dễ.

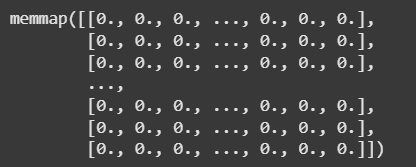
## 4.8 Advanced Array Input and Output

### 4.8.1 Memory-Mapped Files

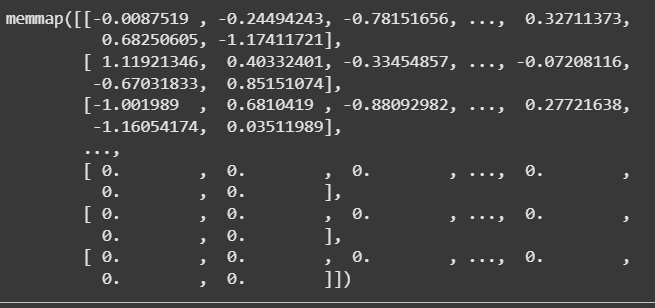
Làm việc với mảng rất lớn (hàng chục–trăm GB) vượt RAM. Cần:

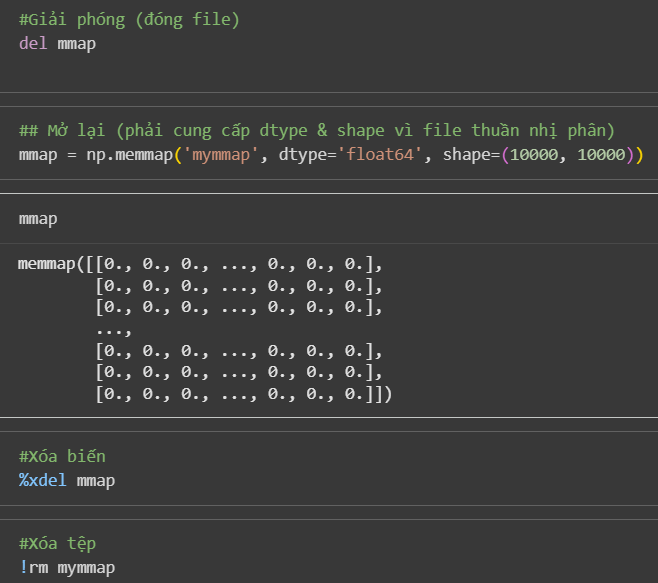
* Đọc/ghi một phần dữ liệu mà không nạp toàn bộ.
* Lưu trữ bền vững, nén/khả mở rộng tốt

Tạo mảng bộ nhớ ánh xạ:



Mảng sau khi ghi ngẫu nhiên 5 hàng đầu :





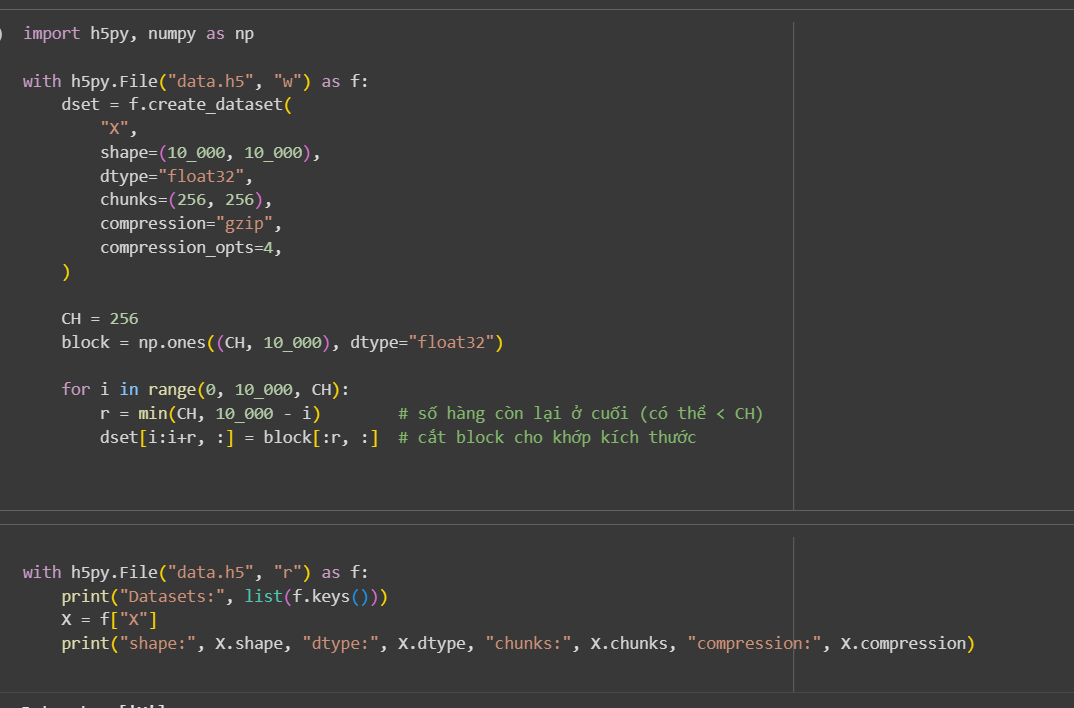
**Nhận xét:**

* np.memmap cho phép xử lý mảng cực lớn vượt RAM bằng cách ánh xạ file nhị phân trên đĩa thành “mảng như-ndarray”.
* Cắt/slice trả về view (không copy), chỉ đọc/ghi phần cần thiết, tiết kiệm I/O và bộ nhớ.
* Ghi vào slice được buffer trong bộ nhớ; dùng .flush() (hoặc khi object bị GC) để đồng bộ xuống đĩa.
* Mở lại file cũ phải chỉ rõ dtype và shape, vì file chỉ là bytes thô, không lưu metadata.
* Hoạt động với cả structured dtypes.
* Cẩn trọng: hiệu năng phụ thuộc pattern truy cập (ngẫu nhiên nhiều sẽ chậm), cần quản lý flush/đóng, và tránh ghi đồng thời không kiểm soát.

### 4.8.2 HDF5 and Other Array Storage Options

Bạn cần tạo và lưu một ma trận rất lớn (10.000 × 10.000, float32) xuống đĩa theo định dạng HDF5, sao cho:

* Không tải toàn bộ vào RAM khi ghi/đọc.
* Hỗ trợ nén và chunking để I/O hiệu quả, mở rộng được đến hàng trăm GB.
* Có thể xem metadata (tên dataset, shape, dtype, chunks, compression) và đọc từng lát nhỏ khi cần



**Nhận xét:**

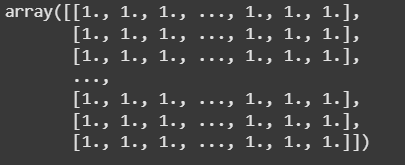
* File HDF5 có 1 dataset tên x (lưu ý: khác hoa/thường so với “X” lúc tạo).
* Kích thước (10 000, 10 000), dtype float32 ⇒ dữ liệu thô ~ 400 MB nếu không nén.
* Chunking (256×256) + gzip: tối ưu đọc/ghi theo lát nhỏ; với ma trận toàn 1 như bạn ghi, gzip sẽ nén rất mạnh (dung lượng thực tế nhỏ hơn nhiều).

Khi truy cập, nên cắt theo bội của 256 hàng/cột để khớp chunk → I/O nhanh hơn.

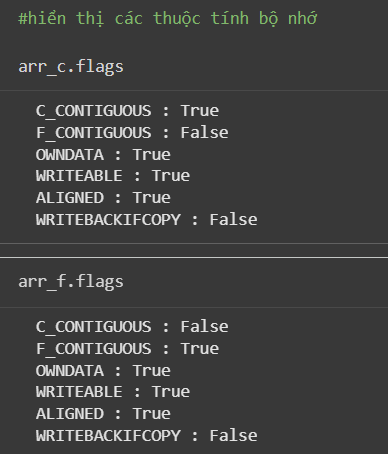
## 4.9 Performance Tips

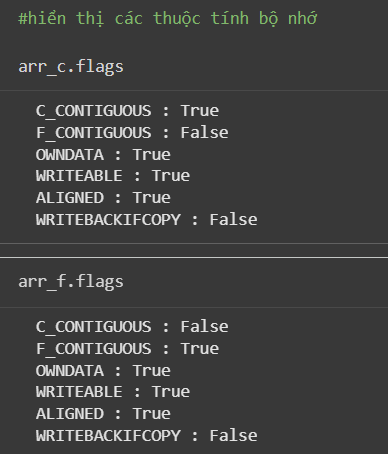
Tối ưu hiệu năng xử lý số liệu với NumPy: thay vòng lặp Python bằng phép toán mảng, tận dụng broadcasting/ufunc, và đặc biệt bố cục bộ nhớ liên tục (contiguous memory) để giảm chi phí truy cập bộ nhớ và tận dụng cache CPU.

Tạo mảng theo C-order (hàng liên tục) và F-order (cột liên tục) trong RAM cùng kích thước và phần tử:



Hiển thị các thuộc tính bộ nhớ của mảng:

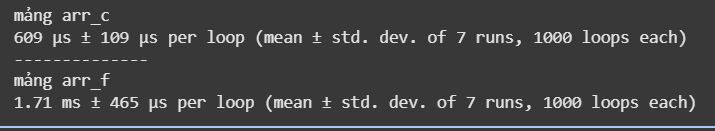


****

**Nhận xét:**

* C\_CONTIGUOUS (C-order): True nếu mảng liên tục theo hàng (row-major). Truy cập theo hàng nhanh hơn cho mảng C-order.
* F\_CONTIGUOUS (Fortran-order): True nếu mảng liên tục theo cột (column-major). Truy cập theo cột nhanh hơn cho mảng F-order.
* OWNDATA: True nếu mảng sở hữu bộ nhớ của nó. False nghĩa là chỉ là view trên bộ nhớ của mảng khác.
* WRITEABLE: True nếu mảng ghi được; False là chỉ đọc (read-only).
* ALIGNED: True nếu dữ liệu được căn lề đúng theo yêu cầu của kiểu dữ liệu (hữu ích cho hiệu năng/ SIMD).
* WRITEBACKIFCOPY: dùng cho cơ chế “copy-on-write”: nếu True, bản sao tạm sẽ được ghi trả lại bộ nhớ gốc khi mảng bị xoá. Thường thấy False trong các thao tác thông thường.

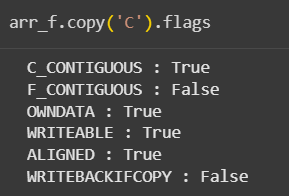
Thử nghiệm kiểm tra tốc độ với phép tổng theo hàng (axis = 1)



**Nhận xét:**

* Tổng theo hàng tận dụng truy cập liên tiếp trong arr\_c → nhanh hơn.
* Với arr\_f, tổng theo hàng là nhảy bước trong bộ nhớ (strided) → chậm hơn.

**Ép lại thứ tự bộ nhớ khi cần:**



**Nhận xét:**

* Bố cục bộ nhớ liên tục (contiguous) ảnh hưởng rõ rệt đến tốc độ vì CPU cache (L1/L2) hoạt động hiệu quả khi dữ liệu nằm liền kề.
* Mặc định NumPy là C-order (row-major); phép toán theo chiều hàng trên mảng C-contiguous thường nhanh hơn so với cùng phép trên mảng F-order (ví dụ arr\_c.sum(1) < arr\_f.sum(1)).
* Thuộc tính arr.flags cho biết mảng có C\_CONTIGUOUS/F\_CONTIGUOUS không; arr.T hoặc một số view/slice có thể mất tính liên tục.
* Nếu cần, dùng arr.copy('C') hoặc arr.copy('F') để ép bố cục mong muốn; đổi lại tốn bộ nhớ và thời gian copy.
* Nhiều đường code C trong NumPy được tối ưu riêng cho trường hợp contiguous — tránh truy cập strided tổng quát chậm hơn.

**Kết luận:**

Khi “vắt kiệt” hiệu năng NumPy, đừng chỉ nghĩ đến vector hóa — hãy chăm sóc bố cục bộ nhớ. Sắp xếp dữ liệu theo trục truy cập chính (C hay F) giúp phép tính tuyến tính (như reduce theo trục đó) ăn cache tốt hơn và bớt tạo stride phức tạp. Trước khi tối ưu thuật toán, hãy kiểm tra arr.flags (và các view/slice) rồi chọn đúng order hoặc copy có chủ đích: một thao tác sắp xếp bộ nhớ đúng chỗ có thể mang lại cải thiện hàng chục phần trăm thời gian chạy, đặc biệt trên mảng lớn.

# V. Tổng kết và bài học

Chapter 13: Data Analysis Examples là một cẩm nang thực tiễn, trình bày toàn bộ quy trình phân tích dữ liệu qua các ví dụ đa dạng như dữ liệu Bitly, MovieLens, và dữ liệu bầu cử FEC. Quá trình này bao gồm các bước cốt lõi từ việc đọc và làm sạch dữ liệu, biến đổi cấu trúc bằng các kỹ thuật mạnh mẽ như hợp nhất (merge) và xoay vòng (pivot\_table), cho đến việc khai thác thông tin chi tiết qua phân tích nhóm (groupby) và cuối cùng là kể lại câu chuyện dữ liệu một cách trực quan qua các biểu đồ. Từ đó, tài liệu rút ra những bài học kinh nghiệm vô giá: làm sạch dữ liệu là nền tảng, cấu trúc dữ liệu đúng đắn quyết định khả năng phân tích, và trực quan hóa là chìa khóa để truyền tải insight một cách hiệu quả. Tựu trung lại, đây là một hướng dẫn toàn diện không chỉ về công cụ, mà còn về tư duy phân tích để biến dữ liệu thô thành những quyết định mang giá trị.