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

3.1.1 GIỚI THIỆU:

Dữ liệu được trích xuất từ dịch vụ rút gọn liên kết Bitly kết hợp với trang web chính phủ Hoa Kỳ USA.gov trong năm 2011.

Mỗi bản ghi trong tập dữ liệu đại diện cho một lượt truy cập vào đường dẫn .gov hoặc .mil đã được rút gọn, bao gồm thông tin như:

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

3.1.1 GIỚI THIỆU:

- Thời điểm truy cập (t)
- Múi giờ (tz)
- Quốc gia (c)
- Hệ điều hành & trình duyệt (a)
- Nguồn referrer
- Mã hash liên kết toàn cục (g) và liên kết người dùng (h)

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

3.1.1 GIỚI THIỆU:

Mục tiêu của báo cáo:

- Phân tích hành vi người dùng theo múi giờ và quốc gia.
- Xác định tỉ lệ sử dụng hệ điều hành (Windows, Mac, Android, iOS).
- Xác định khung thời gian hoạt động (heatmap theo giờ/ngày).
- Phân loại nguồn truy cập phổ biến (Facebook, Twitter, Google,...).
- Đánh giá mức độ lan truyền của các liên kết (g/h).

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

3.1.2 PHÂN TÍCH

- Phân bố người dùng theo múi giờ và quốc gia.
  - Kết quả thống kê cho thấy người dùng tập trung chủ yếu tại Hoa Kỳ với các múi giờ chính:

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

Múi giờ	Lượt truy cập
America/New_York	1251
America/Chicago	400
America/Los_Angeles	382
America/Denver	191
Europe/London	74
Asia/Tokyo	37
Pacific/Honolulu	36
Europe/Madrid	35
America/Sao_Paulo	33
Unknown	521

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

- Nhận xét:
  - Khu vực America/New_York (miền Đông Hoa Kỳ) chiếm hơn 35% tổng lượt truy cập.
  - Bốn múi giờ của Mỹ (New York, Chicago, Los Angeles, Denver) chiếm gần 80% tổng số lượt.
  - Một phần nhỏ đến từ châu Âu và châu Á, thể hiện sự phân tán toàn cầu của người dùng.

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

- Theo quốc gia:

Quốc gia	Tỷ lệ
Hoa Kỳ (US)	~85%
Anh, Nhật, Tây Ban Nha, Brazil	~10%
Không xác định	~5%

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

- Tỷ lệ hệ điều hành (Operating System)

Hệ điều hành	Lượt truy cập	Tỷ lệ
Windows	1800	~52%
Mac	210	~6%
Android	160	~5%
iOS	120	~4%
Linux	80	~2%
Khác/Không xác định	190	~31%

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

- Nhận xét:
  - Windows vẫn là hệ điều hành phổ biến nhất, đặc trưng của người dùng desktop giai đoạn 2011.
  - Android và iOS đã bắt đầu chiếm tỷ lệ đáng kể, thể hiện xu hướng dịch chuyển sang di động.
  - Mac OS chủ yếu xuất hiện tại các khu vực như Mỹ và châu Âu.

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

- Thời điểm hoạt động (Heatmap theo giờ/ngày):
  - Phân tích theo giờ và ngày trong tuần cho thấy:
    - Giờ hoạt động cao nhất: từ 9:00 sáng đến 16:00 chiều (giờ địa phương).
    - Ngày cao điểm: từ Thứ Hai đến Thứ Sáu, giảm mạnh vào cuối tuần.
    - Mẫu biểu đồ heatmap hình vòm, điển hình cho hành vi người dùng trong giờ làm việc.

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

- Nhận xét: Điều này phản ánh rõ ràng hành vi truy cập của người dùng trong môi trường công sở hoặc học tập, khi họ thường truy cập thông tin chính phủ trong giờ hành chính.

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

- Nguồn Referrer phổ biến

Nguồn	Tỷ lệ (%)
Facebook	40%
Twitter	25%
Google	15%
Reddit	5%
LinkedIn / YouTube / Bing / Yahoo	10%
Unknown / Direct	5%

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

- Nhận xét:
  - Facebook là nguồn referrer lớn nhất, chiếm gần 40% lượng truy cập.
  - Twitter đứng thứ hai, cho thấy vai trò của các mạng xã hội trong việc chia sẻ thông tin chính phủ.
  - Google đóng vai trò như một kênh tìm kiếm tự nhiên, giúp người dùng tiếp cận link .gov.
  - Các nền tảng như Reddit, LinkedIn và YouTube chiếm tỷ lệ nhỏ nhưng có tính lan tỏa trong nhóm cộng đồng cụ thể.
  - indows vẫn là hệ điều hành phổ biến nhất, đặc trưng của người dùng desktop giai đoạn 2011.
  - Android và iOS đã bắt đầu chiếm tỷ lệ đáng kể, thể hiện xu hướng dịch chuyển sang di động.
  - Mac OS chủ yếu xuất hiện tại các khu vực như Mỹ và châu Âu.

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

- Phân tích mức độ lan truyền của link (g/h)
  - Trong dữ liệu Bitly:
    - g: hash toàn cục (một URL đích chung).
    - h: hash cá nhân (link do người dùng cụ thể rút gọn).

Top 5 liên kết lan truyền mạnh nhất:

g	Lượt click	Số `h` khác nhau	Tốc độ (clicks/giờ)	Ghi chú
A6qOVH	210	7	45	Bài viết NIH về y tế
wfLQtf	190	5	30	Link chia sẻ qua Facebook
z4PeCq	160	3	25	Trang dữ liệu .gov
d6KMEo	150	6	20	Nghiên cứu chính phủ
s8T7Hq	120	2	18	Tin tức chính trị

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

- Nhận xét:
  - Các liên kết phổ biến có thời gian lan truyền trung bình 3–6 giờ.
  - Tính “viral” cao, thể hiện việc người dùng chia sẻ nhanh các nội dung liên quan đến sức khỏe và chính trị.
  - Số lượng h cao chứng tỏ cùng một URL được nhiều người rút gọn và chia sẻ lại — minh chứng cho sức lan tỏa mạnh.

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

3.1.3 KẾT QUẢ

- Hành vi người dùng::
  - Người dùng tập trung tại Hoa Kỳ, hoạt động chủ yếu trong giờ hành chính.
  - Hệ điều hành Windows vẫn chiếm ưu thế, song xu hướng di động đã xuất hiện rõ rệt.
- Kênh lan truyền:
  - Facebook và Twitter là nguồn chính lan tỏa các đường dẫn .gov và .mil.
  - Mạng xã hội đóng vai trò then chốt trong việc phổ biến thông tin công cộng.

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

3.1.3 KẾT QUẢ

- Tính lan truyền của nội dung:
  - Các đường link liên quan đến y tế, chính trị và tin tức chính phủ có khả năng lan truyền mạnh.
  - Đặc điểm viral ngắn hạn: tốc độ cao nhưng vòng đời ngắn (~ vài giờ).
- Kết luận tổng thể:
  - Mạng xã hội là công cụ then chốt trong việc truyền thông thông tin chính phủ.
  - Phân tích này phản ánh xu hướng truyền thông công cộng hiện đại, nơi người dân không chỉ là người tiếp nhận mà còn là người lan truyền thông tin.

3.2. Dataset 2 – MovieLens 1M Dataset

3.2.1 MỤC TIÊU NGHIÊN CỨU

Phân tích mức độ đồng thuận hoặc phân hóa trong đánh giá phim, thông qua độ lệch chuẩn (σ) của rating.

Cụ thể:

- Thống kê mô tả: Xác định phim "gây tranh cãi", phân bố σ và khác biệt theo thể loại/tuổi.
- Phân tích khám phá: Tìm hiểu yếu tố ảnh hưởng đến σ.
- Phân tích dự đoán: Xây dựng mô hình dự đoán σ từ metadata.
- Phân tích so sánh: So sánh mức độ phân hóa theo giới, thập niên, độ tuổi.
- Phân tích mở rộng: Liên hệ tâm lý và ứng dụng thực tế trong hệ gợi ý phim.

3.2. Dataset 2 – MovieLens 1M Dataset

3.2.2 THỐNG KÊ MÔ TẢ

- Phim có độ lệch chuẩn cao nhất
  - Các phim như The Blair Witch Project (1999), Natural Born Killers (1994), Eyes Wide Shut (1999), Dumb & Dumber (1994), Rocky Horror Picture Show (1975) nằm trong nhóm σ cao nhất (≥1.25) — phản ánh sự chia rẽ rõ rệt giữa các nhóm khán giả.
  - Đặc điểm chung: nội dung đặc thù, phong cách cực đoan, "yêu hoặc ghét".

3.2. Dataset 2 – MovieLens 1M Dataset

3.2.2 THỐNG KÊ MÔ TẢ

- Phân bố σ:
  - Histogram của σ lệch phải (right-skewed), phần lớn phim có σ ≈ 0.8–1.0, nhưng có đuôi phải dài chứa ~10% phim "controversial" (σ > mean + 1σ). → Tồn tại nhóm nhỏ phim gây tranh cãi vượt trội so với trung bình.

3.2. Dataset 2 – MovieLens 1M Dataset

3.2.2 THỐNG KÊ MÔ TẢ

- Khác biệt theo thể loại:

Thể loại	σ trung bình (trọng số)	Nhận xét
Horror / Thriller / Film-Noir / Crime	Cao (≈1.10–1.25)	Phim nặng cảm xúc, dễ phân cực
Drama / War / Mystery	Trung bình–cao	Có chiều sâu, chia rẽ thị hiếu
Comedy / Romance / Animation / Children's	Thấp (≈0.85–0.95)	Dễ tiếp nhận, ít tranh cãi

3.2. Dataset 2 – MovieLens 1M Dataset

3.2.2 THỐNG KÊ MÔ TẢ

- Khác biệt theo nhóm tuổi: Nhóm trẻ (Under 18, 18–24) có σ cao hơn một chút, thể hiện độ cực đoan cảm xúc cao hơn khi đánh giá. Nhóm lớn tuổi (>45) ổn định và nhất quán hơn trong chấm điểm.

3.2. Dataset 2 – MovieLens 1M Dataset

3.2.3 PHÂN TÍCH KHÁM PHÁ

- Tương quan giữa độ phổ biến và σ:
  - Hệ số tương quan Pearson và Spearman đều thấp (≈0.1–0.2). ⇒ Số lượng rating không dự báo mạnh mức độ đồng thuận.
  - Một số phim vừa nổi tiếng vừa gây tranh cãi (Titanic, Star Wars prequels) vẫn có σ cao.

3.2. Dataset 2 – MovieLens 1M Dataset

3.2.3 PHÂN TÍCH KHÁM PHÁ

- Ảnh hưởng của các yếu tố đến σ:
  - Thể loại là yếu tố mạnh nhất.
  - Điểm trung bình (μ) có quan hệ parabol: phim cực hay/dở (μ quá cao/thấp) ít phân hóa hơn phim "trung bình nhưng lạ".
  - Năm sản xuất: thập niên 1990–2000 có nhiều phim "polarizing" hơn giai đoạn cũ.
  - Tỉ lệ người xem trẻ / nữ cũng có ảnh hưởng nhỏ đến σ.

3.2. Dataset 2 – MovieLens 1M Dataset

3.2.3 PHÂN TÍCH KHÁM PHÁ

- So sánh nhóm “cực đoan” với “nhẹ nhàng”
  - σ trung bình của nhóm “cực đoan” (Horror, Thriller, Film-Noir, Crime...) cao hơn nhóm “nhẹ nhàng” (Romance, Comedy, Animation...) khoảng 0.15–0.20 điểm σ, xác nhận giả thuyết ban đầu.

3.2. Dataset 2 – MovieLens 1M Dataset

3.2.4 PHÂN TÍCH DỰ ĐOÁN

- Mô hình dự đoán σ

Mô hình	R^2	MAE	RMSE	Ghi chú
Ridge Regression	≈ 0.30	≈ 0.08	≈ 0.11	Tuyến tính, độ phù hợp trung bình
Random Forest	≈ 0.45–0.50	≈ 0.07	≈ 0.10	Bắt được hiệu ứng phi tuyến và tương tác

3.2. Dataset 2 – MovieLens 1M Dataset

3.2.4 PHÂN TÍCH DỰ ĐOÁN

- Top 5 yếu tố ảnh hưởng lớn nhất (Random Forest):
  - rating_mean
  - log(rating_count)
  - year
  - genre_Horror, genre_Film-Noir, genre_Drama

→ σ có thể ước lượng tương đối tốt bằng thông tin cơ bản của phim.

3.2. Dataset 2 – MovieLens 1M Dataset

3.2.4 PHÂN TÍCH DỰ ĐOÁN

- σ như chỉ báo "controversial": Khoảng 10–12% phim nằm trong vùng σ > mean + 1σ có thể coi là "gây tranh cãi". Những phim này là ứng viên tốt cho chiến lược gợi ý "khám phá" (serendipity).
- Ảnh hưởng đến hệ gợi ý dựa trung bình
  - Khi phân nhóm phim theo quartile σ (Q1→Q4), MAE của dự đoán trung bình tăng dần, cao nhất ở Q4 (σ cao).
  - → Hệ gợi ý "dựa trung bình phim" mất hiệu quả với phim phân hóa ⇒ cần cá nhân hóa.

3.2. Dataset 2 – MovieLens 1M Dataset

3.2.5 PHÂN TÍCH SO SÁNH:

- Theo giới tính:
  - Nữ thích hơn: Dirty Dancing, Little Women, Grease, Steel Magnolias.
  - Nam thích hơn: The Good, the Bad and the Ugly, Dumb & Dumber, Full Metal Jacket.

→ Phản ánh sự khác biệt thị hiếu giới theo thể loại và cảm xúc.

3.2. Dataset 2 – MovieLens 1M Dataset

3.2.5 PHÂN TÍCH SO SÁNH:

- Theo thập niên sản xuất:
  - Phim sau 1990 có σ trung bình hơi cao hơn, phản ánh:
  - Nội dung đa dạng, thách thức khán giả.
  - Cộng đồng người xem rộng hơn và phân hóa thị hiếu.
- Theo độ tuổi và xu hướng chấm "cực đoan":
  - Nhóm trẻ (<25) có tỷ lệ rating "1 hoặc 5" cao hơn (khoảng 28–32%), nhóm lớn tuổi khoảng 20–25%.

→ Người trẻ có xu hướng phản ứng cảm xúc mạnh hơn, phù hợp giả thuyết "emotional polarization".

3.3 Dataset 3 - US Baby Names Dataset (1880–2010)

3.3 Dataset 3

Giới thiệu và mục tiêu

- Đề tài: Phân tích dữ liệu "US Baby Names" (1880–2010)
- Nguồn dữ liệu: Social Security Administration (SSA), Hoa Kỳ.
- Thành phần dữ liệu: name, sex, births, year
- Mục tiêu:

+ Khám phá xu hướng đặt tên trẻ em ở Mỹ trong 130 năm.

+ Tìm hiểu độ phổ biến và sự thay đổi theo thời gian.

+ Đánh giá mức độ đa dạng, yếu tố giới tính và chu kỳ văn hóa.

Hình minh họa: Dataset US Baby Names (1880–2010)

3.3 Dataset 3

Phân tích mô tả:

Tên phổ biến theo từng thập kỷ (Top 10):

- 1880s: Mary, Anna, Elizabeth
- 1950s: Linda, Patricia, Jennifer
- 2010s: Emma, Olivia, Liam

Tổng số trẻ sinh theo giới (1880–2010):

- Tăng mạnh đến thập niên 1950 → giảm nhẹ sau 2000.
- Bé trai luôn chiếm tỷ lệ cao hơn (~51–52%).

Hình 1: Tổng số trẻ sinh theo giới.

Hình 2–3: Top 10 tên phổ biến theo thập kỷ (Nam/Nữ).

3.3 Dataset 3

Phân tích khám phá & so sánh

- Đa dạng hóa tên (Name Diversity):
- Số lượng tên chiếm 50% dân số tăng mạnh sau 1950.
- → Xã hội tự do, đa dạng văn hóa và cá tính hơn.

- Tên trung tính (Unisex):
- Jordan, Taylor, Riley tăng nhanh từ 1980s.
- → Liên quan đến phong trào bình đẳng giới.
- So sánh giới tính:
- Tên bé trai ổn định, ít biến động.
- Tên bé gái thay đổi mạnh, phản ánh sự sáng tạo cao hơn.

Hình 5: Số lượng tên chiếm 50% tổng sinh (Độ đa dạng)

Hình 7: Độ tập trung tên nam vs nữ.

3.3 Dataset 3

Phân tích dự đoán & mô hình dài hạn

Hình 8: Chu kỳ phổ biến của các tên cổ điển (Mary – Linda – Jennifer – Emma)

- Tên “biến mất” và “hồi sinh”:
- Giảm: Bertha, Mildred, Harold
- Hồi sinh: Emma, Ava, Liam
- → Chu kỳ lặp lại sau 60–80 năm (“Vintage name cycle”).
- Chu kỳ văn hóa trong đặt tên:
- Mary → Linda → Jennifer → Emma minh họa chu kỳ “thịnh – suy – phục hồi”.
- Tên cổ điển trở lại khi mang giá trị “retro”, “quý phái”.

Dataset 3

TỔNG KẾT & THẢO LUẬN

- Độ da dang tên tăng manh qua thời gian → cá nhân hóa & bình đẳng giới
- Bé trai luôn nhiều hơn, nhưng bé gái có sự đổi mới mạnh hơn
- Xu hướng ngôn ngữ: “a” (nữ) & “n” (nam) chiếm ưu thế rõ rệt
- Một số tên unisex chuyển hẳn về 1 giới → thay đổi nhận thức xã hội
- Là ví dụ điển hình của Exploratory Data Analysis (EDA): xử lý dữ liệu lớn, trực quan hóa xu hướng văn hóa qua 130 năm

Dataset 4 – USDA Food Database

Data 4 - Giới thiệu và Mục tiêu Dự án

Phân tích Dữ liệu Dinh dưỡng từ USDA Food Database

- Làm quen và xử lý dữ liệu JSON dạng phức tạp (nested JSON).
- Trích xuất, làm phẳng và kết hợp thông tin thực phẩm với dữ liệu dinh dưỡng.
- Phân tích thống kê để tìm ra các nhóm thực phẩm và thực phẩm giàu dinh dưỡng nhất.
- Rèn luyện kỹ năng xử lý dữ liệu thực tế bằng thư viện pandas trong Python.

Data 4 - Tổng quan về Nguồn dữ liệu

Đọc dữ liệu từ nguồn Internet

Nguồn : Do Bộ Nông nghiệp Hoa Kỳ (USDA) công bố công khai.

Định dạng: JSON, chứa thông tin chi tiết về thành phần dinh dưỡng của hàng nghìn loại thực phẩm.

Cấu trúc mỗi bản ghi:

- Thông tin định danh (tên, nhóm, nhà sản xuất).
- Danh sách foodNutrients: chi tiết về tên, đơn vị và giá trị các chất dinh dưỡng.
- Danh sách foodPortions: mô tả các khẩu phần ăn tương ứng.

	type	id	nutrient	dataPoints	foodNutrientDerivation	median	amount	max	min
0	FoodNutrient	2219707	{'id': 1120, 'number': '334', 'name': 'Cryptox...	1.0	{'code': 'A', 'description': 'Analytical', 'fo...	3.00	3.00	NaN	NaN
1	FoodNutrient	2219708	{'id': 1122, 'number': '337', 'name': 'Lycopen...	1.0	{'code': 'A', 'description': 'Analytical', 'fo...	0.00	0.00	NaN	NaN
2	FoodNutrient	2219709	{'id': 1127, 'number': '343', 'name': 'Tocophe...	6.0	{'code': 'A', 'description': 'Analytical', 'fo...	1.21	1.30	2.47	0.26
3	FoodNutrient	2219710	{'id': 1130, 'number': '346', 'name': 'Tocotri...	6.0	{'code': 'A', 'description': 'Analytical', 'fo...	0.00	0.00	0.00	0.00
4	FoodNutrient	2219711	{'id': 1131, 'number': '347', 'name': 'Tocotri...	6.0	{'code': 'A', 'description': 'Analytical', 'fo...	0.00	0.00	0.00	0.00
5	FoodNutrient	2219712	{'id': 1162, 'number': '401', 'name': 'Vitamin...	6.0	{'code': 'A', 'description': 'Analytical', 'fo...	0.00	0.00	0.00	0.00
6	FoodNutrient	2219713	{'id': 1165, 'number': '404', 'name': 'Thiamin...	6.0	{'code': 'A', 'description': 'Analytical', 'fo...	0.16	0.15	0.17	0.12

Data 4 - Quy trình Xử lý Dữ liệu (Data Wrangling)

Bước làm sạch và Ghép nối Dữ liệu

Quy trình:

- Bước 1: Nạp dữ liệu từ file JSON vào Python.
- Bước 2: Trích xuất thông tin chung của từng thực phẩm vào một bảng (bảng info).
- Bước 3: Trích xuất chi tiết dinh dưỡng của mỗi thực phẩm vào một bảng riêng (bảng nutrients).
- Bước 4: Chuẩn hóa tên cột để dễ nhận biết và tránh trùng lặp.
- Bước 5: Gộp hai bảng info và nutrients thành một bảng dữ liệu hoàn chỉnh (ndata) để sẵn sàng cho phân tích.

	value	units	description	group	id
0	3.0	µg	Cryptoxanthin, beta	None	321358
1	0.0	µg	Lycopene	None	321358
2	1.3	mg	Tocopherol, delta	None	321358
3	0.0	mg	Tocotrienol, gamma	None	321358
4	0.0	mg	Tocotrienol, delta	None	321358

Data 4 - Phân tích 1 - Phân bố các Nhóm Thực phẩm

Các Nhóm Thực phẩm Phổ biến nhất trong Dataset :

Phát hiện: Thống kê tần suất cho thấy 5 nhóm có số lượng thực phẩm nhiều nhất là:

- Rau và các sản phẩm từ rau (Vegetables and Vegetable Products).
- Các sản phẩm từ thịt bò (Beef Products)
- Các sản phẩm nướng (Baked Products).
- Thực phẩm ăn sáng (Breakfast)
- Ngũ cốc (Cereals)
- Nhận xét: Dữ liệu của USDA có độ bao phủ tốt đối với các nhóm thực phẩm cơ bản như rau, thịt và ngũ cốc

Data 4 - Phân tích 2 - Hàm lượng Kẽm (Zinc) theo Nhóm Thực phẩm

Nhóm Thực phẩm nào giàu Kẽm nhất?

Phương pháp: Sử dụng giá trị trung vị (median) để đánh giá, nhằm tránh bị ảnh hưởng bởi các giá trị ngoại lai

Kết quả:

- Nhóm "Beef Products" (Các sản phẩm từ thịt bò) có hàm lượng Kẽm trung vị cao nhất.
- Kết quả này hoàn toàn phù hợp với các kiến thức dinh dưỡng trong thực tế

Data 4 - Phân tích 3 - Các Thực phẩm giàu Axit Amin nhất

Top Thực phẩm cho mỗi loại Axit Amin :

Những phát hiện chính:

- Soy protein isolate (đạm đậu nành tinh khiết) là nguồn giàu nhất cho phần lớn các loại axit amin thiết yếu (như Leucine, Valine, Phenylalanine).
- Gelatins (dạng bột, không đường) là nguồn cung cấp dồi dào nhất cho Alanine, Glycine và Proline.
- Dữ liệu còn bao gồm các thực phẩm đặc thù của vùng Alaska (như thịt cá voi, hải cẩu) cũng là những nguồn giàu axit amin.

3.5 Dataset 5 – 2012 Federal Election Commission (FEC) Database

3.5 Dataset 5

Đọc dữ liệu từ nguồn Internet

- 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.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1001731 entries, 0 to 1001730
Data columns (total 16 columns):
 #   Column             Non-Null Count    Dtype
---  ------             --------------    -----
 0   cmte_id            1001731 non-null  object
 1   cand_id            1001731 non-null  object
 2   cand_nm            1001731 non-null  object
 3   contbr_nm          1001731 non-null  object
 4   contbr_city        1001712 non-null  object
 5   contbr_st          1001727 non-null  object
 6   contbr_zip         1001620 non-null  object
 7   contbr_employer    988002 non-null   object
 8   contbr_occupation  993301 non-null   object
 9   contb_receipt_amt  1001731 non-null  float64
 10  contb_receipt_dt   1001731 non-null  object
 11  receipt_desc       14166 non-null    object
 12  memo_cd            92482 non-null    object
 13  memo_text          97770 non-null    object
 14  form_tp            1001731 non-null  object
 15  file_num           1001731 non-null  int64
dtypes: float64(1), int64(1), object(14)
memory usage: 122.3+ MB
None
```

3.5 Dataset 5

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
```

```python
# Đọc data từ Internet

url = "https://raw.githubusercontent.com/wesm/pydata-book/3rd-edition/datasets/fec/P00000001-ALL.csv"

fec = pd.read_csv(url, low_memory=False)
fec.head(10)
```

cand_id	cand_nm	contbr_nm	contbr_city	contbr_st	contbr_zip	contbr_employer	contbr_occupation	contb_receipt_amt	contb_receipt_dt	receipt_desc	memo_cd	memo_
P20002978	Bachmann, Michelle	HARVEY, WILLIAM	MOBILE	AL	366010290	RETIRED	RETIRED	250.0	20-JUN-11	NaN	NaN	
P20002978	Bachmann, Michelle	HARVEY, WILLIAM	MOBILE	AL	366010290	RETIRED	RETIRED	50.0	23-JUN-11	NaN	NaN	
P20002978	Bachmann, Michelle	SMITH, LANIER	LANETT	AL	368633403	INFORMATION REQUESTED	INFORMATION REQUESTED	250.0	05-JUL-11	NaN	NaN	
P20002978	Bachmann, Michelle	BLEVINS, DARONDA	PIGGOTT	AR	724548253	NONE	RETIRED	250.0	01-AUG-11	NaN	NaN	
	Bachmann	WARDENBURG	HOT									

3.5 Dataset 5

Phân tích mô tả

Ai là hai ứng cử viên có tổng tiền quyên góp cao nhất ?

Kết quả cho thấy Barack Obama và Mitt Romney là 2 ứng cử viên có tổng số tiền quyên góp cao nhất.

cand_nm	contb_receipt_amt
Obama, Barack	1.335026e+08
Romney, Mitt	8.521925e+07
Paul, Ron	2.072257e+07
Perry, Rick	1.841151e+07
Gingrich, Newt	1.201183e+07
Santorum, Rick	1.072809e+07
Cain, Herman	7.047265e+06
Pawlenty, Timothy	4.255054e+06
Huntsman, Jon	3.204350e+06
Bachmann, Michelle	2.677435e+06
Johnson, Gary Earl	5.669616e+05
Roemer, Charles E. 'Buddy' III	3.674575e+05
McCotter, Thaddeus G	3.703000e+04

3.5 Dataset 5

Trung bình mỗi người quyên góp bao nhiêu cho mỗi ứng cử viên?

- Rick Perry là ứng viên nhận được tổng tiền quyên góp cao nhất, vượt xa các đối thủ khác.
- Các ứng viên như Romney, Huntsman và Pawlenty cũng nhận được mức đóng góp khá lớn.
- Trong khi đó, Obama chỉ đứng giữa bảng, cho thấy dữ liệu này có thể đại diện cho một thời điểm hoặc nhóm quyên góp cụ thể, không phải toàn chiến dịch.

cand_nm	contb_receipt_amt
Bachmann, Michelle	203.762190
Cain, Herman	350.488133
Gingrich, Newt	251.931290
Huntsman, Jon	771.017921
Johnson, Gary Earl	459.450243
McCotter, Thaddeus G	500.405405
Obama, Barack	224.847985
Paul, Ron	144.149966
Pawlenty, Timothy	934.150184
Perry, Rick	1356.280804
Roemer, Charles E. 'Buddy' III	62.070530
Romney, Mitt	794.740686
Santorum, Rick	230.419260

3.5 Dataset 5

Các bang (State) nào quyên góp nhiều nhất ?

```
contbr_st
CA    43118223.68
TX    30405507.20
NY    27444922.03
IL    22011790.66
FL    20904918.76
MA    12054402.40
VA     9788893.45
PA     8520262.85
GA     7727072.36
NJ     7686113.38
Name: contb_receipt_amt, dtype: float64
```

California là bang có tổng tiền quyên góp cao nhất, vượt xa các bang khác.

3.5 Dataset 5

Phân tích so sánh

Giữa Barack Obama và Mitt Romney, ai nhận được nhiều quyên góp nhỏ (<$200) hơn?

cand_nm	total_small_donations_usd	number_of_small_donations
Obama, Barack	24197815.34	438321
Romney, Mitt	2460298.48	38820

- Barack Obama nhận được nhiều quyên góp nhỏ (<$200) hơn hẳn Mitt Romney, cả về tổng số tiền và số lượt đóng góp.
- Điều này cho thấy chiến dịch của Obama có mạng lưới ủng hộ rộng và phổ biến hơn trong dân chúng.
- Ngược lại, Romney chủ yếu dựa vào các nhà tài trợ lớn thay vì quyên góp nhỏ lẻ.

3.5 Dataset 5

Phân tích so sánh

Nghề nghiệp nào đóng góp nhiều nhất cho mỗi bên?

```
Top 15 nghề ủng hộ Obama:
contbr_occupation
RETIRED                    25305116.38
ATTORNEY                   11141982.97
UNKNOWN                     8858213.36
INFORMATION REQUESTED       4866973.96
HOMEMAKER                   4248875.80
PHYSICIAN                   3735124.94
LAWYER                      3160478.87
CONSULTANT                  2459912.71
PROFESSOR                   2165071.08
CEO                         2073284.79
PRESIDENT                   1878509.95
NOT EMPLOYED                1709188.20
EXECUTIVE                   1355161.05
TEACHER                     1251469.15
WRITER                      1084188.88
Name: Obama, Barack, dtype: float64
```

```
Top 15 nghề ủng hộ Romney:
contbr_occupation
RETIRED                                    11508473.59
INFORMATION REQUESTED PER BEST EFFORTS     11396894.84
HOMEMAKER                                   8147446.22
ATTORNEY                                    5364718.82
PRESIDENT                                   2491244.89
EXECUTIVE                                   2300947.03
C.E.O.                                      1968386.11
INVESTOR                                    1537595.12
CONSULTANT                                  1424894.01
PHYSICIAN                                   1368023.96
FINANCE                                     1143592.25
SELF-EMPLOYED                               1092538.44
REAL ESTATE                                 1054801.00
BANKER                                      1009186.24
OWNER                                        875186.24
Name: Romney, Mitt, dtype: float64
```

3.5 Dataset 5

Phân tích so sánh

Nghề nghiệp nào đóng góp nhiều nhất cho mỗi bên?

- Cả Obama và Romney đều nhận được nhiều quyên góp nhất từ nhóm nghỉ hưu (RETIRED), phản ánh lượng cử tri lớn tuổi tích cực đóng góp.
- Những nghề như luật sư, bác sĩ, giáo sư ủng hộ Obama mạnh hơn, thể hiện xu hướng nghiêng về giới trí thức và khu vực công.
- Ngược lại, Romney nhận nhiều từ giới kinh doanh và tài chính như CEO, chủ tịch, nhà đầu tư, cho thấy sức ủng hộ từ khối doanh nghiệp và giới thượng lưu.

3.5 Dataset 5

Phân tích so sánh

Có mối quan hệ nào giữa nghề nghiệp và định hướng chính trị không?

Nghề nghiêng về Obama (tỷ trọng cao):

cand_nm	Obama, Barack	Romney, Mitt
occ		
ZOOLOGY EDUCATION	1.0	0.0
-	1.0	0.0
.NET PROGRAMMER	1.0	0.0
07/13/1972	1.0	0.0
12K ADVOCATE	1.0	0.0
13D	1.0	0.0
1SG RETIRED	1.0	0.0
1ST ASSISTANT DIRECTOR 2ND UNIT	1.0	0.0
1ST GRADE TEACHER	1.0	0.0
1ST VP WEALTH MANAGEMENT	1.0	0.0

Nghề nghiêng về Romney (tỷ trọng cao):

cand_nm	Obama, Barack	Romney, Mitt
occ		
LEAD SCANNER	0.0	1.0
CAPITALIST	0.0	1.0
PRIVATE INVESTER	0.0	1.0
CAPITAL PARTNER	0.0	1.0
CAPITAL MANAGER	0.0	1.0
HOMEWARD BOUND BUILDERS LLC	0.0	1.0
CAPITAL ADVISOR	0.0	1.0
CAPITAL ACCOUNTANT	0.0	1.0
PRIVATE INVESTING	0.0	1.0
DIRECTOR OF COMPETITIVE INTELLIGENCE A	0.0	1.0

3.5 Dataset 5

Phân tích so sánh

Có mối quan hệ nào giữa nghề nghiệp và định hướng chính trị không?

- Các nghề như ENGINEER, TEACHER, LAWYER, SCIENTIST có tỷ lệ ủng hộ Obama cao hơn, thể hiện xu hướng nghiêng về Đảng Dân chủ của giới trí thức và công nghệ.
- Ngược lại, các nghề như INVESTOR, EXECUTIVE, FINANCE, BUSINESS OWNER lại đóng góp nhiều cho Romney, phản ánh sự ủng hộ mạnh từ giới kinh doanh và tài chính.
- Như vậy, nghề nghiệp có mối quan hệ rõ ràng với định hướng chính trị, phản ánh sự phân hóa kinh tế – xã hội trong hành vi quyên góp.

3.5 Dataset 5

Phân tích so sánh

Các ngành (công nghệ, giáo dục, tài chính) có xu hướng ủng hộ đảng nào?

cand_nm	Obama, Barack	Romney, Mitt
industry		
OTHER	1.195507e+08	78811381.65
EDUCATION	6.532388e+06	791634.18
HEALTHCARE	3.601266e+06	1449357.60
FINANCE	3.086851e+06	6526546.95
GOVERNMENT	2.054404e+06	540890.95
TECH	1.051805e+06	216096.20

3.5 Dataset 5

Phân tích so sánh

Các ngành (công nghệ, giáo dục, tài chính) có xu hướng ủng hộ đảng nào?

- Ngành giáo dục (EDUCATION) và công nghệ (TECH) quyên góp nhiều hơn cho Barack Obama, thể hiện xu hướng ủng hộ Đảng Dân chủ của giới trí thức và công nghệ.
- Ngược lại, ngành tài chính (FINANCE) và chăm sóc sức khỏe (HEALTHCARE) đóng góp mạnh cho Mitt Romney, phản ánh sự ủng hộ từ giới kinh doanh và tài chính đối với Đảng Cộng hòa.
- Nhìn chung, các ngành tri thức và công nghệ nghiêng về Obama, còn ngành tài chính và doanh nghiệp nghiêng về Romney.

3.5 Dataset 5

	contb_receipt_amt
0	(100, 1000]
1	(0, 100]
2	(100, 1000]
3	(100, 1000]
4	(100, 1000]
...	...
1001726	(1000, 10000]
1001727	(1000, 10000]
1001728	(100, 1000]
1001729	(100, 1000]
1001730	(1000, 10000]

1001731 rows × 1 columns

Phân tích so sánh

Tỷ lệ quyên góp nhỏ (< 1000)so với lớn(>1000) của mỗi ứng cử viên?

- Phần lớn các khoản quyên góp cho Barack Obama nằm trong nhóm nhỏ hơn $1.000, cho thấy chiến dịch dựa nhiều vào đóng góp đại chúng, quy mô nhỏ.
- Ngược lại, Mitt Romney nhận tỷ lệ cao hơn từ các khoản lớn trên $1.000, phản ánh sự ủng hộ mạnh từ giới tài trợ giàu có và doanh nghiệp.
- Nhìn chung, Obama có nền tảng ủng hộ rộng rãi, còn Romney có ít người ủng hộ nhưng giá trị trung bình cao hơn.

3.5 Dataset 5

Phân tích so sánh

Ứng cử viên nào có lượng người ủng hộ đông đảo hơn (nhiều người, ít tiền mỗi người)?

cand_nm	unique_donors	total_amount	avg_per_donor
Obama, Barack	196631	1.358774e+08	691.027494
Romney, Mitt	66960	8.833591e+07	1319.233983

- Barack Obama có số lượng người ủng hộ nhiều gấp gần 3 lần so với Mitt Romney, cho thấy mạng lưới quyên góp rộng và phổ biến hơn.
- Tuy nhiên, mức quyên góp trung bình của Obama (≈$691/người) thấp hơn Romney (≈$1.319/người).
- Điều này cho thấy Obama thu hút nhiều người quyên góp nhỏ, còn Romney dựa vào ít người nhưng đóng góp lớn hơn.

3.5 Dataset 5

Phân tích so sánh

Bang nào đóng góp nhiều nhất cho mỗi ứng cử viên?

	cand_nm	contbr_st	contb_receipt_amt
6	Bachmann, Michelle	CA	357464.77
101	Cain, Herman	TX	947138.02
122	Gingrich, Newt	FL	1620117.50
174	Huntsman, Jon	CA	403495.00
229	Johnson, Gary Earl	CA	84784.88
280	McCotter, Thaddeus G	MI	18200.00
302	Obama, Barack	CA	23370680.84
364	Paul, Ron	CA	2781956.09
441	Pawlenty, Timothy	MN	1046118.38
517	Perry, Rick	TX	10782562.75
547	Roemer, Charles E. 'Buddy' III	LA	81527.24
590	Romney, Mitt	CA	10908232.46
695	Santorum, Rick	TX	1506343.12

- California (CA) là bang đóng góp nhiều nhất cho hầu hết các ứng viên, đặc biệt là Barack Obama và Mitt Romney.
- Texas (TX) cũng nổi bật với mức quyên góp cao cho các ứng viên Cộng hòa như Perry và Santorum.
- Điều này cho thấy các bang đông dân, kinh tế mạnh như CA và TX đóng vai trò chủ chốt trong huy động tài chính bầu cử.

3.5 Dataset 5

Phân tích so sánh

Có sự khác biệt vùng miền (Bờ Đông / Bờ Tây) trong hành vi quyên góp không?

cand_nm coast	Obama, Barack	Romney, Mitt
EAST COAST	59423244.82	42616790.81
OTHER	46078269.84	32415753.59
WEST COAST	30375912.58	13303363.13

- Kết quả cho thấy có sự khác biệt vùng miền rõ rệt trong hành vi quyên góp.
- Cả Bờ Đông (East Coast) và Bờ Tây (West Coast) đều quyên góp nhiều hơn cho Barack Obama, đặc biệt Bờ Đông dẫn đầu với hơn 59 triệu USD.
- Điều này phản ánh xu hướng truyền thống: Bờ Đông và Bờ Tây nghiêng về Đảng Dân chủ, trong khi các bang nội địa (Other) có xu hướng ủng hộ Mitt Romney nhiều hơn.

APPENDIX ADVANCED NUMPY

A.1 ndarray Object Internals

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)
-

A.1 ndarray Object Internals

2. Nội dung & Giải thích chi tiết

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).

A.1 ndarray Object Internals

2. Nội dung & Giải thích chi tiết

Cấu trúc ndarray gồm bốn thành phần chính:

- 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.

A.1 ndarray Object Internals

2. Nội dung & Giải thích chi tiết

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.

A.1 ndarray Object Internals

Ví dụ về Shape và Strides:

```
In [12]: np.ones((10, 5)).shape
Out[12]: (10, 5)
```

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).

A.1 ndarray Object Internals

Ví dụ về Shape và Strides:

```
In [13]: np.ones((3, 4, 5), dtype=np.float64).strides
Out[13]: (160, 40, 8)
```

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.

A.1 ndarray Object Internals

3. 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:

```
In [14]: ints = np.ones(10, dtype=np.uint16)

In [15]: floats = np.ones(10, dtype=np.float32)

In [16]: np.issubdtype(ints.dtype, np.integer)
Out[16]: True
```

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).

Ví dụ:

```
In [14]: ints = np.ones(10, dtype=np.uint16)

In [15]: floats = np.ones(10, dtype=np.float32)

In [16]: np.issubdtype(ints.dtype, np.integer)
Out[16]: True
```

Giải thích:

- 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.

A.1 ndarray Object Internals

4. 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.

A.1 ndarray Object Internals

4. Kết quả & Nhận xét

- 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.

A.2 Advanced Array Manipulation

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

A.2 Advanced Array Manipulation

2. Các bước & Khái niệm chính:

- 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:

```
In [20]: arr = np.arange(8)

In [21]: arr
Out[21]: array([0, 1, 2, 3, 4, 5, 6, 7])

In [22]: arr.reshape((4, 2))
Out[22]:
array([[0, 1],
 [2, 3],
 [4, 5],
 [6, 7]])
```

- 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.

A.2 Advanced Array Manipulation

2. Các bước & Khái niệm chính:

- 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:

```
In [29]: arr = np.arange(15).reshape((5, 3))

In [30]: arr
Out[30]:
array([[ 0, 1, 2],
 [ 3, 4, 5],
 [ 6, 7, 8],
 [ 9, 10, 11],
 [12, 13, 14]])

In [31]: arr.ravel()

Out[31]: array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
10, 11, 12, 13, 14])
```

- 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).

A.2 Advanced Array Manipulation

2. Các bước & Khái niệm chính:

- 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:

```
In [33]: arr = np.arange(12).reshape((3, 4))

In [34]: arr
Out[34]:
array([[ 0, 1, 2, 3],
 [ 4, 5, 6, 7],
 [ 8, 9, 10, 11]])

In [35]: arr.ravel()
Out[35]: array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11])

In [36]: arr.ravel('F')
Out[36]: array([ 0, 4, 8, 1, 5, 9, 2, 6, 10, 3, 7, 11])
```

- 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).

A.2 Advanced Array Manipulation

2. Các bước & Khái niệm chính:

- Concatenating và Splitting Arrays

Ví dụ:

```
In [37]: arr1 = np.array([[1, 2, 3], [4, 5, 6]])

In [38]: arr2 = np.array([[7, 8, 9], [10, 11, 12]])

In [39]: np.concatenate([arr1, arr2], axis=0)
Out[39]:
array([[ 1, 2, 3],
 [ 4, 5, 6],
 [ 7, 8, 9],
 [10, 11, 12]])

In [40]: np.concatenate([arr1, arr2], axis=1)
Out[40]:
array([[ 1, 2, 3, 7, 8, 9],
 [ 4, 5, 6, 10, 11, 12]])
```

- 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.

A.2 Advanced Array Manipulation

2. Các bước & Khái niệm chính:

- Repeating Elements – Hàm repeat và tile

Ví dụ:

```
In [55]: arr = np.arange(3)

In [56]: arr
Out[56]: array([0, 1, 2])

In [57]: arr.repeat(3)
Out[57]: array([0, 0, 0, 1, 1, 1, 2, 2, 2])
```

- 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.

A.2 Advanced Array Manipulation

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.

A3 - BROADCASTING

A3 – Broadcasting

Khái niệm

- Broadcasting là cơ chế tự động mở rộng kích thước mảng khi thực hiện phép toán giữa các mảng khác shape.
- Giúp loại bỏ vòng lặp, tăng tốc và giữ code ngắn gọn.

Quy tắc

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

The Broadcasting Rule

Two arrays are compatible for broadcasting if for each *trailing dimension* (i.e., starting from the end) the axis lengths match or if either of the lengths is 1. Broadcasting is then performed over the missing or length 1 dimensions.

Figure A-4. Broadcasting over axis 0 with a 1D array

A3 - Broadcasting

Ứng dụng Broadcasting

- Ví dụ 1 – Demean theo cột, Thực hiện arr - arr.mean(0) → broadcasting tự động theo trục 0.
- Ví dụ 2 – Demean theo hàng, Cần reshape (4,) → (4,1) để broadcast theo trục 1.

```
arr - arr.mean(1).reshape(4,1)
```

- Ví dụ 3 – Mảng 3D, Broadcast dọc theo trục sâu (axis=2).

Figure A-5. Broadcasting over axis 1 of a 2D array

See Figure A-6 for another illustration, this time adding a two-dimensional array to a three-dimensional one across axis 0.

A3 – Broadcasting

Hàm tổng quát hóa Broadcasting

Hàm demean_axis():

```
def demean_axis(arr, axis=0):
    means = arr.mean(axis)

    # This generalizes things like [:, :, np.newaxis] to N dimensions
    indexer = [slice(None)] * arr.ndim
    indexer[axis] = np.newaxis
    return arr - means[indexer]
```

Tự động thêm chiều 1 tại trục chỉ định, giúp broadcast đúng hướng.

Ứng dụng

- Chuẩn hóa dữ liệu nhiều chiều
- Tránh tạo mảng tạm → tiết kiệm bộ nhớ
- Dễ mở rộng cho N chiều

A.4 ADVANCED UFUNC USAGE

A.4 Advanced ufunc Usage

- Khái niệm
- ufunc (universal function): hàm vector hóa thực hiện phép toán theo từng phần tử.
- Viết bằng C, rất nhanh, hỗ trợ broadcasting, reduce, accumulate, outer, reduceat.
- ◆ Ưu điểm
- Không cần vòng lặp Python
- Tự động tối ưu bộ nhớ & SIMD
- Có thể tạo ufunc tùy chỉnh (custom ufuncs)

- Ý nghĩa của sơ đồ “ufunc layer”
- Mục tiêu: minh họa rằng khi bạn viết 1 phép toán như: z = np.add(x, y)
- thì:
- Dù bạn chỉ gõ 1 dòng Python,
- Thực chất NumPy không lặp Python từng phần tử, mà:
- Python Layer chỉ gửi lệnh.
- NumPy Layer (C API) xử lý logic vector hóa, broadcasting.
- C-Level Loop (SIMD / BLAS) thực thi phép tính thực sự trong bộ nhớ.

Cấu trúc thực thi của NUMPY UFUNC

A.4 Advanced ufunc Usage

Các phương thức ufunc nâng cao

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.

A.4 Advanced ufunc Usage

Tự tạo ufunc Python

Cách 1 – np.frompyfunc

-->> Trả về mảng dtype=object.

```
In [139]: def add_elements(x, y):
   .....:     return x + y

In [140]: add_them = np.frompyfunc(add_elements, 2, 1)

In [141]: add_them(np.arange(8), np.arange(8))
Out[141]: array([0, 2, 4, 6, 8, 10, 12, 14], dtype=object)
```

Cách 2 – np.vectorize

-->> Cho phép chọn kiểu dữ liệu đầu ra.

```
In [142]: add_them = np.vectorize(add_elements, otypes=[np.float64])

In [143]: add_them(np.arange(8), np.arange(8))
Out[143]: array([ 0.,  2.,  4.,  6.,  8., 10., 12., 14.])
```

A.4 Advanced ufunc Usage

So sánh hiệu năng

Phương pháp:	Thời gian (10000 phần tử)
• vectorize Python	• ~2.4 ms
~	
• np.add (C-level)	• ~2.9 µs ⚡

```
In [144]: arr = rng.standard_normal(10000)

In [145]: %timeit add_them(arr, arr)
2.43 ms +- 30.5 us per loop (mean +- std. dev. of 7 runs, 100 loops each)

In [146]: %timeit np.add(arr, arr)
2.88 us +- 47.9 ns per loop (mean +- std. dev. of 7 runs, 100000 loops each)
```

Kết luận: A3 & A4

- Broadcasting: tự mở rộng chiều → phép toán nhanh, ngắn, chính xác.
- ufunc nâng cao: cung cấp công cụ xử lý mảng hiệu quả mà không cần vòng lặp.
- Có thể tự tạo ufunc Python để mở rộng khả năng của NumPy.
- Hiểu rõ broadcasting và ufunc giúp tối ưu hiệu năng cho các thuật toán khoa học dữ liệu.

A.5 Structured and Record Arrays

Structured Arrays là gì?

- Khái niệm: Là một ndarray đặc biệt trong NumPy, cho phép lưu trữ các "bản ghi" (records) chứa nhiều trường (fields) với các kiểu dữ liệu khác nhau (heterogeneous).
- Tương đồng với:
- Một struct trong ngôn ngữ C.
- Một hàng (row) trong bảng SQL.
- Mục đích chính: Biểu diễn dữ liệu dạng bảng trong cùng một khối bộ nhớ liên tục, tối ưu cho hiệu suất và xử lý nhị phân.

A.5 Structured and Record Arrays

Các loại và cách truy cập

1. Dtype cơ bản: Mỗi bản ghi có các trường đơn giản.

- Truy cập toàn bộ một trường bằng tên: sarr['x'].

2. Trường đa chiều (Multidimensional): Một trường có thể là một mảng, hữu ích cho việc lưu trữ vector cố định như tọa độ 3D.

3. Trường lồng nhau (Nested): Một trường có thể chứa một cấu trúc con, cho phép biểu diễn dữ liệu phức tạp dạng cây.

- Truy cập lồng nhau: data['x']['a'].

```
import numpy as np
dtype = [('x', np.float64), ('y', np.int32)]
sarr = np.array([(1.5, 6), (np.pi, -2)], dtype=dtype)
```

A.5 Structured and Record Arrays

Ứng dụng & So sánh với Pandas

Điểm mạnh vượt trội:

- Hiệu suất cao: Dữ liệu lưu liên tục giúp đọc/ghi file nhị phân cực nhanh (.tofile(), .fromfile()).
- Tương thích C/C++: Layout bộ nhớ giống struct trong C, cho phép chia sẻ dữ liệu trực tiếp mà không cần chuyển đổi.

So sánh nhanh:

- Structured Array: Là công cụ cấp thấp (low-level), tối ưu cho I/O nhị phân và các tác vụ cần hiệu năng bộ nhớ.
- Pandas DataFrame: Là công cụ cấp cao (high-level), linh hoạt, mạnh mẽ cho các thao tác phân tích dữ liệu như groupby, join, merge.

A.6 More About Sorting

Các phương pháp Sắp xếp chính :

1. Sắp xếp tại chỗ (In-place): arr.sort()

- Thay đổi trực tiếp mảng gốc, không tạo bản sao.
- Ưu điểm: Nhanh và tiết kiệm bộ nhớ.

2. Sắp xếp trả về bản sao: np.sort()

- Tạo ra một mảng mới đã được sắp xếp, không ảnh hưởng mảng gốc.
- Cho phép sắp xếp theo hàng (axis=1) hoặc cột (axis=0).

3. Sắp xếp gián tiếp: argsort()

- Không trả về giá trị, mà trả về chỉ số (index) của các phần tử sau khi sắp xếp.
- Hữu ích khi cần áp dụng cùng một thứ tự sắp xếp cho nhiều mảng khác nhau.

A.6 More About Sorting

Ứng dụng thực tế của Sắp xếp :

Tìm kiếm nhị phân: searchsorted() :

- Hoạt động trên mảng đã sắp xếp, trả về chỉ số mà một phần tử mới có thể được chèn vào để duy trì thứ tự.

Ứng dụng chính: Phân nhóm dữ liệu (Binning):

- Kết hợp searchsorted() với pandas.groupby() để phân loại dữ liệu vào các khoảng giá trị một cách hiệu quả.

Các bài toán thường gặp:

- Khoa học dữ liệu: Chọn top-k sản phẩm, người dùng....
- Xử lý dữ liệu lớn: Giảm chi phí tính toán khi chỉ cần các giá trị cực trị.

A.6 More About Sorting

Các kỹ thuật Sắp xếp nâng cao :

Sắp xếp nhiều khóa: lexsort() :

- Hoạt động như ORDER BY trong SQL, sắp xếp dựa trên một chuỗi các khóa theo thứ tự ưu tiên.

Sắp xếp ổn định: kind='mergesort' :

- Đảm bảo các phần tử có giá trị bằng nhau sẽ giữ nguyên thứ tự tương đối ban đầu của chúng.

Sắp xếp một phần: partition() :

- Rất hiệu quả cho bài toán tìm "top-k" (ví dụ: tìm 5 giá trị lớn nhất) mà không cần sắp xếp toàn bộ mảng.

A.7 Writing Fast NumPy Functions with Numba

Bài toán: Tính giá trị trung bình của hiệu hai mảng số thực có kích thước rất lớn

```
N = 10_000_000
x = rng.standard_normal(N)
y = rng.standard_normal(N)
```

```
def mean_distance(x, y):
  nx = len(x)
  result = 0.0
  count = 0
  for i in range(nx):
    result += x[i] - y[i]
    count += 1
```

A.7

```
# Sử dụng Generator mới của NumPy để tái lập kết quả
rng = np.random.default_rng(seed=42)
N = 10_000_000

x = rng.standard_normal(N)
y = rng.standard_normal(N)

# So sánh tốc độ thực thi
print("Check (Python) bằng hàm :")
%timeit mean_distance(x, y)
print("----------------")

print("Check (NumPy):")
%timeit (x - y).mean()
```

```
Check (Python) bằng hàm :
79 ms ± 9.57 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
----------------
Check (NumPy):
45.4 ms ± 2.26 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

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.

A.7

```
@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
```

```
Check (Numba njit):
19.7 ms ± 710 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

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.

A.7

So sánh tốc độ thực thi cho mean(x - y) (N = 10 triệu)
Numba @njit
19.7 ms | 4.01× nhanh hơn Python
NumPy (x - y).mean()
45.4 ms | 1.74× nhanh hơn Python
Python for-loop
79.0 ms | 1.00× nhanh hơn Python
0
10
20
30
40
50
60
70
80
Thời gian (ms) – thấp hơn là tốt hơn

A.7

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.

Creating Custom numpy.ufunc Objects with Numba

```
#Tạo một hàm cộng phần tử bằng Numba — dạng ufunc

from numba import vectorize
@vectorize
def nb_add(x, y):
  return x + y
```

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.

A.8 Advanced Array Input and Output

A.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

```
def mean_distance(x, y):
  nx = len(x)
  result = 0.0
  count = 0
  for i in range(nx):
    result += x[i] - y[i]
    count += 1
```

A.8.1

Tạo mảng bộ nhớ ánh xạ:

```
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.]])
```

Mảng sau khi ghi ngẫu nhiên 5 hàng đầu

```
memmap([[-0.0087519 , -0.24494243, -0.78151656, ...,  0.32711373,
          0.68250605, -1.17411721],
        [ 1.11921346,  0.40332401, -0.33454857, ..., -0.07208116,
         -0.67031833,  0.85151074],
        [-1.001989  ,  0.6810419 , -0.88092982, ...,  0.27721638,
         -1.16054174,  0.03511989],
        ...,
        [ 0.        ,  0.        ,  0.        , ...,  0.        ,
          0.        ,  0.        ],
        [ 0.        ,  0.        ,  0.        , ...,  0.        ,
          0.        ,  0.        ],
        [ 0.        ,  0.        ,  0.        , ...,  0.        ,
          0.        ,  0.        ]])
```

A.8.1

```
#Giải phóng (đóng file)
del mmap
```

```
## Mở lại (phải cung cấp dtype & shape vì file thuần nhị phân)
mmap = np.memmap('mymmap', dtype='float64', shape=(10000, 10000))
```

```
mmap
```

```
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.]])
```

```
#Xóa biến
%xdel mmap
```

```
#Xóa tệp
!rm mymmap
```

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.

A.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

A.8.2

```python
import h5py, numpy as np

with h5py.File("data.h5", "w") as f:
    dset = f.create_dataset(
        "X",
        shape=(10_000, 10_000),
        dtype="float32",
        chunks=(256, 256),
        compression="gzip",
        compression_opts=4,
    )

    CH = 256
    block = np.ones((CH, 10_000), dtype="float32")

    for i in range(0, 10_000, CH):
        r = min(CH, 10_000 - i)        # số hàng còn lại ở cuối (có thể < CH)
        dset[i:i+r, :] = block[:r, :]  # cắt block cho khớp kích thước
```

```python
with h5py.File("data.h5", "r") as f:
    print("Datasets:", list(f.keys()))
    X = f["X"]
    print("shape:", X.shape, "dtype:", X.dtype, "chunks:", X.chunks, "compression:", X.compression)
```

A.8.2

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.

A.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.

A.9

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

```
array([[1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       ...,
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.]])
```

A.9

```
#hiển thị các thuộc tính bộ nhớ

arr_c.flags
```

```
  C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
```

```
arr_f.flags
```

```
  C_CONTIGUOUS : False
  F_CONTIGUOUS : True
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
```

Hiển thị các thuộc tính bộ nhớ của mảng

- 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.

A.9

```
#hiển thị các thuộc tính bộ nhớ

arr_c.flags

  C_CONTIGUOUS : True
  F_CONTIGUOUS : False
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False

arr_f.flags

  C_CONTIGUOUS : False
  F_CONTIGUOUS : True
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
```

- 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.

A.9

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

```
mảng arr_c
609 µs ± 109 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
--------------
mảng arr_f
1.71 ms ± 465 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

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.

A.9

Kết luận

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.