

Khoa học dữ liệu

Nguyên tắc cơ bản

cho Python và

MongoDB

–  
Giấy David



Apress®

[www.allitebooks.com](http://www.allitebooks.com)

Khoa học dữ liệu  
Nguyên tắc cơ bản cho  
Python và MongoDB

Giáy David

Apress®

## Nguyên tắc cơ bản về khoa học dữ liệu cho Python và MongoDB

Giấy David

Logan, Utah, Hoa Kỳ

ISBN-13 (pbk): 978-1-4842-3596-6

ISBN-13 (điện tử): 978-1-4842-3597-3

<https://doi.org/10.1007/978-1-4842-3597-3>

Thư viện Quốc hội Số kiểm soát: 2018941864

Bản quyền © 2018 của David Paper

Công trình này là có bản quyền. Tất cả các quyền được Nhà xuất bản bảo lưu, cho dù toàn bộ hay một phần tài liệu có liên quan, đặc biệt là quyền dịch thuật, in lại, sử dụng lại các hình minh họa, trích dẫn, phát sóng, sao chép trên vi phim hoặc bằng bất kỳ cách vật lý nào khác, và truyền tải hoặc lưu trữ thông tin và truy xuất, điều chỉnh điện tử, phần mềm máy tính hoặc bằng phương pháp tương tự hoặc không tương tự hiện đã biết hoặc sau này được phát triển.

Tên thương mại, logo và hình ảnh có thể xuất hiện trong cuốn sách này. Thay vì sử dụng biểu tượng nhãn hiệu với mỗi lần xuất hiện tên, logo hoặc hình ảnh đã đăng ký nhãn hiệu, chúng tôi chỉ sử dụng tên, logo và hình ảnh theo cách biên tập và vì lợi ích của chủ sở hữu nhãn hiệu, không có ý định vi phạm nhãn hiệu.

Việc sử dụng tên thương mại, nhãn hiệu, nhãn hiệu dịch vụ và các thuật ngữ tương tự trong án phầm này, ngay cả khi chúng không được xác định như vậy, không được coi là một biểu hiện quan điểm về việc chúng có thuộc quyền sở hữu hay không.

Mặc dù những lời khuyên và thông tin trong cuốn sách này được cho là đúng và chính xác tại ngày xuất bản, cả tác giả, biên tập viên cũng như nhà xuất bản đều không chịu bất kỳ trách nhiệm pháp lý nào đối với bất kỳ lỗi hoặc thiếu sót nào có thể xảy ra. Nhà xuất bản không bao đảm, rõ ràng hay ngụ ý, đối với tài liệu có trong tài liệu này.

Giám đốc điều hành, Apress Media LLC: Welmoed Spahr

Biên tập viên mua lại: Jonathan Gennick

Biên tập phát triển: Laura Berendson

Phối hợp biên tập: Jill Balzano

Bìa được thiết kế bởi eStudioCalamar

Ảnh bìa được thiết kế bởi Freepik ([www.freepik.com](http://www.freepik.com))

Được phân phối cho việc buôn bán sách trên toàn thế giới bởi Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Điện thoại 1-800-SPRINGER, fax (201) 348-4505, e-mail [order-ny@springer-sbm.com](mailto:order-ny@springer-sbm.com), hoặc truy cập [www.springeronline.com](http://www.springeronline.com). Apress Media, LLC là một công ty trách nhiệm hữu hạn ở California và thành viên (chủ sở hữu) duy nhất là Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc là một tập đoàn Delaware.

Để biết thông tin về bản dịch, vui lòng gửi email tới địa chỉ [right@apress.com](mailto:right@apress.com) hoặc truy cập <http://www.apress.com/rights-permissions>.

Các đầu sách Apress có thể được mua với số lượng lớn để sử dụng trong học tập, công ty hoặc quảng cáo. Các phiên bản sách điện tử và giấy phép cũng có sẵn cho hầu hết các đầu sách. Để biết thêm thông tin, hãy tham khảo trang web Bán hàng loạt sách in và sách điện tử của chúng tôi tại <http://www.apress.com/bulk-sales>.

Bất kỳ mã nguồn hoặc tài liệu bổ sung nào khác được tác giả tham chiếu trong cuốn sách này đều có sẵn cho độc giả trên GitHub thông qua trang sản phẩm của cuốn sách, có tại [www.apress.com/9781484235966](http://www.apress.com/9781484235966). Để biết thêm thông tin chi tiết, vui lòng truy cập <http://www.apress.com/mã nguồn>.

In trên giấy không có axit

Gửi tới Lady, Sam, Bruce, Malik, John, Moonshadow và Moonbeam, những người luôn ủng hộ và yêu thương vô điều kiện.

Gửi tới đội ngũ nhân viên của Apress vì tất cả sự hỗ trợ và nỗ lực của bạn trong việc thực hiện dự án này. Cuối cùng, xin gửi lời cảm ơn đặc biệt đến Jonathan vì đã tìm thấy tôi trên Amazon, Jill vì đã ủng hộ một tác giả khó tính và Mark vì đã đánh giá kỹ thuật kỹ lưỡng và mang tính xây dựng.

# Mục lục

Giới thiệu về tác giả	ix
Giới thiệu về Người đánh giá kỹ thuật	xii
Sự nhìn nhận	xiii
<b>Chương 1: Giới thiệu</b>	<b>1</b>
Nguyên tắc cơ bản về Python	1
Hàm và Chuỗi	3
Danh sách, Bộ dữ liệu và Tù điện	5
Đọc và ghi dữ liệu	12
Danh sách Hiệu	15
Máy phát điện	18
Ngẫu nhiên hóa dữ liệu	22
MongoDB và JSON	27
Trục quan hóa	34
<b>Chương 2: Mô phỏng Monte Carlo và Hàm mật độ</b>	<b>37</b>
Mô phỏng Chứng khoán	37
Phản tích What-If	42
Mô phỏng nhu cầu sản phẩm	44
Ngẫu nhiên Số dùng Hàm Xác suất và Mật độ Tích lũy	52

## Mục lục

### Chương 3: Đại số tuyến tính 67

Không Gian Véc Tô	57
Toán Vector	68
Toán Ma Trận	75
Các Phép Biến Đổi Ma Trận Cơ Bản	84
Ứng dụng Ma trận Pandas	88

### Chương 4: Dốc độ dốc 97

Tối thiểu hóa chức năng đơn giản (và Tối đa hóa)	97
Giảm thiểu chức năng sigmoid (và tối đa hóa)	104
Kiểm soát giảm thiểu khoảng cách Euclidean cho kích thước bước	109
Ôn định giảm thiểu khoảng cách Euclidean với	
Mô phỏng Monte Carlo	112
Thay thế một phương pháp NumPy để đẩy nhanh Euclidean	
Giảm thiểu Khoảng cách	115
Tối thiểu hóa và tối đa hóa độ dốc ngẫu nhiên	118

### Chương 5: Làm việc với dữ liệu 129

Ví dụ về dữ liệu một chiều	129
Ví dụ về dữ liệu hai chiều	132
Tương quan dữ liệu và thống kê cơ bản	135
Pandas Correlation và Heat Map Ví dụ	138
Nhiều ví dụ trực quan hóa khác nhau	141
Dọn dẹp Tệp CSV bằng Pandas và JSON	146
Cắt lát và thái hạt lựu	148
Data Cube	149
Chia tỷ lệ và sắp xếp dữ liệu	154

## Mục lục

### Chương 6: Khám phá dữ liệu

167

Bản đồ nhiệt	167
Phân tích thành phần chính	170
Mô phỏng tốc độ	179
Dữ liệu lớn	182
Twitter	201
Web Scraping	205

## Mục lục

211

## Giới thiệu về tác giả



David Paper là giáo sư chính thức tại Đại học Bang Utah thuộc khoa Hệ thống Thông tin Quản lý. Cuốn sách của anh ấy Lập trình web cho doanh nghiệp: Lập trình hướng đối tượng PHP với Oracle được xuất bản vào năm 2015 bởi Routledge. Ông cũng có hơn 70 án phẩm trên các tạp chí được giới thiệu như Phương pháp nghiên cứu tổ chức, Truyền thông của ACM,

Thông tin & Quản lý, Tạp chí Quản lý Tài nguyên Thông tin, Truyền thông của AIS, Tạp chí Nghiên cứu Ứng dụng và Trường hợp Công nghệ Thông tin, và Lập kế hoạch Tầm xa. Anh ấy cũng đã phục vụ trong một số ban biên tập với nhiều cương vị khác nhau, bao gồm cả phó tổng biên tập.

Bên cạnh việc lớn lên trong các doanh nghiệp gia đình, Tiến sĩ Paper đã làm việc cho Texas Instruments, DLS, Inc. và Cơ quan quản lý doanh nghiệp nhỏ Phoenix.

Ông đã thực hiện công việc tư vấn về IS cho IBM, AT&T, Octel, Bộ Giao thông Vận tải Utah và Phòng thí nghiệm Động lực học Không gian.

Các lĩnh vực nghiên cứu và giảng dạy của Tiến sĩ Paper bao gồm khoa học dữ liệu, học máy, tái cấu trúc quy trình, lập trình hướng đối tượng, quản lý quan hệ khách hàng điện tử, quản lý thay đổi, thương mại điện tử và tích hợp doanh nghiệp.

## Giới thiệu về Người đánh giá kỹ thuật

Mark Furman, MBA là kỹ sư hệ thống, tác giả, giáo viên và doanh nhân. Trong 16 năm qua, anh ấy đã làm việc trong lĩnh vực Công nghệ thông tin, tập trung vào các hệ thống dựa trên Linux và lập trình bằng Python, làm việc cho nhiều công ty bao gồm Host Gator, Interland, Suntrust Bank, AT&T và Winn-Dixie. Hiện tại, anh ấy đang tập trung sự nghiệp của mình vào phong trào nhà sản xuất và đã ra mắt Tech Forge ([techforge.org](http://techforge.org)), tập trung vào việc giúp mọi người bắt đầu không gian nhà sản xuất và giúp duy trì không gian hiện tại. Ông có bằng Thạc sĩ Quản trị Kinh doanh của Đại học Ohio. Bạn có thể theo dõi anh ấy trên Twitter @mfurman.

# Sự nhìn nhận

Bước vào phân tích dữ liệu của tôi bắt đầu bằng việc khám phá Python để phân tích dữ liệu của Wes McKinney, mà tôi đặc biệt giới thiệu cho mọi người. Bước vào khoa học dữ liệu của tôi bắt đầu bằng việc khám phá Khoa học dữ liệu từ đầu của Joel Grus.

Cuốn sách của Joel có thể không dành cho người yếu tim, nhưng nó chắc chắn là một thử thách mà tôi rất vui vì đã chấp nhận! Cuối cùng, tôi cảm ơn tất cả những người đóng góp cho stackoverflow, những người không thể thiếu các giải pháp lập trình.

## CHƯƠNG 1

# Giới thiệu

Khoa học dữ liệu là một lĩnh vực liên ngành bao gồm các phương pháp, quy trình và hệ thống khoa học để trích xuất kiến thức hoặc hiểu biết sâu sắc từ dữ liệu ở nhiều dạng khác nhau, có cấu trúc hoặc không có cấu trúc. Nó rút ra các nguyên tắc từ toán học, thống kê, khoa học thông tin, khoa học máy tính, học máy, trực quan hóa, khai thác dữ liệu và phân tích dự đoán. Tuy nhiên, về cơ bản nó dựa trên nền tảng toán học.

Cuốn sách này giải thích và áp dụng các nguyên tắc cơ bản của khoa học dữ liệu rất quan trọng đối với các chuyên gia kỹ thuật như DBA và nhà phát triển, những người đang chuyển hướng sự nghiệp sang thực hành khoa học dữ liệu. Đây là một cuốn sách dựa trên ví dụ cung cấp các ví dụ viết mã Python hoàn chỉnh để bổ sung và làm rõ các khái niệm khoa học dữ liệu, đồng thời làm phong phú thêm trải nghiệm học tập. Ví dụ mã hóa bao gồm trực quan hóa bất cứ khi nào thích hợp. Cuốn sách là tiền thân cần thiết để áp dụng và triển khai các thuật toán học máy, bởi vì nó giới thiệu cho người đọc các nguyên tắc cơ bản của khoa học dữ liệu.

Cuốn sách là khép kín. Tất cả các kỹ năng toán học, thống kê, ngẫu nhiên và lập trình cần thiết để nắm vững nội dung đều được đề cập trong cuốn sách. Kiến thức chuyên sâu về lập trình hướng đối tượng là không cần thiết, bởi vì các ví dụ hoạt động và hoàn chỉnh được cung cấp và giải thích. Các ví dụ chuyên sâu và phức tạp khi cần thiết để đảm bảo thu được sự nhạy bén về khoa học dữ liệu phù hợp. Cuốn sách giúp bạn xây dựng các kỹ năng cơ bản cần thiết để làm việc và hiểu các thuật toán khoa học dữ liệu phức tạp.

## Chương 1 Giới thiệu

Nguyên tắc cơ bản về khoa học dữ liệu theo ví dụ là một điểm khởi đầu tuyệt vời cho những người quan tâm đến việc theo đuổi sự nghiệp trong khoa học dữ liệu. Giống như bất kỳ ngành khoa học nào, các nguyên tắc cơ bản của khoa học dữ liệu là điều kiện tiên quyết để có năng lực. Nếu không thông thạo toán học, thống kê, thao tác dữ liệu và mã hóa, con đường dẫn đến thành công cùng lắm là “gập ghềnh”. Các ví dụ viết mã trong cuốn sách này ngắn gọn, chính xác và đầy đủ, đồng thời bổ sung hoàn hảo cho các khái niệm khoa học dữ liệu được giới thiệu.

Cuốn sách được tổ chức thành sáu chương. Chương 1 giới thiệu các nguyên tắc cơ bản về lập trình với “Python” cần thiết để làm việc, biến đổi và xử lý dữ liệu cho các ứng dụng khoa học dữ liệu. Chương 2 giới thiệu mô phỏng Monte Carlo để ra quyết định và phân phối dữ liệu để xử lý thống kê. Chương 3 giới thiệu đại số tuyến tính áp dụng với vectơ và ma trận. Chương 4 giới thiệu thuật toán giảm độ dốc tối thiểu hóa (hoặc tối đa hóa) các chức năng, điều này rất quan trọng vì hầu hết các vấn đề khoa học dữ liệu là các vấn đề tối ưu hóa. Chương 5 tập trung vào việc trộn, làm sạch và chuyển đổi dữ liệu để giải quyết các vấn đề về khoa học dữ liệu. Chương 6 tập trung vào khám phá dữ liệu bằng cách giảm kích thước, quét web và làm việc với các tập dữ liệu lớn một cách hiệu quả.

Mã lập trình Python cho tất cả các ví dụ mã hóa và tệp dữ liệu có sẵn để xem và tải xuống thông qua Apress tại [www.apress.com/9781484235966](http://www.apress.com/9781484235966). Hướng dẫn liên kết cụ thể được bao gồm trên các trang bản quyền của cuốn sách.

Để cài đặt mô-đun Python, pip là chương trình cài đặt ưu tiên. Vì thế, để cài đặt mô-đun matplotlib từ dấu nhắc Anaconda: pip install matplotlib. Anaconda là một bản phân phối mã nguồn mở phổ biến rộng rãi của Python (và R) để xử lý dữ liệu quy mô lớn, phân tích dự đoán và tính toán khoa học giúp đơn giản hóa việc quản lý và triển khai gói. Tôi đã làm việc với các bản phân phối khác với kết quả không đạt yêu cầu, vì vậy tôi đánh giá cao Anaconda.

## Nguyên tắc cơ bản về Python

Python có một số tính năng giúp nó rất phù hợp để học và làm khoa học dữ liệu. Nó miễn phí, viết mã tương đối đơn giản, dễ hiểu và có nhiều thư viện hữu ích để hỗ trợ giải quyết vấn đề về khoa học dữ liệu. Nó cũng cho phép tạo mẫu nhanh hầu như bất kỳ kịch bản khoa học dữ liệu nào và trình diễn các khái niệm khoa học dữ liệu một cách rõ ràng, dễ hiểu thái độ.

Mục tiêu của chương này không phải là dạy Python nói chung, mà là trình bày, giải thích và làm rõ các tính năng cơ bản của ngôn ngữ (chẳng hạn như logic, cấu trúc dữ liệu và thư viện) giúp tạo nguyên mẫu, áp dụng và/hoặc giải quyết các vấn đề về khoa học dữ liệu.

Các nguyên tắc cơ bản về Python bao gồm nhiều hoạt động với các ví dụ mã hóa liên quan như sau:

1. hàm và chuỗi
2. danh sách, bộ dữ liệu và từ điển
3. đọc và ghi dữ liệu
4. hiểu danh sách
5. máy phát điện
6. ngẫu nhiên hóa dữ liệu
7. MongoDB và JSON
8. hình dung

## Hàm và Chuỗi

Các hàm Python là các hàm hạng nhất, có nghĩa là chúng có thể được sử dụng làm tham số, giá trị trả về, được gán cho biến và được lưu trữ trong cấu trúc dữ liệu. Đơn giản, các hàm hoạt động giống như một biến điển hình. Chức năng có thể được

## Chương 1 Giới thiệu

tùy chỉnh hoặc tích hợp sẵn. Tùy chỉnh được tạo bởi lập trình viên, trong khi tích hợp sẵn là một phần của ngôn ngữ. Chuỗi là loại rất phổ biến được đặt trong dấu nháy đơn hoặc kép.

Ví dụ mã sau đây định nghĩa các hàm tùy chỉnh và sử dụng các hàm tích hợp sẵn:

```
def num_to_str(n):
    trả về str(n)

def str_to_int(s):
    trả về int(s)

def str_to_float(f):
    trả về float(f)

if __name__ == "__main__":
    ký hiệu bấm cho phép nhận xét một dòng
    ,,, dấu ngoặc kép cho phép nhận xét nhiều dòng

    float_num = 999.01
    int_num = 87
    float_str = '23.09'
    int_str = '19'
    string = 'bò nâu giờ ra sao'
    s_float = num_to_str(float_num) s_int =
    = num_to_str(int_num) i_str =
    str_to_int(int_str) f_str =
    str_to_float(float_str) ) in
    ( s_float, 'is', type(s_float)) print
    (s_int, 'is', type(s_int)) print
    (f_str, 'is', type(f_str)) print
    (i_str, 'is', type(i_str))
```

## Chương 1 Giới thiệu

```
print ('\nstring', ''' '' co', len(chuỗi), + chuỗi + 'ký tự')
```

```
str_ls = string.split()
in ('chia chuỗi:', str_ls)
print ('danh sách đã tham gia:', ' '.join(str_ls))
```

Đầu ra:

```
999.01 is <class 'str'>
87 is <class 'str'>
23.09 is <class 'float'>
19 is <class 'int'>

string "how now brown cow" has 17 characters
split string: ['how', 'now', 'brown', 'cow']
joined list: how now brown cow
```

Một phong cách mã hóa phổ biến là trình bày các chức năng và nhập thư viện đầu tiên, tiếp theo là khối mã chính. Ví dụ mã bắt đầu với ba hàm tùy chỉnh lần lượt chuyển đổi số thành chuỗi, chuỗi thành số và chuỗi thành float. Mỗi hàm tùy chỉnh trả về một hàm tích hợp sẵn để Python thực hiện chuyển đổi. Khối chính bắt đầu với ý kiến. Nhận xét một dòng được biểu thị bằng ký hiệu # (băm). Nhận xét nhiều dòng được biểu thị bằng ba dấu nháy đơn liên tiếp. Năm dòng tiếp theo gán giá trị cho các biến. Bốn dòng sau đây chuyển đổi từng loại biến thành loại khác. Chẳng hạn, hàm num\_to\_str() chuyển đổi biến float\_num thành kiểu chuỗi. Năm dòng tiếp theo in các biến có kiểu dữ liệu Python được liên kết của chúng. Hàm tích hợp type() trả về loại đối tượng đã cho. Bốn dòng còn lại in và thao tác với một biến chuỗi.

## Chương 1 Giới thiệu

## Danh sách, Bộ dữ liệu và Từ điển

Danh sách là các bộ sưu tập được sắp xếp với các giá trị được phân tách bằng dấu phẩy giữa các dấu ngoặc vuông. Các chỉ số bắt đầu từ 0 (không). Các mục trong danh sách không cần phải cùng loại và có thể được cắt, nối và thao tác theo nhiều cách.

Ví dụ mã sau tạo một danh sách, thao tác và cắt nó, tạo một danh sách mới và thêm các phần tử vào danh sách đó từ một danh sách khác và tạo một ma trận từ hai danh sách:

nhập numpy dưới dạng np

```
nếu __name__ == "__main__":
    ls = ['cam', 'chuối', 10, 'lá', 77.009, 'cây', 'mèo']
    print ('độ dài danh sách:', len(ls), 'mục')
    print ('số lượng mèo:', ls.count('cat'), ',', 'chỉ số mèo:',
          ls.index('cat'))
    print ('\nthao tác danh sách:')
    mèo = ls.pop(6)
    in ('con mèo:', con mèo, ', danh sách:', ls)
    ls.insert(0, 'con mèo')
    ls.append(99)
    in (ls)
    ls[7] = '11'
    in (ls)
    ls.pop(1)
    in (ls)
```

## Chương 1 Giới thiệu

```
ls.pop()
print (ls)
print ('\nslice danh
sách:') print ('3 phần tử đầu tiên:',
ls[:3]) print ('3 phần tử cuối cùng:',
ls[3:]) print ('bắt đầu từ thứ 2 đến chỉ số
5:', ls[1:5]) print ('bắt đầu thứ 3 từ cuối danh sách:', 
ls[-3:]) print ('bắt đầu từ thứ 2 đến tiếp theo cho đến cuối
danh sách :', ls[1:-1]) print ('\n tạo danh sách mới
từ danh sách khác:')
print ('list:', ls)
fruit = ['orange'] more_fruit = ['apple',
'kiwi' , 'lê']
fruit.append(more_fruit)
print
('appended:', fruit)
fruit.pop(1)
fruit.extend(more_fruit)
print ('extended:',
fruit) a, b = fruit[2], fruit[1] print
('slices:', a, b) print ('\n tạo
ma trận từ hai
danh sách:') matrix =
np.array([ls, fruit]) print (ma trận) print ('Hàng thứ nhất:', ma trận[0])
```

## Chương 1 Giới thiệu

Đầu ra:

```
list length: 7 items
cat count: 1 , cat index: 6

manipulate list:
cat: cat , list: ['orange', 'banana', 10, 'leaf', 77.009, 'tree']
['cat', 'orange', 'banana', 10, 'leaf', 77.009, 'tree', 99]
['cat', 'orange', 'banana', 10, 'leaf', 77.009, 'tree', '11']
['cat', 'banana', 10, 'leaf', 77.009, 'tree', '11']
['cat', 'banana', 10, 'leaf', 77.009, 'tree']

slice list:
lst 3 elements: ['cat', 'banana', 10]
last 3 elements: ['leaf', 77.009, 'tree']
start at 2nd to index 5: ['banana', 10, 'leaf', 77.009]
start 3 from end to end of list: ['leaf', 77.009, 'tree']
start from 2nd to next to end of list: ['banana', 10, 'leaf', 77.009]

create new list from another list:
list: ['cat', 'banana', 10, 'leaf', 77.009, 'tree']
appended: ['orange', ['apple', 'kiwi', 'pear']]
extended: ['orange', 'apple', 'kiwi', 'pear']
slices: kiwi apple

create matrix from two lists:
[['cat', 'banana', 10, 'leaf', 77.009, 'tree']
 ['orange', 'apple', 'kiwi', 'pear']]
1st row: ['cat', 'banana', 10, 'leaf', 77.009, 'tree']
2nd row: ['orange', 'apple', 'kiwi', 'pear']
```

Ví dụ mã bắt đầu bằng cách nhập NumPy, đây là gói cơ bản (thư viện, mô-đun) cho tính toán khoa học. Nó rất hữu ích cho đại số tuyến tính, là nền tảng của khoa học dữ liệu. Hãy nghĩ về các thư viện Python như các lớp không lồ với nhiều phương thức. Khởi chính bắt đầu bằng cách tạo danh sách ls, in ra độ dài, số phần tử (mục), số phần tử mèo và chỉ số của phần tử mèo. Mã tiếp tục bằng thao tác ls. Đầu tiên, phần tử thứ 7 (chỉ số 6) được bật lên và gán cho biến cat. Hãy nhớ rằng, các chỉ số danh sách bắt đầu từ 0. Hàm pop() loại bỏ con mèo khỏi ls. Thứ hai, cat được thêm trở lại ls ở vị trí đầu tiên (chỉ số 0) và 99 được thêm vào cuối danh sách. Hàm append() thêm một đối tượng vào cuối danh sách. Thứ ba, chuỗi '11' được thay thế cho phần tử thứ 8 (chỉ số 7).

Cuối cùng, phần tử thứ 2 và phần tử cuối cùng được lấy ra từ ls. Mã tiếp tục bằng cách cắt ls. Đầu tiên, in ba phần tử đầu tiên với ls[:3].

## Chương 1 Giới thiệu

Thứ hai, in ba phần tử cuối cùng với `ls[3:]`. Thứ ba, in bắt đầu bằng phần tử thứ 2 tới các phần tử có chỉ số lên tới 5 với `ls[1:5]`. Thứ tư, in ba phần tử bắt đầu từ đầu đến cuối với `ls[-3:]`. Thứ năm, in bắt đầu từ phần tử thứ 2 đến phần tử cuối cùng với `ls[1:-1]`.

Mã tiếp tục bằng cách tạo một danh sách mới từ một danh sách khác. Đầu tiên, tạo trái cây với một yếu tố. Nối danh sách thứ hai `more_fruit` vào trái cây. Lưu ý rằng `append` thêm danh sách `more_fruit` là thành phần thứ 2 của trái cây, đây có thể không phải là thứ bạn muốn. Vì vậy, thứ ba, bật phần tử thứ 2 của trái cây và mở rộng `more_fruit` thành trái cây. Chức năng  `mở rộng()` làm sáng tỏ một danh sách trước khi thêm nó vào. Bằng cách này, trái cây bây giờ có bốn yếu tố. Thứ tư, gán phần tử thứ 3 cho `a` và phần tử thứ 2 cho `b` và in các lát cắt. Python cho phép gán nhiều biến trên một dòng, rất tiện lợi và ngắn gọn. Đoạn mã kết thúc bằng cách tạo một ma trận từ hai danh sách `-ls` và `fruit-và in` nó. Ma trận Python là một mảng hai chiều (2-D) bao gồm các hàng và cột, trong đó mỗi hàng là một danh sách.

Một bộ là một chuỗi các đối tượng Python bất biến được đặt trong dấu ngoặc đơn. Không giống như danh sách, bộ dữ liệu không thể thay đổi. Bộ dữ liệu thuận tiện với các hàm trả về nhiều giá trị.

Ví dụ mã sau tạo một bộ, cắt nó, tạo một danh sách và tạo một ma trận từ tuple và danh sách:

nhập numpy dưới dạng `np`

```
nếu __name__ == "__main__":
    tup = ('cam', 'chuối', 'nho', 'táo', 'nho')
    in ('độ dài tuple:', len(tup))
    print ('số lượng nho:', tup.count('nho'))
    in ('\nslice tuple:')
    print ('3 phần tử đầu tiên:', tup[:3])
    print('3 phần tử cuối cùng', tup[3:])
    print ('bắt đầu từ thứ 2 đến chỉ số 5', tup[1:5])
    print ('bắt đầu 3 từ đầu đến cuối của tuple:', tup[-3:])
```

## Chương 1 Giới thiệu

```
print ('bắt đầu từ thứ 2 đến tiếp theo cho đến hết bộ:', tup[1:-1])
print ('\n tạo danh sách và tạo ma trận từ nó và tuple:')
trái cây = ['lê', 'bưởi', 'dưa vàng', 'kiwi', 'mận']
matrix = np.array([tup, fruit])
in (ma trận)
```

Đầu ra:

```
tuple length: 5
grape count: 2

slice tuple:
lst 3 elements: ('orange', 'banana', 'grape')
last 3 elements ('apple', 'grape')
start at 2nd to index 5 ('banana', 'grape', 'apple', 'grape')
start 3 from end to end of tuple: ('grape', 'apple', 'grape')
start from 2nd to next to end of tuple: ('banana', 'grape', 'apple')

create list and create matrix from it and tuple:
[['orange' 'banana' 'grape' 'apple' 'grape']
 ['pear' 'grapefruit' 'cantaloupe' 'kiwi' 'plum']]
```

Mã bắt đầu bằng cách nhập NumPy. Khối chính bắt đầu bằng cách tạo tuple tup, in chiều dài, số phần tử (mục), số phần tử nho và chỉ số của loại nho. Mã tiếp tục bằng cách cắt tup. Đầu tiên, in ba phần tử đầu tiên với tup[:3]. Thứ hai, in ba phần tử cuối cùng bằng tup[3:]. Thứ ba, in bắt đầu từ phần tử thứ 2 tới các phần tử có chỉ số lên tới 5 với tup[1:5]. Thứ tư, in ba phần tử bắt đầu từ đầu đến cuối bằng tup[-3:]. Thứ năm, in bắt đầu từ phần tử thứ 2 đến phần tử cuối cùng với tup[1:-1]. Mã tiếp tục bằng cách tạo danh sách trái cây mới và tạo ma trận từ tup và trái cây.

Từ điển là một tập hợp các mục không có thứ tự được xác định bởi khóa/cặp giá trị. Nó là một cấu trúc dữ liệu cực kỳ quan trọng để làm việc với dữ liệu. Ví dụ sau đây rất đơn giản, nhưng phần tiếp theo trình bày một ví dụ phức tạp hơn dựa trên tập dữ liệu.

## Chương 1 Giới thiệu

Ví dụ mã sau tạo một từ điển, xóa một phần tử, thêm một phần tử, tạo danh sách các phần tử từ điển và duyệt qua danh sách:

```
if __name__ == "__main__":
    audio = {'amp':'Linn', 'preamp':'Luxman', 'loa':'Energy', 'ic':'Crystal
             Ultra', 'pc':'JPS ', 'power':'Equi-Tech', 'sp':'Crystal
             Ultra', 'cdp':'Nagra', 'up':'Esoteric'}
    del audio['up']
    print ('phần tử dict "đã xóa";') print
    (âm thanh, '\n') print
    (phần tử 'dict "đã thêm";')
    audio['up'] = 'Oppo'
    print ( âm thanh,
    '\n') print ('trình phát phô quát:', âm thanh['up'],
    '\n') dict_ls = [âm
    thanh] video = {'tv':'LG 65C7 OLED', 'stp':' DISH', 'HDMI':'DH Labs',
    'cáp' : 'dõi'}
    print ('danh sách các phần tử dict;')
    dict_ls.append(video)
    for i, row in enumerate(dict_ls):
        print ('row', i, ':')
        print (row)
```

Đầu ra:

```
dict "deleted" element;
{'amp': 'Linn', 'preamp': 'Luxman', 'speakers': 'Energy', 'ic': 'Crystal Ultra',
 'pc': 'JPS', 'power': 'Equi-Tech', 'sp': 'Crystal Ultra', 'cdp': 'Nagra'}

dict "added" element;
{'amp': 'Linn', 'preamp': 'Luxman', 'speakers': 'Energy', 'ic': 'Crystal Ultra',
 'pc': 'JPS', 'power': 'Equi-Tech', 'sp': 'Crystal Ultra', 'cdp': 'Nagra', 'up':
 'Oppo'}

universal player: Oppo

list of dict elements;
row 0 :
{'amp': 'Linn', 'preamp': 'Luxman', 'speakers': 'Energy', 'ic': 'Crystal Ultra',
 'pc': 'JPS', 'power': 'Equi-Tech', 'sp': 'Crystal Ultra', 'cdp': 'Nagra', 'up':
 'Oppo'}
row 1 :
{'tv': 'LG 65C7 OLED', 'stp': 'DISH', 'HDMI': 'DH Labs', 'cable': 'coax'}
```

## Chương 1 Giới thiệu

Khối chính bắt đầu bằng cách tạo âm thanh từ điển với một số yếu tố. Nó tiếp tục bằng cách xóa một phần tử có khóa và giá trị Bí truyền, rồi hiển thị. Tiếp theo, một phần tử mới có phím lên và phần tử Oppo được thêm lại và hiển thị. Phần tiếp theo tạo danh sách có âm thanh từ điển, tạo video từ điển và thêm từ điển mới vào danh sách. Phần cuối cùng sử dụng vòng lặp for để duyệt qua danh sách từ điển và hiển thị hai từ điển. Một chức năng rất hữu ích có thể được sử dụng với câu lệnh lặp là enumerate(). Nó thêm một bộ đếm vào một lần lặp. Iterable là một đối tượng có thể được lặp đi lặp lại. Hàm enumerate() rất hữu ích vì bộ đếm được tạo và tăng tự động, nghĩa là ít mã hơn.

## Đọc và ghi dữ liệu

Khả năng đọc và ghi dữ liệu là nền tảng cho bất kỳ nỗ lực khoa học dữ liệu nào. Tất cả các tệp dữ liệu có sẵn trên trang web. Các loại dữ liệu cơ bản nhất là văn bản và CSV (Giá trị được phân tách bằng dấu phẩy). Vì vậy, đây là nơi chúng ta sẽ bắt đầu.

Ví dụ sau đọc một tệp văn bản và dọn dẹp nó để xử lý. Sau đó, nó đọc tệp văn bản đã được làm sạch trước, lưu nó dưới dạng tệp CSV, đọc tệp CSV, chuyển đổi nó thành danh sách các phần tử OrderedDict và chuyển đổi danh sách này thành danh sách các phần tử từ điển thông thường.

### nhập csv

xác định read\_txt (f):

```
với open(f, 'r') là f:  
    d = f.readlines()  
    trả lại [x.strip() cho x trong d]
```

def conv\_csv(t, c):

```
    dữ liệu = read_txt(t)  
    với open(c, 'w', newline='') dưới dạng csv_file:
```

```

writer = csv.writer(csv_file) cho
dòng trong dữ liệu:

ls = line.split()
writer.writerow(ls)

def read_csv(f):
    ,
nội dung =
với open(f, 'r') là f:
    reader = csv.reader(f)
    danh sách trả về(reader)

def read_dict(f, h):
    input_file = csv.DictReader(open(f), tên trường=h) trả về
    input_file

def od_to_d(od):
    trả lại chính tả(od)

if __name__ == "__main__":
    f =
        'data/names.txt' data =
            read_txt(f) print
            ('mẫu dữ liệu tệp văn bản:') for i,
            row in enumerate(data):
                nếu i <
                    3: in (hàng)
    csv_f = 'data/names.csv'
    conv_csv(f, csv_f)
    r_csv = read_csv(csv_f)
    print ('\ntext to csv sample:') for
    i, row in enumerate(r_csv): if i <
        3: print
            (row ) tiêu
    đè = ['đầu tiên', 'cuối cùng']

```

## Chương 1 Giới thiệu

```
r_dict = read_dict(csv_f, tiêu đề)
dict_ls = []
print ('\ncsv cho mẫu dict đã đặt hàng:')
đối với tôi, hàng trong liệt kê (r_dict):
    r = od_to_d(hàng)
    dict_ls.append(r)
    nếu tôi < 3:
        in (hàng)
print ('\ndanh sách mẫu phần tử từ điển:')
đối với tôi, hàng trong liệt kê (dict_ls):
    nếu tôi < 3:
        in (hàng)
```

Đầu ra:

```
text file data sample:
Adam Baum
Adam Zapel
Al Bino

text to csv sample:
['Adam', 'Baum']
['Adam', 'Zapel']
['Al', 'Bino']

csv to ordered dict sample:
OrderedDict([('first', 'Adam'), ('last', 'Baum')])
OrderedDict([('first', 'Adam'), ('last', 'Zapel')])
OrderedDict([('first', 'Al'), ('last', 'Bino')])

list of dictionary elements sample:
{'first': 'Adam', 'last': 'Baum'}
{'first': 'Adam', 'last': 'Zapel'}
{'first': 'Al', 'last': 'Bino'}
```

Mã bắt đầu bằng cách nhập thư viện csv, thực hiện các lớp để đọc và ghi dữ liệu dạng bảng ở định dạng CSV. Nó tiếp tục với năm chức năng. Hàm read\_txt() đọc tệp văn bản (.txt) và loại bỏ (xóa) các ký tự không liên quan bằng khả năng hiểu danh sách, đây là một cách hay

## Chương 1 Giới thiệu

để xác định và tạo một danh sách trong Python. Việc hiểu danh sách sẽ được đề cập sau trong phần tiếp theo. Hàm `conv_csv()` chuyển đổi văn bản thành tệp CSV và lưu vào đĩa. Hàm `read_csv()` đọc tệp CSV và trả về dưới dạng danh sách. Hàm `read_dict()` đọc tệp CSV và trả về danh sách các phần tử `OrderedDict`. `OrderedDict` là một lớp con từ điển ghi nhớ thứ tự mà nội dung của nó được thêm vào, trong khi một từ điển thông thường không theo dõi thứ tự chèn. Cuối cùng, hàm `od_to_d()` chuyển đổi phần tử `OrderedDict` thành phần tử từ điển thông thường. Theo tôi, làm việc với một phần tử từ điển thông thường trực quan hơn nhiều. Khởi chính bắt đầu bằng cách đọc tệp văn bản và làm sạch tệp để xử lý. Tuy nhiên, không có quá trình xử lý nào được thực hiện với tệp đã được làm sạch này trong mã. Nó chỉ được đưa vào trong trường hợp bạn muốn biết cách hoàn thành nhiệm vụ này. Mã tiếp tục bằng cách chuyển đổi tệp văn bản thành CSV, được lưu vào đĩa. Sau đó, tệp CSV được đọc từ đĩa và một vài bản ghi được hiển thị. Tiếp theo, một danh sách tiêu đề được tạo để lưu trữ các khóa cho từ điển chưa được tạo. Danh sách `dict_ls` được tạo để chứa các phần tử từ điển. Mã tiếp tục bằng cách tạo một danh sách `OrderedDict r_dict`. Danh sách `OrderedDict` sau đó được lặp lại để mỗi phần tử có thể được chuyển đổi thành một phần tử từ điển thông thường và được thêm vào `dict_ls`. Một vài bản ghi được hiển thị trong quá trình lặp lại. Cuối cùng, `dict_ls` được lặp lại và một vài bản ghi được hiển thị. Tôi thực sự khuyên bạn nên dành chút thời gian để tự làm quen với các cấu trúc dữ liệu này vì chúng được sử dụng rộng rãi trong ứng dụng khoa học dữ liệu.

## Danh sách hiểu

Khả năng hiểu danh sách cung cấp một cách ngắn gọn để tạo danh sách. Logic của nó được đặt trong dấu ngoặc vuông chứa một biểu thức theo sau bởi mệnh đề `for` và có thể được bổ sung thêm bởi mệnh đề `for` hoặc `if`.

Hàm `read_txt()` trong phần trước bao gồm khả năng hiểu danh sách sau:

```
[x.strip() cho x trong d]
```

## Chương 1 Giới thiệu

Logic loại bỏ các ký tự không liên quan khỏi chuỗi trong iterable d. trong này trường hợp, d là danh sách các chuỗi.

Ví dụ mã sau đây chuyển đổi dặm thành km, điều khiển vật nuôi và tính toán tiền thường bằng khả năng hiểu danh sách:

```

nếu __name__ == "__main__":
    dặm =
        [100, 10, 9,5, 1000, 30] kilômét = [x *
            1,60934 cho x tính bằng dặm] print ('dặm sang
            kilômét:') for i, row in
            enumerate(kilômét) : print ('{:>4} {:>8}{:>8}
            {:>2}'. format(dặm[i], 'dặm là',
            vòng(hàng,2), 'km')) print ('\npet:') pet = ['mèo', 'chó',
            'thỏ', 'vẹt', 'chuột
            lang', 'cá'] print (thú cưng) print ('\npets:') pet = [x + 's' if x != 'fish'
            other x for x
            in pet] print (pets)

subset = [x for x in pet if x != 'fish' and x != 'rabbits'

và x != 'vẹt' và x != 'chuột lang'] print ('\nthú nuôi
phổ biến nhất:') print (tập con[1],
've', tập con[0]) doanh số = [9000, 20000,
50000 , 100000] print ('\nbonuses:') bonus = [0
if x < 10000 other x * .02
if x >= 10000 and x <= 20000 other x

*
    0,03 cho x khi bán hàng]

print (bonus)
print ('\nbonus dict:') people
= ['dave', 'sue', 'al', 'sukki'] d = {} for i, row
in
enumerate(people):

```

```
d[row] = bonus[i]
in (d, '\n')
in ('{:<5} {:<5}'.format('emp', 'bonus'))
cho k, y trong d.items():
    in ('{:<5} {:>6}'.format(k, y))
```

Đầu ra:

```
miles to kilometers:
100 miles is 160.93 km
10 miles is 16.09 km
9.5 miles is 15.29 km
1000 miles is 1609.34 km
30 miles is 48.28 km

pet:
['cat', 'dog', 'rabbit', 'parrot', 'guinea pig', 'fish']

pets:
['cats', 'dogs', 'rabbits', 'parrots', 'guinea pigs', 'fish']

most common pets:
dogs and cats

bonuses:
[0, 400.0, 1500.0, 3000.0]

bonus dict:
{'dave': 0, 'sue': 400.0, 'al': 1500.0, 'sukki': 3000.0}

emp    bonus
dave      0
sue     400.0
al      1500.0
sukki 3000.0
```

Khối chính bắt đầu bằng cách tạo hai danh sách-dặm và km. Các danh sách km được tạo với khả năng hiểu danh sách, nhân mỗi giá trị dặm với 1,60934. Lúc đầu, việc hiểu danh sách có vẻ khó hiểu, nhưng thực hành sẽ giúp việc này trở nên dễ dàng hơn theo thời gian. Khối chính tiếp tục bằng cách in dặm và km liên quan. Hàm format() cung cấp các tùy chọn định dạng phức tạp. Mỗi giá trị dặm là {:>4} với tối đa bốn ký tự được căn phải. Mỗi chuỗi dặm và kilômét đều được căn phải {:>8}.

## Chương 1 Giới thiệu

với tối đa tám ký tự. Cuối cùng, mỗi chuỗi cho km được căn phẩi (:>2) với tối đa hai ký tự. Điều này thoạt nghe có vẻ hơi phức tạp, nhưng nó thực sự khá hợp lý (và tao nhã) khi bạn đã quen với nó. Khối chính tiếp tục bằng cách tạo danh sách thú cưng và thú cưng. Danh sách vật nuôi được tạo với khả năng hiểu danh sách, làm cho vật nuôi trở thành số nhiều nếu nó không phải là cá. Tôi khuyên bạn nên nghiên cứu cách hiểu danh sách này trước khi tiếp tục, bởi vì chúng chỉ trở nên phức tạp hơn. Mã tiếp tục bằng cách tạo một danh sách tập hợp con với khả năng hiểu danh sách, chỉ bao gồm chó và mèo. Phần tiếp theo tạo hai danh sách–bán hàng và tiền thưởng. Phần thưởng được tạo với khả năng hiểu danh sách tính toán phần thưởng cho mỗi giá trị bán hàng. Nếu doanh số bán hàng dưới 10.000, không có tiền thưởng nào được trả. Nếu doanh số từ 10.000 đến 20.000 (bao gồm cả), tiền thưởng là 2% doanh số. Cuối cùng, nếu doanh số lớn hơn 20.000, tiền thưởng là 3% doanh số. Lúc đầu, tôi bối rối với cách hiểu danh sách này nhưng bây giờ nó có ý nghĩa với tôi. Vì vậy, hãy thử một số của riêng bạn và bạn sẽ nắm được ý chính của nó. Phần cuối cùng tạo danh sách mọi người để liên kết với từng giá trị bán hàng, tiếp tục bằng cách tạo từ điển để giữ tiền thưởng cho từng người và kết thúc bằng cách lặp lại các phần tử từ điển. Các định dạng là khá thanh lịch.

Tiêu đề bên trái minh đúng cách cho emp và bonus. Mỗi mục được định dạng sao cho người đó được căn trái với tối đa năm ký tự (:<5) và phần thưởng được căn phẩi với tối đa sáu ký tự (:>6).

## máy phát điện

Trình tạo là một loại trình lặp đặc biệt, nhưng nhanh hơn nhiều vì các giá trị chỉ được tạo khi cần thiết. Quá trình này được gọi là đánh giá lười biếng (hoặc trì hoãn). Các trình lặp điển hình chậm hơn nhiều vì chúng được tích hợp hoàn toàn vào bộ nhớ. Trong khi các hàm thông thường trả về các giá trị, các trình tạo sẽ mang lại chúng. Cách tốt nhất để duyệt và truy cập các giá trị từ trình tạo là sử dụng vòng lặp. Cuối cùng, việc hiểu danh sách có thể được chuyển đổi thành trình tạo bằng cách thay thế dấu ngoặc vuông bằng dấu ngoặc đơn.

## Chương 1 Giới thiệu

Ví dụ mã sau đọc tệp CSV và tạo danh sách các phần tử OrderedDict.

Sau đó, nó chuyển đổi các phần tử danh sách thành các phần tử từ điển thông thường. Mã tiếp tục bằng cách mô phỏng thời gian để hiểu danh sách, hiểu trình tạo và trình tạo. Trong quá trình mô phỏng, một danh sách thời gian cho mỗi lần được tạo. Mô phỏng là sự bắt chước của một quy trình hoặc hệ thống trong thế giới thực theo thời gian và nó được sử dụng rộng rãi trong khoa học dữ liệu.

nhập csv, thời gian, numpy dưới dạng np

```
def read_dict(f, h):
    input_file = csv.DictReader(open(f), tên trường=h)
    trả lại (input_file)

def conv_reg_dict(d):
    trả về [dict(x) cho x trong d]

def sim_times(d, n):
    tôi = 0
    lsd, lsgc = [], []
    trong khi tôi < n:
        bắt đầu = time.clock()
        [x cho x trong d]
        time_d = time.clock() - bắt đầu
        lsd.append(time_d)
        bắt đầu = time.clock()
        (x cho x trong d)
        time_gc = time.clock() - bắt đầu
        lsgc.append(time_gc)
        tôi += 1
    trả lại (lsd, lsgc)
```

## Chương 1 Giới thiệu

```

def gen(d):
    năng suất (x cho x trong d)

def sim_gen(d, n): i
    = 0

    lsg = []
    trình tạo = gen(d)
    trong khi i < n:
        start = time.clock() cho
        hàng trong trình tạo:
            Không có
            time_g = time.clock() - bắt đầu
            lsg.append(time_g) i
            += 1
        trình tạo = gen(d) trả
    về lsg

def avg_ls(ls):
    trả về np.mean(ls)

nếu __name__ == '__main__':
    f = 'dữ liệu/tên.csv'
    tiêu đề = ['đầu tiên', 'cuối
    cùng'] r_dict = read_dict(f, tiêu đề)
    dict_ls = conv_reg_dict(r_dict)
    n = 1000

    ls_times, gc_times = sim_times(dict_ls, n) g_times =
    sim_gen(dict_ls, n) avg_ls =
    np.mean(ls_times) avg_gc =
    np.mean(gc_times) avg_g =
    np.mean(g_times) gc_ls =
    round((avg_ls / avg_gc ), 2) g_ls =
    round((avg_ls / avg_g), 2)

```

```

print ('hiểu trình tạo:')
in (gc_ls, 'nhanh gấp nhiều lần khả năng hiểu danh sách\n')
in ('máy phát điện:')
print (g_ls, 'nhanh gấp nhiều lần khả năng hiểu danh sách')

```

Đầu ra:

```

generator comprehension:
9.46 times faster than list comprehension

generator:
9.66 times faster than list comprehension

```

Mã bắt đầu bằng cách nhập các thư viện csv, time và numpy. Chức năng `read_dict()` chuyển đổi tệp CSV (.csv) thành danh sách các phần tử `OrderedDict`. Hàm `conv_reg_dict()` chuyển đổi danh sách các phần tử `OrderedDict` thành danh sách các phần tử từ điển thông thường (để xử lý dễ dàng hơn). Hàm `sim_times()` chạy mô phỏng tạo hai danh sách-lsd và lsgc. Danh sách lsd chứa n lần chạy để nén danh sách và danh sách lsgc chứa n lần chạy để hiểu trình tạo. Sử dụng mô phỏng cung cấp bức tranh chính xác hơn về thời gian thực cần thiết cho cả hai quá trình này bằng cách chạy đi chạy lại chúng (n lần). Trong trường hợp này, mô phỏng được chạy 1.000 lần ( $n = 1000$ ). Tất nhiên, bạn có thể chạy mô phỏng nhiều hay ít lần tùy ý. Hàm `gen()` và `sim_gen()` hoạt động cùng nhau. Hàm `gen()` tạo một trình tạo. Hàm `sim_gen()` mô phỏng trình tạo n lần. Tôi phải tạo hai chức năng này vì việc tạo ra một trình tạo yêu cầu một quy trình khác với việc tạo ra một mức độ hiểu của trình tạo. Hàm `avg_ls()` trả về giá trị trung bình (trung bình) của một danh sách các số. Khối chính bắt đầu bằng cách đọc tệp CSV (tệp mà chúng ta đã tạo trước đó trong chương) thành danh sách các phần tử `OrderedDict` và chuyển đổi nó thành danh sách các phần tử từ điển thông thường. Mã tiếp tục bằng cách mô phỏng thời gian chạy của việc hiểu danh sách và hiểu trình tạo 1.000 lần (Mô phỏng đầu tiên tính toán 1.000 thời gian chạy để duyệt qua danh sách từ điển được tạo trước đó để hiểu cả danh sách và trình tạo và trả về

## Chương 1 Giới thiệu

một danh sách các thời gian chạy cho mỗi. Mô phỏng thứ 2 tính toán 1.000 thời gian chạy bằng cách duyệt qua danh sách từ điển cho trình tạo và trả về danh sách các thời gian chạy đó. Đoạn mã kết thúc bằng cách tính toán thời gian chạy trung bình cho từng kỹ thuật trong số ba kỹ thuật–hiểu danh sách, hiểu trình tạo và trình tạo–và so sánh các giá trị trung bình đó.

Các mô phỏng xác minh rằng sự hiểu biết của trình tạo nhiều hơn mười lần và trình tạo nhanh hơn tám lần so với khả năng hiểu danh sách (thời gian chạy sẽ thay đổi tùy theo PC của bạn). Điều này có ý nghĩa vì khả năng hiểu danh sách lưu trữ tất cả dữ liệu trong bộ nhớ, trong khi trình tạo đánh giá (một cách lười biếng) khi cần dữ liệu. Đường nhiên, lợi thế về tốc độ của máy phát điện trở nên quan trọng hơn với các tập dữ liệu lớn. Nếu không có mô phỏng, thời gian chạy không thể được xác minh vì chúng tôi ngẫu nhiên nhận được thời gian của đồng hồ hệ thống bên trong.

## Ngẫu nhiên hóa dữ liệu

Một quá trình ngẫu nhiên là một họ các biến ngẫu nhiên từ một số không gian xác suất vào một không gian trạng thái (whew!). Đơn giản, đó là một quá trình ngẫu nhiên xuyên thời gian. Ngẫu nhiên hóa dữ liệu là quá trình chọn các giá trị từ một mẫu theo cách không thể đoán trước với mục tiêu mô phỏng thực tế. Mô phỏng cho phép ứng dụng ngẫu nhiên hóa dữ liệu trong khoa học dữ liệu. Phần trước đã trình bày cách mô phỏng có thể được sử dụng để so sánh thực tế các lần lặp (hiểu danh sách, hiểu trình tạo và trình tạo).

Trong Python, các số giả ngẫu nhiên được sử dụng để mô phỏng tính ngẫu nhiên của dữ liệu (thực tế). Chúng không thực sự ngẫu nhiên vì thế hệ thứ nhất không có số trước đó. Chúng tôi phải cung cấp một hạt giống (hoặc hạt giống ngẫu nhiên) để khởi tạo trình tạo số giả ngẫu nhiên. Thư viện ngẫu nhiên triển khai các trình tạo số giả ngẫu nhiên cho các phân phối dữ liệu khác nhau và `random.seed()` được sử dụng để tạo số hạt giống (thế hệ thứ nhất) ban đầu.

## Chương 1 Giới thiệu

Ví dụ mã sau đây đọc tệp CSV và chuyển đổi nó thành danh sách các thành phần từ điển thông thường. Mã tiếp tục bằng cách tạo một số ngẫu nhiên được sử dụng để lấy một phần tử ngẫu nhiên từ danh sách. Tiếp theo, một trình tạo gồm ba phần tử được chọn ngẫu nhiên được tạo và hiển thị. Mã tiếp tục bằng cách hiển thị ba phần tử được xáo trộn ngẫu nhiên từ danh sách. Phần tiếp theo của mã tạo hạt ngẫu nhiên một cách xác định, có nghĩa là tất cả các số ngẫu nhiên được tạo sẽ giống nhau dựa trên hạt giống. Vì vậy, các phần tử được hiển thị sẽ luôn giống nhau trừ khi hạt giống bị thay đổi. Sau đó, mã này sử dụng thời gian của hệ thống để tạo các số ngẫu nhiên một cách không xác định và hiển thị ba phần tử đó. Tiếp theo, các số ngẫu nhiên không xác định được tạo bằng một phương pháp khác và ba phần tử đó được hiển thị. Phần cuối cùng tạo một danh sách tên để các phương pháp lấy mẫu và lựa chọn ngẫu nhiên có thể được sử dụng để hiển thị các phần tử.

nhập csv, ngẫu nhiên, thời gian

```
def read_dict(f, h):
    input_file = csv.DictReader(open(f), tên trường=h)
    trả lại (input_file)

def conv_reg_dict(d):
    trả về [dict(x) cho x trong d]

def r_inds(ls, n):
    chiều dài = len(ls) - 1
    năng suất [random.randrange(độ dài) cho _ trong phạm vi (n)]

def get_slice(ls, n):
    trả lại ls[:n]

def p_line():
    in ()
```

## Chương 1 Giới thiệu

```

if __name__ == '__main__':
    f = 'data/names.csv'

    headers = ['first', 'last']
    r_dict = read_dict(f, headers)
    dict_ls = conv_reg_dict(r_dict) n
    = len(dict_ls) r
    = random.randrange(0, n-1)
    print ('chỉ mục được chọn ngẫu nhiên:', r)
    r print ('phần tử được chọn ngẫu nhiên:', dict_ls[r])
    phần tử = 3

    trình tạo = next(r_inds(dict_ls, phần tử))

    p_line()

    () print (phần tử, 'chỉ số được tạo ngẫu nhiên:', trình tạo)
    print (phần tử, 'phần tử dựa trên chỉ số:') cho
    hàng trong trình tạo:
        print (dict_ls[row])

    x = [[i] for i in range( n-1)]
    random.shuffle(x)

    p_line()

    print ('thứ nhất', phần tử, 'phần tử được xáo
    trộn:') ind = get_slice(x,
    phần tử) cho hàng trong ind:
        in (dict_ls[row[0]]) hạt
    giống = 1

    Random_seed = Random.seed(seed)
    rs1 = Random.randrange(0, n-1)

    p_line()

    print ('hạt giống xác định', str(seed) + ':', rs1)
    print ('phần tử tương ứng:', dict_ls [rs1]) t =
    time.time()
    random_seed = random.seed(t)

```

## Chương 1 Giới thiệu

```
rs2 = random.randrange(0, n-1)
p_line()
print ('hạt thời gian không xác định', str(t) + ' index:', rs2)
print ('phần tử tương ứng:', dict_ls[rs2], '\n') print
(phần tử, 'phần tử ngẫu nhiên được gieo theo thời gian:')
for i in range(phần tử):
    r = Random.randint(0, n-1)
    print (dict_ls[r], r)
Random.seed = Random.seed()
rs3 = Random.randrange(0, n-1)
p_line()
print ('tự động không xác định hạt giống:', rs3) print ('phần tử tương ứng:', dict_ls[rs3], '\n')
print (phần tử, 'phần tử ngẫu nhiên tự động gieo:')
cho tôi trong phạm vi (phần tử):
    r = random.randint(0, n-1)
    print (dict_ls[r], r)
tên = []
cho hàng trong
dict_ls: tên = hàng['cuối'] + ', ' +
    hàng['đầu tiên']

tên.append(name) p_line() print (elements, 'names
with "random.choice()":') cho hàng trong phạm vi(elements):
    print (random.choice(names))

p_line()
print (elements, 'names with "random.sample()":')
print (random.sample(names, elements))
```

## Chương 1 Giới thiệu

Đầu ra:

```

randomly selected index: 85
randomly selected element: {'first': 'Heidi', 'last': 'Clare'}

3 randomly generated indicies: [10, 77, 136]
3 elements based on indicies:
({'first': 'Amanda', 'last': 'Lynn'}
 {'first': 'Eaton', 'last': 'Wright'}
 {'first': 'Rich', 'last': 'Mann'})

1st 3 shuffled elements:
({'first': 'Gene', 'last': 'Poole'}
 {'first': 'Marty', 'last': 'Graw'}
 {'first': 'Wanda', 'last': 'Rinn'})

deterministic seed 1: 34
corresponding element: {'first': 'April', 'last': 'Schauer'}

non-deterministic time seed 1512777603.6807067 index: 18
corresponding element: {'first': 'Anita', 'last': 'Job'}

3 random elements seeded with time:
({'first': 'Jay', 'last': 'Walker'}) 96
({'first': 'Dick', 'last': 'Tator'}) 70
({'first': 'Anita', 'last': 'Schhauer'}) 23

non-deterministic auto seed: 127
corresponding element: {'first': 'Olive', 'last': 'Hoyl'}

3 random elements auto seed:
({'first': 'Royal', 'last': 'Payne'}) 142
({'first': 'Harry', 'last': 'Legg'}) 84
({'first': 'Ty', 'last': 'Knotts'}) 161

3 names with "random.choice()":
Beard, Harry
Carr, Dusty
Gaiter, Ali

3 names with "random.sample()":
['Friesel, Andy', 'Cade, Barry', 'Walker, Jay']

```

Mã bắt đầu bằng cách nhập các thư viện csv, ngẫu nhiên và thời gian.

Các chức năng `read_dict()` và `conv_reg_dict()` đã được giải thích.

Hàm `r_inds()` tạo danh sách n phần tử ngẫu nhiên từ danh sách từ điển.

Để có được độ dài phù hợp, một người bị trừ đi vì Python

danh sách bắt đầu tại chỉ số không. Hàm `get_slice()` tạo danh sách n phần tử được xáo trộn ngẫu nhiên từ danh sách từ điển. Hàm `p_line()` in ra một dòng trống. Khối chính bắt đầu bằng cách đọc tệp CSV và chuyển đổi nó thành danh sách các phần tử từ điển thông thường. Mã tiếp tục bằng cách tạo một số ngẫu nhiên với `random.randrange()` dựa trên số chỉ mục từ danh sách từ điển và hiển thị chỉ mục cũng như phần tử từ điển liên quan. Tiếp theo, một trình tạo được tạo và phổ biến với ba phần tử được xác định ngẫu nhiên. Các chỉ số và các yếu tố liên quan được in từ trình tạo. Phần tiếp theo của mã xáo trộn ngẫu nhiên các chỉ số và đặt chúng vào danh sách x. Một giá trị chỉ mục được tạo bằng cách cắt ba phần tử ngẫu nhiên dựa trên các chỉ số đã xáo trộn được lưu trữ trong danh sách x.

Ba yếu tố sau đó được hiển thị. Mã tiếp tục bằng cách tạo một hạt giống ngẫu nhiên xác định bằng cách sử dụng một số cố định (hạt giống) trong hàm. Vì vậy, số ngẫu nhiên được tạo bởi hạt giống này sẽ giống nhau mỗi khi chương trình được chạy. Điều này có nghĩa là phần tử từ điển được hiển thị cũng sẽ giống nhau. Tiếp theo, hai phương pháp tạo số ngẫu nhiên không xác định được trình bày `random.seed(t)` và `random.seed()`

trong đó t thay đổi theo thời gian hệ thống và không sử dụng tham số nào sẽ tự động thay đổi các số ngẫu nhiên. Các yếu tố được tạo ngẫu nhiên được hiển thị cho từng phương pháp. Phần cuối cùng của mã tạo một danh sách các tên chỉ chứa họ và tên, vì vậy có thể sử dụng `Random.choice()` và `Random.sample()`.

## MongoDB và JSON

MongoDB là một cơ sở dữ liệu dựa trên tài liệu được phân loại là NoSQL. NoSQL (Không chỉ cơ sở dữ liệu SQL) là một cách tiếp cận thiết kế cơ sở dữ liệu có thể chứa nhiều loại mô hình dữ liệu, bao gồm các định dạng khóa-giá trị, tài liệu, cột và biểu đồ. Nó sử dụng các tài liệu giống như JSON với các lược đồ. Nó tích hợp rất tốt với Python. Một bộ sưu tập MongoDB về mặt khái niệm giống như một bảng trong cơ sở dữ liệu quan hệ và

## Chương 1 Giới thiệu

một tài liệu về mặt khái niệm giống như một hàng. JSON là một định dạng trao đổi dữ liệu nhẹ giúp con người dễ dàng đọc và viết. Máy cũng dễ dàng phân tích cú pháp và tạo.

Truy vấn cơ sở dữ liệu từ MongoDB được xử lý bởi PyMongo. PyMongo là một bản phân phối Python chứa các công cụ để làm việc với MongoDB. Đây là công cụ hiệu quả nhất để làm việc với MongoDB bằng các tiện ích của Python. PyMongo được tạo ra để tận dụng những lợi thế của Python dưới dạng ngôn ngữ lập trình và MongoDB dưới dạng cơ sở dữ liệu. Thư viện pymongo là trình điều khiển riêng cho MongoDB, có nghĩa là nó được tích hợp vào ngôn ngữ Python. Vì nó là bản địa, nên thư viện pymongo sẽ tự động có sẵn (không cần phải nhập vào mã).

Ví dụ mã sau đây đọc tệp CSV và chuyển đổi nó thành danh sách các thành phần từ điển thông thường. Mã tiếp tục bằng cách tạo một tệp JSON từ danh sách từ điển và lưu nó vào đĩa. Tiếp theo, mã kết nối với MongoDB và chèn dữ liệu JSON. Phần cuối cùng của mã thao tác dữ liệu từ cơ sở dữ liệu MongoDB. Đầu tiên, tất cả dữ liệu trong cơ sở dữ liệu được truy vấn và một vài bản ghi được hiển thị. Thứ hai, cơ sở dữ liệu được tua lại. Tua lại đặt con trỏ quay lại bản ghi cơ sở dữ liệu đầu tiên. Cuối cùng, các truy vấn khác nhau được thực hiện.

```
nhập json, csv, sys, os
sys.path.append(os.getcwd()+'/classes')
kết nối nhập khẩu

def read_dict(f, h):
    input_file = csv.DictReader(open(f), tên trường=h)
    trả lại (input_file)

def conv_reg_dict(d):
    trả về [dict(x) cho x trong d]
```

```
def dump_json(f, d):
    với open(f, 'w') là f:
        json.dump(d, f)

def read_json(f):
    với open(f) là f:
        return json.load(f)

if __name__ == '__main__':
    f = 'data/names.csv'

    headers = ['first', 'last']
    r_dict = read_dict(f, headers)
    dict_ls = conv_reg_dict(r_dict)
    json_file = 'data/names.json'
    dữ liệu dump_json(json_file, dict_ls)
    obj = read_json(json_file)
    conn.conn('test')
    db = obj.getDB()
    name = db.names

    tên.drop()
    cho tôi, hàng trong liệt kê (dữ liệu): hàng
    ['_id'] = i.name.insert_one( hàng )
    n = 3
    print('1st', n, 'names:')
    people = names.find()
    for i, row in enumerate(people):
        if i < n:
            print(row)
    people.rewind()
    print('\n1st', n, 'names with rewind:')
    for i, row in enumerate(people):
```

## Chương 1 Giới thiệu

```

if i <
    n: print
(row) print ('\nquery 1st', n,
'names') first_n =
name.find().limit(n)
    cho hàng
trong first_n: print (row) print
 ('\nquery last ', n, 'names')
length = name.find().count() last_n =
    name.find().skip(length
- n) cho hàng trong last_n:
print (row) fnames =
['Ella', ' Lou'] lnames = ['Vader', 'Pole'] print ('\nquery
Ella:') query_1st_in_list =
    name.find( {'first':{$in':
[fnames[0]]}}) for hàng trong query_1st_in_list:
print (row) print
    ('\nquery
Ella or Lou:') query_1st =
names.find( {'first':{$in':fnames}} ) cho hàng trong query_1st:
print (row) print
    ('\nquery
Lou Pole:') query_and = name.find( {'first':fnames[1],
'last':lnames[1]} ) cho hàng trong query_and: print
(row) print ('\nquery first tên Ella hoặc họ Cực:') query_or = name.find(

```

```

    cho hàng trong
        query_or:
    print (row)
pattern = '^Sch' print ('\nquery
regex pattern:') query_like = name.find( {'last':
{ '$regex':pattern}} )
    cho hàng
trong query_like :
print (row) pid = name.count() doc = {'_id':pid,
'first':'Wendy',
'last':'Day'} name.insert_one(doc)
print ('\nnhiều thị tài liệu đã thêm :')
q_added =
name.find({'first':'Wendy'}) print
(q_added.next()) print ('\nquery n
    _ tài liệu
cuối cùng:') q_n = name.find().skip((pid -n)+1) trong phạm vi (n): để in

```

Kết nối lớp:

kết nối lớp:

```

từ pymongo nhập MongoClient
client = MongoClient('localhost', port=27017)
def __init__(self, dbname):
    self.db = conn.client[dbname]
def getDB(self):
    return self.db

```

## Chương 1 Giới thiệu

Đầu ra:

```

lst 3 names:
{'_id': 0, 'first': 'Adam', 'last': 'Baum'}
{'_id': 1, 'first': 'Adam', 'last': 'Zapel'}
{'_id': 2, 'first': 'Al', 'last': 'Bino'}

lst 3 names with rewind:
{'_id': 0, 'first': 'Adam', 'last': 'Baum'}
{'_id': 1, 'first': 'Adam', 'last': 'Zapel'}
{'_id': 2, 'first': 'Al', 'last': 'Bino'}

query lst 3 names
{'_id': 0, 'first': 'Adam', 'last': 'Baum'}
{'_id': 1, 'first': 'Adam', 'last': 'Zapel'}
{'_id': 2, 'first': 'Al', 'last': 'Bino'}

query last 3 names
{'_id': 163, 'first': 'Will', 'last': 'Power'}
{'_id': 164, 'first': 'Willie', 'last': 'Waite'}
{'_id': 165, 'first': 'Willie', 'last': 'Makeit'}

query Ella:
{'_id': 79, 'first': 'Ella', 'last': 'Vader'}

query Ella or Lou:
{'_id': 79, 'first': 'Ella', 'last': 'Vader'}
{'_id': 108, 'first': 'Lou', 'last': 'Pole'}

query Lou Pole:
{'_id': 108, 'first': 'Lou', 'last': 'Pole'}

query first name Ella or last name Pole:
{'_id': 79, 'first': 'Ella', 'last': 'Vader'}
{'_id': 108, 'first': 'Lou', 'last': 'Pole'}

query regex pattern:
{'_id': 23, 'first': 'Anita', 'last': 'Schhauer'}
{'_id': 34, 'first': 'April', 'last': 'Schauer'}

display added document:
{'_id': 166, 'first': 'Wendy', 'last': 'Day'}

query last n documents:
{'_id': 164, 'first': 'Willie', 'last': 'Waite'}
{'_id': 165, 'first': 'Willie', 'last': 'Makeit'}
{'_id': 166, 'first': 'Wendy', 'last': 'Day'}

```

## Chương 1 Giới thiệu

Mã bắt đầu bằng cách nhập các thư viện json, csv, sys và os. Tiếp theo, một đường dẫn (sys.path.append) đến kết nối lớp được thiết lập. Phương thức getcwd() (từ thư viện os) lấy thư mục làm việc hiện tại cho các lớp. Lớp conn sau đó được nhập. Tôi đã xây dựng lớp này để đơn giản hóa việc kết nối với cơ sở dữ liệu từ bất kỳ chương trình nào. Mã tiếp tục với bốn chức năng. Các chức năng read\_dict() và conv\_reg\_dict() đã được giải thích trước đó. Hàm dump\_json() ghi dữ liệu JSON vào đĩa. Hàm read\_json() đọc dữ liệu JSON từ đĩa. Khối chính bắt đầu bằng cách đọc tệp CSV và chuyển đổi nó thành danh sách các phần tử từ điển thông thường. Tiếp theo, danh sách được kết xuất vào đĩa dưới dạng JSON. Mã tiếp tục bằng cách tạo thử nghiệm phiên bản kết nối PyMongo làm đối tượng và gán nó cho biến obj. Bạn có thể tạo bất kỳ trường hợp nào bạn muốn, nhưng thử nghiệm là mặc định. Tiếp theo, phiên bản cơ sở dữ liệu được gán cho db theo phương thức getDB() từ obj. Tên bộ sưu tập sau đó được tạo trong MongoDB và được gán cho tên biến. Khi tạo mẫu, tôi luôn bỏ bộ sưu tập trước khi thao tác với nó. Điều này giúp loại bỏ các lỗi chính trùng lặp. Mã tiếp tục bằng cách chèn dữ liệu JSON vào bộ sưu tập. Đôi với mỗi tài liệu trong bộ sưu tập MongoDB, tôi tạo rõ ràng các giá trị khóa chính bằng cách gán các số thứ tự cho \_id. MongoDB chỉ sử dụng \_id làm mã định danh khóa chính cho từng tài liệu trong bộ sưu tập. Nếu bạn không tự đặt tên cho nó, một mã định danh hệ thống sẽ tự động được tạo, theo ý kiến của tôi thì rất lộn xộn. Đoạn mã tiếp tục với truy vấn PyMongo names.find(), truy vấn này truy xuất tất cả tài liệu từ bộ sưu tập tên. Ba bản ghi được hiển thị chỉ để xác minh rằng truy vấn đang hoạt động. Để sử dụng lại một truy vấn đã được truy cập, phải thực hiện tua lại(). Truy vấn PyMongo tiếp theo truy cập và hiển thị ba (n = 3) tài liệu. Truy vấn tiếp theo truy cập và hiển thị ba tài liệu cuối cùng. Tiếp theo, chúng tôi chuyển sang các truy vấn phức tạp hơn.

Đầu tiên, truy cập tài liệu với tên Ella. Thứ hai, truy cập các tài liệu có tên Ella hoặc Lou. Thứ ba, truy cập tài liệu Lou Pole. Thứ tư, truy cập các tài liệu có tên Ella hoặc họ Pole. Tiếp theo, một biểu thức chính quy được sử dụng để truy cập tài liệu có họ bắt đầu bằng

## Chương 1 Giới thiệu

Sch. Biểu thức chính quy là một chuỗi các ký tự xác định mẫu tìm kiếm.

Cuối cùng, thêm một tài liệu mới, hiển thị nó và hiển thị ba tài liệu cuối cùng tài liệu trong bộ sưu tập.

## Hình dung

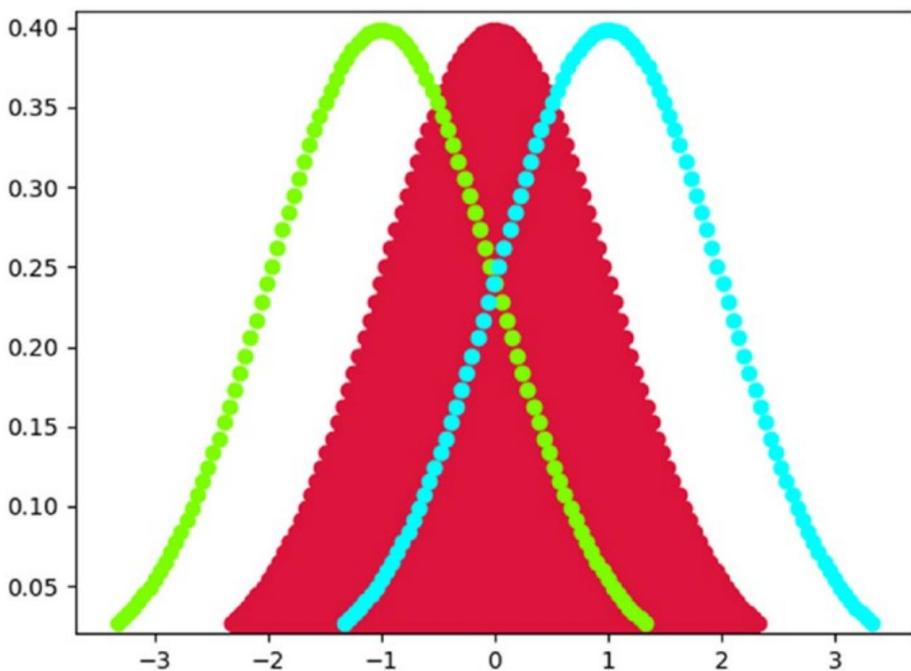
Trục quan hóa là quá trình biểu diễn dữ liệu bằng đồ họa và tận dụng các biểu diễn này để hiểu sâu hơn về dữ liệu. Trục quan hóa là một trong những kỹ năng quan trọng nhất trong khoa học dữ liệu vì nó tạo điều kiện thuận lợi cho cách chúng ta xử lý một lượng lớn dữ liệu phức tạp.

Ví dụ mã sau đây tạo và vẽ một tập hợp dữ liệu được phân phối bình thường. Sau đó, nó dịch chuyển dữ liệu sang trái (và đồ thị) và dịch chuyển dữ liệu sang phải (và đồ thị). Phân phối chuẩn là phân phối xác suất đối xứng về giá trị trung bình và rất quan trọng đối với khoa học dữ liệu vì đây là mô hình tuyệt vời về cách các sự kiện xảy ra tự nhiên trong thực tế.

```
nhập matplotlib.pyplot dưới dạng plt
từ định mức nhập scipy.stats
nhập numpy dưới dạng np

nếu __name__ == '__main__':
    x = np.linspace(norm.ppf(0,01), Norm.ppf(0,99), num=100)
    x_left = x - 1
    x_right = x + 1
    y = định mức.pdf(x)
    plt.ylim(0,02, 0,41)
    plt.scatter(x, y, color='đỏ thẫm')
    plt.fill_between(x, y, color='đỏ thẫm')
    plt.scatter(x_left, y, color='chartreuse')
    plt.scatter(x_right, y, color='cyan')
    plt.show()
```

Đầu ra:



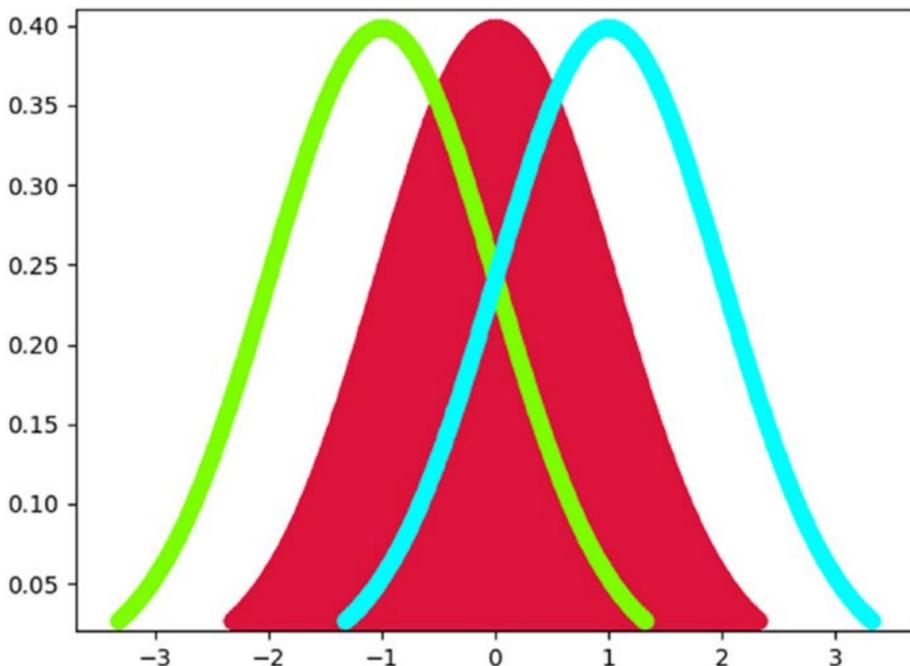
Hình 1-1. Dữ liệu phân phối thông thường

Ví dụ mã (Hình 1-1) bắt đầu bằng cách nhập các thư viện `matplotlib`, `scipy` và `numpy`. Thư viện `matplotlib` là một mô-đun vẽ sơ đồ 2-D tạo ra các số liệu chất lượng xuất bản ở nhiều định dạng bản cứng và môi trường tương tác trên các nền tảng. Thư viện SciPy cung cấp các định tuyến số hiệu quả và thân thiện với người dùng để tích hợp và tối ưu hóa số. Khối chính bắt đầu bằng cách tạo một chuỗi 100 số giữa 0,01 và 0,99. Lý do là phân phối bình thường là dựa trên xác suất, phải nằm trong khoảng từ 0 đến 1. Mã tiếp tục bằng cách dịch chuyển chuỗi một đơn vị sang trái và một đơn vị sang phải để vẽ đồ thị sau này. Phương thức `ymin()` được sử dụng để kéo biểu đồ xuống dưới cùng (trục x). Biểu đồ phân tán được tạo cho dữ liệu gốc, một đơn vị ở bên trái và một đơn vị ở bên phải, với các màu khác nhau để tạo hiệu ứng.

## Chương 1 Giới thiệu

Trên dòng đầu tiên của khói chính trong hàm `linespace()`, hãy tăng số lượng điểm dữ liệu từ `num = 100` đến `num = 1000` và xem điều gì sẽ xảy ra. Kết quả là sự phân bố bình thường trở nên trơn tru hơn, bởi vì nhiều dữ liệu hơn cung cấp một bức tranh chân thực hơn về thế giới tự nhiên.

Đầu ra:



Hình 1-2. Làm mịn dữ liệu phân phối bình thường

Làm mịn hoạt động (Hình 1-2) vì phân phối chuẩn bao gồm của biến ngẫu nhiên liên tục. Biến ngẫu nhiên liên tục là một biến ngẫu nhiên với một tập hợp các giá trị vô hạn và không đếm được. Vì vậy, nhiều dữ liệu hơn tạo ra chủ nghĩa hiện thực mang tính dự đoán hơn. Vì chúng tôi không thể thêm dữ liệu vô hạn nên chúng tôi làm việc với càng nhiều dữ liệu càng tốt. Sự đánh đổi là nhiều dữ liệu hơn tăng lên tài nguyên xử lý của máy tính và thời gian thực hiện. Do đó, các nhà khoa học dữ liệu phải cân nhắc sự đánh đổi này khi tiến hành nghề của họ.

## CHƯƠNG 2

# Monte Carlo Mô phỏng và Hàm mật độ

Mô phỏng Monte Carlo (MCS) áp dụng lấy mẫu ngẫu nhiên lặp đi lặp lại (tính ngẫu nhiên) để thu được kết quả bằng số cho việc giải bài toán xác định. Nó được sử dụng rộng rãi trong tối ưu hóa, tích hợp số và ra quyết định dựa trên rủi ro. Hàm mật độ xác suất và tích lũy là các phép đo thống kê áp dụng phân phối xác suất cho các biến ngẫu nhiên và có thể được sử dụng cùng với MCS để giải bài toán xác định.

---

Lưu ý Bạn đọc có thể tham khảo file mã nguồn tải về để xem các hình màu trong chương này.

---

### Mô phỏng chứng khoán

Ví dụ đầu tiên là giả thuyết và đơn giản, nhưng hữu ích trong việc chứng minh sự ngẫu nhiên hóa dữ liệu. Nó bắt đầu với một cổ phiếu hú cáu có giá 20 đô la. Sau đó, nó dự đoán giá trong 200 ngày và các lô.

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

nhập matplotlib.pyplot dưới dạng plt, numpy dưới dạng np  
từ số liệu thống kê nhập scipy

```
def cum_price(p, d, m, s): data
    = [] for d
    in range(d): prob =
        stats.norm.rvs(loc=m, scale=s) price = (p *
        prob) data.append (giá)
    p = dữ liệu trả về giá
```

```
nếu __name__ == "__main__":
    stk_price, days, mean, s = 20, 200, 1,001, 0,005 data =
    cum_price(stk_price, days, mean, s) plt.plot(data,
    color='lime') plt.ylabel('Price')
    plt.xlabel ('ngày')
    plt.title('giá đóng
    cửa cổ phiếu') plt.show()
```

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

Đầu ra:



Hình 2-1. Cốt truyện ngẫu nhiên đơn giản

Mã bắt đầu bằng cách nhập các thư viện `matplotlib`, `numpy` và `scipy`.  
 Nó tiếp tục với hàm `cum_price()`, tạo ra 200 số ngẫu nhiên được phân  
 phối bình thường (một số cho mỗi ngày) với `Norm_rvs()`. Dữ liệu ngẫu  
 nhiên là chìa khóa. Khối chính tạo ra các biến. Giá trị trung bình được  
 đặt cao hơn 1 một chút và (các) độ lệch chuẩn ở một số rất nhỏ để tạo ra  
 giá cổ phiếu tăng chậm. `Mean (mu)` là sự thay đổi trung bình về giá trị.  
 Độ lệch chuẩn là sự thay đổi hoặc phân tán trong dữ liệu. Với `s` là  
 0,005, dữ liệu của chúng tôi có rất ít biến thể. Tức là các số trong tập dữ  
 liệu của chúng tôi rất gần nhau. Hãy nhớ rằng đây không phải là một kịch  
 bản có thật! Mã tiếp tục bằng cách vẽ kết quả như trong Hình 2-1.

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

Ví dụ tiếp theo thêm MCS vào hỗn hợp với vòng lặp while lặp lại 100 lần:

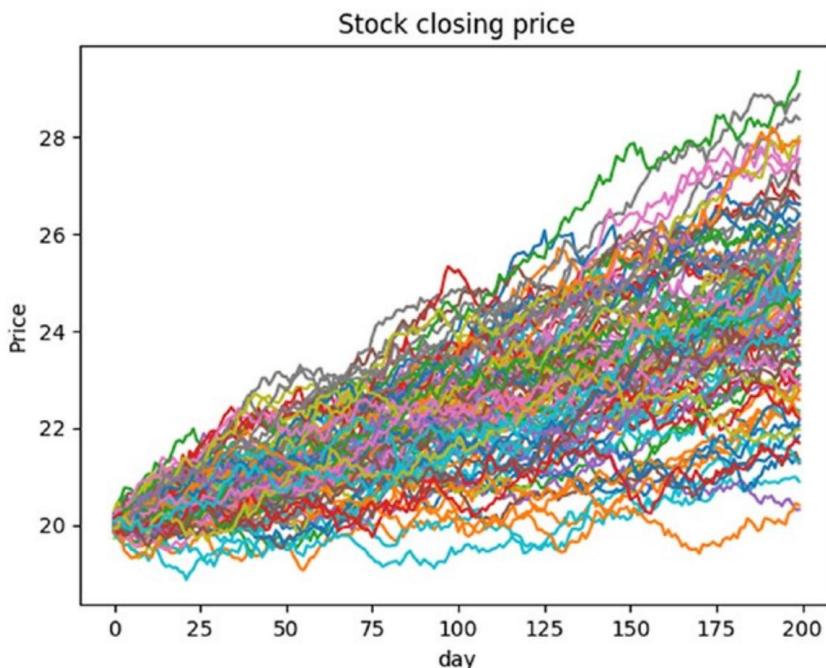
nhập matplotlib.pyplot dưới dạng plt, numpy dưới dạng np  
từ số liệu thống kê nhập scipy

```
def cum_price(p, d, m, s): data
    = [] for d
    in range(d): prob =
        stats.norm.rvs(loc=m, scale=s) price = (p *
        prob) data.append (giá)
    p = dữ liệu trả về giá
```

```
nếu __name__ == "__main__":
    stk_price, ngày, mu, sigma = 20, 200, 1,001, 0,005
    x = 0
    trong khi x < 100:
        dữ liệu = cum_price(stk_price, ngày, mu, sigma)
        plt.plot(data)
        x += 1
    plt.ylabel('Giá')
    plt.xlabel('ngày')
    plt.title('Giá đóng cửa cổ phiếu')
    plt.show()
```

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

Đầu ra:



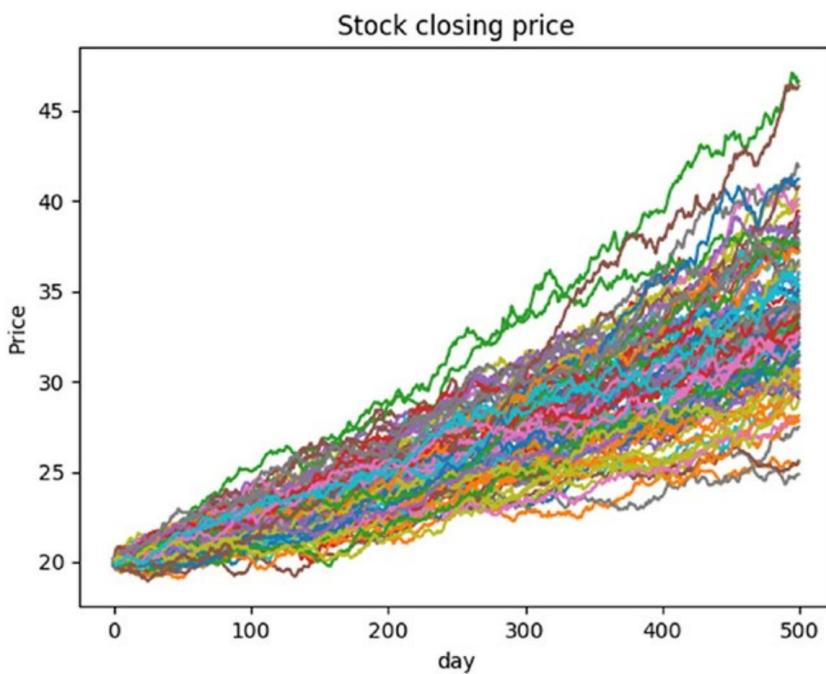
Hình 2-2. Âm mưu tăng cường mô phỏng Monte Carlo

Vòng lặp while cho phép chúng ta hình dung (như trong Hình 2-2) 100 kết quả giá cổ phiếu có thể xảy ra trong 200 ngày. Lưu ý rằng mu (trung bình) và sigma (độ lệch chuẩn) được sử dụng. Ví dụ này cho thấy sức mạnh của MCS đối với việc ra quyết định.

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

**Những gì nếu phân tích**

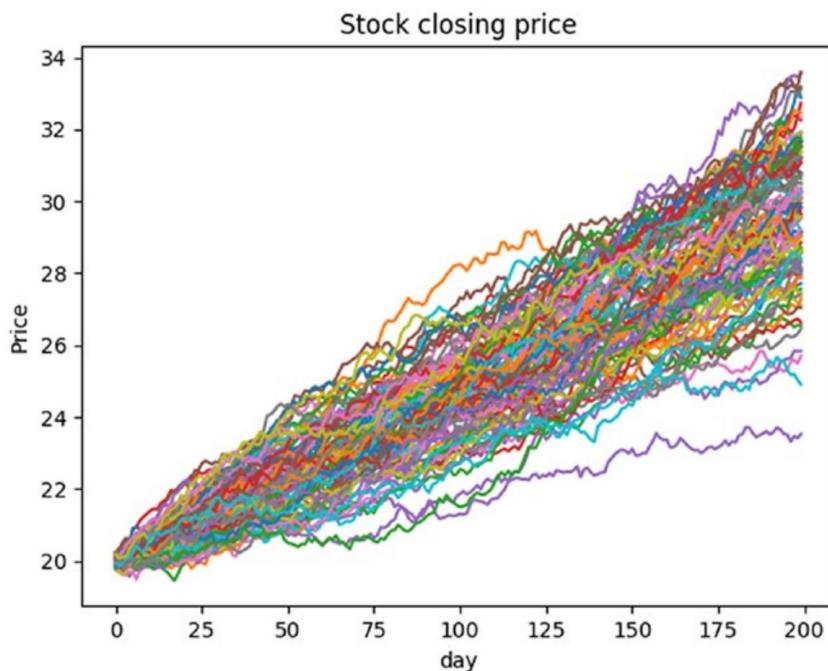
Phân tích What-If thay đổi các giá trị trong một thuật toán để xem chúng ảnh hưởng đến kết quả như thế nào. Đảm bảo mỗi lần chỉ thay đổi một biến, nếu không bạn sẽ không biết nguyên nhân gây ra thay đổi. Trong ví dụ trước, điều gì sẽ xảy ra nếu chúng ta thay đổi ngày thành 500 trong khi vẫn giữ nguyên các giá trị khác (giống nhau)? Về cơ bản thay đổi này dẫn đến kết quả như sau (Hình 2-3):



Hình 2-3. Phân tích What-If trong 500 ngày

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

Lưu ý rằng sự thay đổi về giá chậm hơn. Thay đổi mu (trung bình) thành 1,002 (đừng quên thay đổi ngày trở lại 200) dẫn đến thay đổi nhanh hơn (trung bình lớn hơn) như sau (Hình 2-4):



Hình 2-4. Phân tích What-If cho  $\mu = 1,002$

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

Thay đổi sigma thành 0,02 dẫn đến nhiều biến thể hơn như sau (Hình 2-5):



Hình 2-5. Phân tích What-If cho sigma = 0,02

### Mô phỏng nhu cầu sản phẩm

Xác suất rời rạc là xác suất của mỗi giá trị ngẫu nhiên rời rạc xuất hiện trong một không gian mẫu hoặc quần thể. Một biến ngẫu nhiên giả định các giá trị khác nhau được xác định một cách tình cờ. Biến ngẫu nhiên rời rạc chỉ có thể nhận một số giá trị đếm được. Ngược lại, biến ngẫu nhiên liên tục có thể nhận vô số giá trị trên một dòng khoảng thời gian như một phân phối bình thường.

Trong ví dụ mă, nhu cầu về một sản phẩm hư cấu được dự đoán bởi bốn kết quả xác suất riêng biệt: 10% biến ngẫu nhiên đó là 10.000 đơn vị, 35% biến ngẫu nhiên đó là 20.000 đơn vị, 30% biến ngẫu nhiên đó là 40.000 đơn vị và 25% biến ngẫu nhiên đó là biến là 60.000 đơn vị. Đơn giản,

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

10% nhu cầu thời gian là 10.000, 35% nhu cầu thời gian là 20.000, 30% nhu cầu thời gian là 40.000 và 25% nhu cầu thời gian là 60.000.

Kết quả rời rạc phải tổng 100%. Mã này chạy MCS trên thuật toán sản xuất xác định lợi nhuận cho từng kết quả riêng biệt và vẽ biểu đồ kết quả.

nhập matplotlib.pyplot dưới dạng plt, numpy dưới dạng np

```
def nhu_cầu():
```

```
    p = np.random.uniform(0,1) nếu p
```

```
< 0,10:
```

```
        trả lại 10000
```

```
elif p >= 0,10 và p < 0,45: trả về  
20000
```

```
elif p >= 0,45 và p < 0,75: trả về  
40000
```

khác:

```
        trả lại 60000
```

```
def sản_xuất(nhu_cầu, đơn_vị, giá, unit_cost, thải_bỏ):
```

```
    units_sold = tối_thiếu(dơn_vị, nhu_cầu)
```

```
    doanh_thu = units_sold * giá
```

```
    tổng_chi_phí = đơn_vị * unit_cost
```

```
    units_not_sold = đơn_vị - nhu_cầu nếu
```

```
    units_not_sold > 0: thải
```

```
        bỏ_chi_phí = thải_bỏ * đơn_vị_không_bán khác:
```

```
        xử_lý_chi_phí = 0 lợi
```

```
    nhuận = doanh_thu - tổng_chi_phí - xử_lý_chi_phí trả_lại
```

lợi nhuận

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

```

def mcs(x, n, đơn vị, giá, chi phí đơn vị, xử lý): lời nhuận =
    [] trong khi
    x <= n:
        d = nhu cầu()
        v = sản xuất(d, đơn vị, giá, chi phí đơn vị, xử lý) lời nhuận.append(v)

        x += 1

    trả lại lời nhuận

def max_bar(ls): tup
    = max(enumerate(ls)) return
    tup[0] - 1

nếu __name__ == "__main__": đơn
    vị = [10000, 20000, 40000, 60000] giá, đơn
    vị_chi phí, xử lý = 4, 1,5, 0,2 avg_p = [] x, n =
    1, 10000 lời
    nhuận_10 = mcs(x,
    n, đơn vị [0], giá, đơn vị_chi phí, xử lý) avg_p.append(np.mean(profit_10))
    print ('Lợi nhuận cho {:.0f}'.format(đơn
    vị[0]),
        'đơn vị: ${:,.2f}'.format(np.mean(profit_10))) profit_20
    = mcs(x, n, units[1], price, unit_cost, thải bỏ)
    avg_p.append(np.mean(np .mean(profit_20))) print
    ('Lợi nhuận cho {:.0f}'.format(đơn vị[1]),
        'đơn vị: ${:,.2f}'.format(np.mean(profit_20))) lời nhuận_40
    = mcs(x, n, đơn vị[2], giá, đơn vị_chi phí, xử lý) avg_p.append(np.mean(lời
    nhuận_40 )) print ('Lợi nhuận cho
    {:.0f}'.format(đơn vị[2]),
        'đơn vị: ${:,.2f}'.format(np.mean(profit_40))) profit_60
    = mcs(x, n, units[3], price, unit_cost, hủy bỏ) avg_p.append(np.mean(profit_60 ))

```

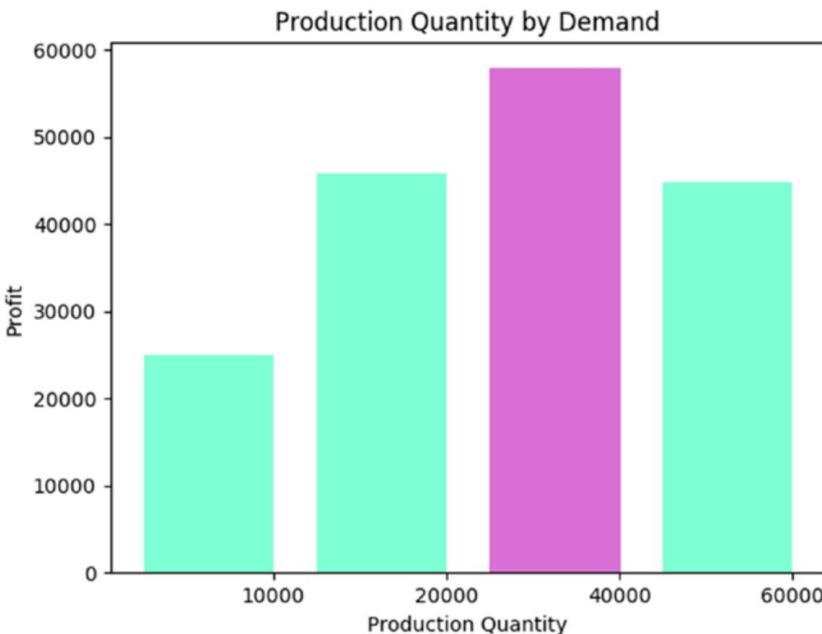
## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

```
print('Lợi nhuận cho {:.0f}'.format(dơn vị[3]),  
      'đơn vị: {:.2f}'.format(np.mean(profit_60)))  
nhᾶn = ['10000', '20000', '40000', '60000'] pos  
= np.arange(len(nhᾶn)) width  
= 0,75 # đặt khoảng cách giữa các ngăn nhỏ hơn 1,0  
plt.figure(2)  
ax = plt.axes()  
ax.set_xticks(pos + (width / 2))  
ax.set_xticklabels(nhᾶn)  
barlist = plt.bar(pos, avg_p, width, color='aquamarine')  
barlist[max_bar(avg_p)].set_color ('lan')  
plt.ylabel('Lợi  
nhận') plt.xlabel('Số lượng Sản  
xuất') plt.title('Số lượng Sản xuất theo Nhu  
cầu') plt.show()
```

Đầu ra:

```
Profit for 10,000 units: $25,000.00  
Profit for 20,000 units: $45,829.40  
Profit for 40,000 units: $57,886.60  
Profit for 60,000 units: $44,882.40
```

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ



Hình 2-6. trực quan hóa số lượng sản xuất

Mã bắt đầu bằng cách nhập các thư viện `matplotlib` và `numpy`. Nó tiếp tục với bốn chức năng. Hàm `nhu cầu()` bắt đầu bằng cách tạo ngẫu nhiên xác suất phân phối đồng đều. Nó tiếp tục bằng cách trả về một trong bốn kết quả xác suất riêng biệt được thiết lập bởi ván đề mà chúng tôi muôn giải quyết. Chức năng `sản xuất()` trả về lợi nhuận dựa trên thuật toán mà tôi nghĩ ra. Hãy nhớ rằng bất kỳ thuật toán cơ sở lợi nhuận nào cũng có thể được thay thế, điều này làm sáng tỏ tính linh hoạt đáng kinh ngạc của MCS. Hàm `mcs()` chạy mô phỏng 10.000 lần. Việc tăng số lần chạy mang lại độ chính xác dự đoán tốt hơn với chi phí là nhiều tài nguyên xử lý máy tính và thời gian chạy hơn. Hàm `max_bar()` thiết lập thanh cao nhất trong biểu đồ thanh để chiếu sáng tốt hơn. Khối chính bắt đầu bằng cách mô phỏng lợi nhuận cho từng kết quả xác suất rời rạc, đồng thời in và hiển thị kết quả. MCS dự đoán rằng số lượng sản xuất 40.000 chiếc sẽ mang lại lợi nhuận cao nhất, như trong Hình 2-6.

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

Việc tăng số lượng mô phỏng MCS dẫn đến dự đoán thực tế chính xác hơn vì nó dựa trên lý luận ngẫu nhiên (ngẫu nhiên hóa dữ liệu). Bạn cũng có thể thay thế bất kỳ phân phối xác suất rời rạc nào dựa trên nhu cầu giải quyết vấn đề của mình bằng cấu trúc mã này.

Như đã đề cập trước đó, bạn có thể sử dụng bất kỳ thuật toán nào bạn muốn dự đoán với MCS, biến nó thành một công cụ cực kỳ linh hoạt cho các nhà khoa học dữ liệu.

Chúng tôi có thể nâng cao độ chính xác hơn nữa bằng cách chạy MCS trên MCS. Ví dụ về mã sử dụng cùng một thuật toán và quy trình như trước, nhưng thêm một MCS trên MCS ban đầu để có được dự đoán chính xác hơn:

nhập matplotlib.pyplot dưới dạng plt, numpy dưới dạng np

nhu cầu chắc chắn():

```
p = np.random.uniform(0,1)
nếu p < 0,10:
    trả lại 10000
Elif p >= 0,10 và p < 0,45:
    trả lại 20000
eif p >= 0,45 và p < 0,75:
    trả lại 40000
khác:
    trả lại 60000
```

def sản xuất(nhu cầu, đơn vị, giá, unit\_cost, thải bỏ):

```
units_sold = min(đơn vị, nhu cầu)
doanh thu = số lượng bán * giá
tổng_chi phí = đơn vị * chi phí đơn vị
units_not_sold = đơn vị - nhu cầu
nếu units_not_sold > 0:
    xử lý_chi phí = xử lý * units_not_sold
```

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

khác:

```

xử lý_chi phí = 0
lợi nhuận = doanh thu - tổng_chi phí - xử lý_chi phí lợi
nhiệt trả lại

def mcs(x, n, đơn vị, giá, chi phí đơn vị, xử lý): lợi nhuận =
[] trong khi
x <= n:
    d = nhu cầu()
    v = sản xuất(d, đơn vị, giá, chi phí đơn vị, xử lý) lợi nhuận.append(v)

    x += 1
trả lại lợi nhuận

def display(p, i):
    print ('Lợi nhuận cho {:.0f}'.format(units[i]), 'units: $'
           {:.2f}'.format(np.mean(p)) )

nếu __name__ == "__main__":
    đơn
    vị = [10000, 20000, 40000, 60000] giá, đơn
    vị_chi phí, xử lý = 4, 1,5, 0,2 avg_ls = [] x, n,
    y, z = 1,
    10000, 1, 1000 trong khi y <= z:
        profit_10 =
            mcs(x, n, units[0], price, unit_cost, thanh lý) profit_20 =
            mcs(x, n,
            units[1], price, unit_cost, thanh lý) avg_profit =
np.mean(profit_20 ) profit_40 = mcs(x,
n, units[2], price, unit_cost, thanh lý) avg_profit =
np.mean(profit_40) profit_60 = mcs(x,
n, units[3], giá, unit_cost, thanh lý)

```

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

```

avg_profit = np.mean(profit_60)
avg_ls.append({'p10':np.mean(profit_10),
               'p20':np.mean(profit_20),
               'p40':np.mean(profit_40),
               'p60':np.mean(profit_60)})

y += 1
mcs_p10, mcs_p20, mcs_p40, mcs_p60 = [], [], [], []
cho hàng trong avg_ls:
    mcs_p10.append(hàng['p10'])
    mcs_p20.append(hàng['p20'])
    mcs_p40.append(hàng['p40'])
    mcs_p60.append(hàng['p60'])

hiển thị(np.mean(mcs_p10), 0)
hiển thị(np.mean(mcs_p20), 1)
hiển thị(np.mean(mcs_p40), 2)
hiển thị(np.mean(mcs_p60), 3)

```

Đầu ra:

```

Profit for 10,000 units: $25,000.00
Profit for 20,000 units: $45,800.24
Profit for 40,000 units: $57,980.97
Profit for 60,000 units: $44,996.88

```

Mã cho ví dụ này giống như mã trước đó, ngoại trừ vòng lặp MCS while (trong khi  $y \leq z$ ). Trong vòng lặp này, lợi nhuận được tính như trước khi sử dụng hàm `mcs()`, nhưng mỗi kết quả mô phỏng được thêm vào danh sách `avg_ls`. Vì vậy, `avg_ls` chứa 1.000 ( $z = 1000$ ) kết quả mô phỏng của kết quả mô phỏng ban đầu. Độ chính xác được tăng lên, nhưng cần nhiều tài nguyên máy tính và thời gian chạy hơn. Chạy 1.000 mô phỏng trên MCS ban đầu mất hơn một phút, rất nhiều thời gian xử lý!

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

## Tính ngẫu nhiên sử dụng các hàm mật độ tích lũy và xác suất

Tính ngẫu nhiên giả dạng thành thực tế (thế giới tự nhiên) trong khoa học dữ liệu, vì không thể dự đoán được tương lai. Nghĩa là, ngẫu nhiên hóa là cách các nhà khoa học dữ liệu mô phỏng thực tế. Nhiều dữ liệu hơn có nghĩa là độ chính xác và dự đoán tốt hơn (thực tế hơn). Nó đóng một vai trò quan trọng trong mô phỏng sự kiện rủi ro và giải quyết vấn đề xác định. Ngẫu nhiên hóa được sử dụng trong nhiều lĩnh vực như thống kê, MCS, mật mã, thống kê, y học và khoa học.

Mật độ của một biến ngẫu nhiên liên tục là mật độ xác suất của nó chức năng (PDF). PDF là xác suất mà một biến ngẫu nhiên có giá trị  $x$ , trong đó  $x$  là một điểm trong khoảng của một mẫu. Xác suất này được xác định bằng tích phân PDF của biến ngẫu nhiên trên phạm vi (khoảng) của mẫu. Nghĩa là, xác suất được đưa ra bởi khu vực dưới hàm mật độ, nhưng phía trên trực hoành và giữa các giá trị thấp nhất và cao nhất của phạm vi. Một tích phân (tích phân) là một đối tượng toán học có thể được hiểu là một khu vực dưới một đường cong phân phối bình thường. Hàm phân phối tích lũy (CDF) là xác suất mà một biến ngẫu nhiên có giá trị nhỏ hơn hoặc bằng  $x$ . Nghĩa là, CDF tích lũy tất cả các xác suất nhỏ hơn hoặc bằng  $x$ . Hàm điểm phần trăm (PPF) là hàm nghịch đảo của CDF. Nó thường được gọi là hàm phân phối tích lũy nghịch đảo (ICDF). ICDF rất hữu ích trong khoa học dữ liệu vì nó là giá trị thực được liên kết với một khu vực

dưới PDF. Vui lòng tham khảo [www.itl.nist.gov/div898/handbook/eda/phan3/eda362.htm](http://www.itl.nist.gov/div898/handbook/eda/phan3/eda362.htm) để có một lời giải thích tuyệt vời về các hàm mật độ.

Như đã nêu trước đó, xác suất được xác định bằng tích phân PDF của biến ngẫu nhiên trong khoảng thời gian của một mẫu. Nghĩa là, tích phân được dùng để xác định xác suất của một biến ngẫu nhiên nào đó nằm trong một khoảng (mẫu) nhất định. Trong giải tích, tích phân đại diện cho một loại hàm (nguyên hàm) có đạo hàm là tích phân. Ký hiệu tích phân biểu thị tích phân, trong khi tích phân là hàm

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

tích phân trong tích phân xác định hoặc không xác định. Định lý cơ bản của giải tích liên quan đến việc đánh giá các tích phân xác định với các tích phân không xác định. Lý do duy nhất tôi đưa thông tin này vào đây là để nhấn mạnh tầm quan trọng của phép tính đối với khoa học dữ liệu. Một khía cạnh khác của phép tính quan trọng đối với khoa học dữ liệu, "hệ số giảm dần", được trình bày sau trong Chương 4.

Mặc dù những giải thích lý thuyết là vô giá, nhưng chúng có thể không trực quan. Một cách tuyệt vời để hiểu rõ hơn về các khái niệm này là xem xét một ví dụ.

Trong ví dụ mã, biểu đồ 2-D được tạo cho PDF, CDF và ICDF (PPF). Ý tưởng về một bản đồ màu được bao gồm trong ví dụ. Bản đồ màu là một bảng tra cứu chỉ định các màu sẽ được sử dụng để hiển thị hình ảnh được làm mờ. Một hình ảnh được chỉnh màu là một hình ảnh được mã hóa hiệu quả bằng cách ánh xạ các pixel của nó vào một bảng màu chỉ chứa những màu thực sự có trong hình ảnh. Thư viện matplotlib bao gồm vô số bản đồ màu. Vui lòng tham khảo [https://matplotlib.org/examples/color/colormaps\\_reference.html](https://matplotlib.org/examples/color/colormaps_reference.html) cho các bản đồ màu có sẵn.

```

nhập matplotlib.pyplot dưới dạng plt
từ định mức nhập scipy.stats
nhập numpy dưới dạng np

nếu __name__ == '__main__':
    x = np.linspace(norm.ppf(0,01), Norm.ppf(0,99), num=1000)
    y1 = định mức.pdf(x)
    plt.figure('PDF')
    plt.xlim(x.min()-.1, x.max()+0.1)
    plt.ylim(y1.min(), y1.max()+0.01)
    plt.xlabel('x')
    plt.ylabel('Mật độ xác suất')
    plt.title('PDF bình thường')
    plt.scatter(x, y1, c=x, cmap='jet')

```

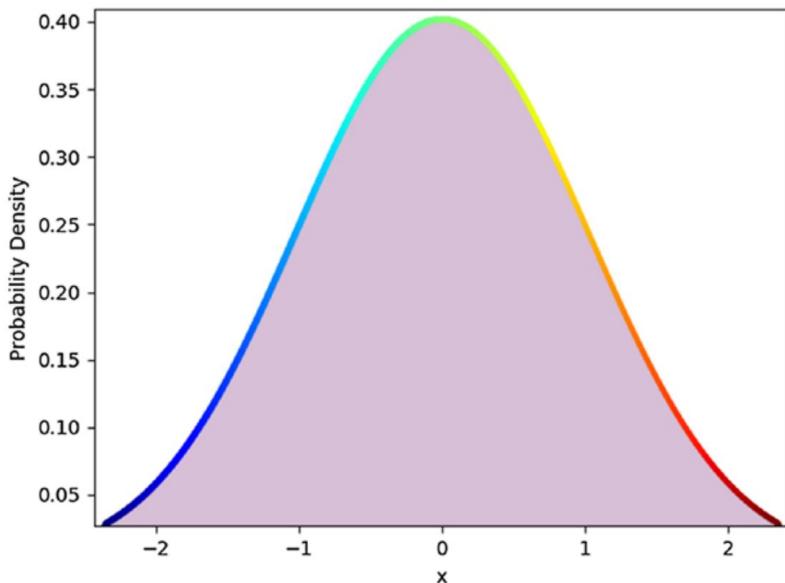
## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

```
plt.fill_between(x, y1, color='thistle')
plt.show()
plt.close('PDF')
plt.figure('CDF')
plt.xlabel('x')
plt.ylabel('Xác suất')
plt.title('CDF bình
thường') y2
=normal.cdf(x) plt.scatter(x, y2,
c=x,
cmap='jet')
plt.show()
plt.close('CDF')

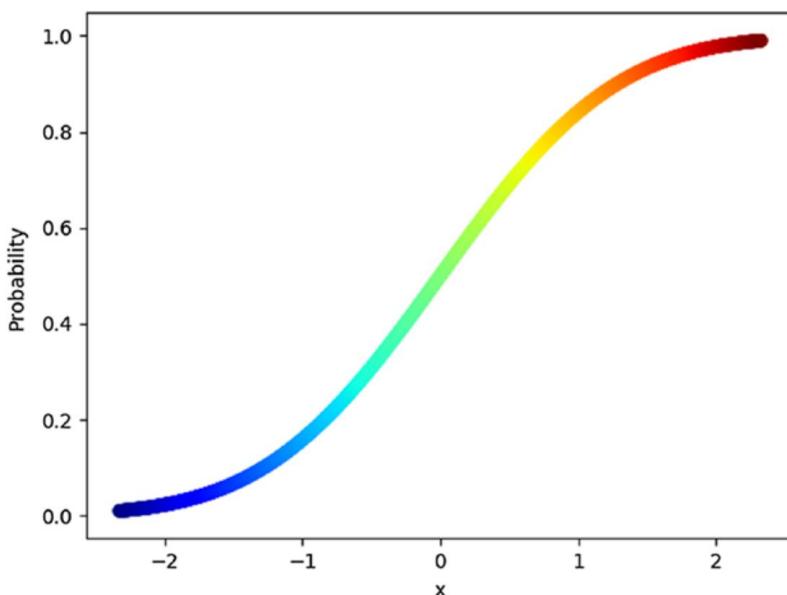
plt .figure('ICDF')
plt.xlabel('Xác
sựt') plt.ylabel('x') plt.title('ICDF
bình
thường (PPF)') y3 = Norm.ppf(x) plt.scatter(x, y3 , c=x, cmap='jet') plt.sh
```

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

Đầu ra:

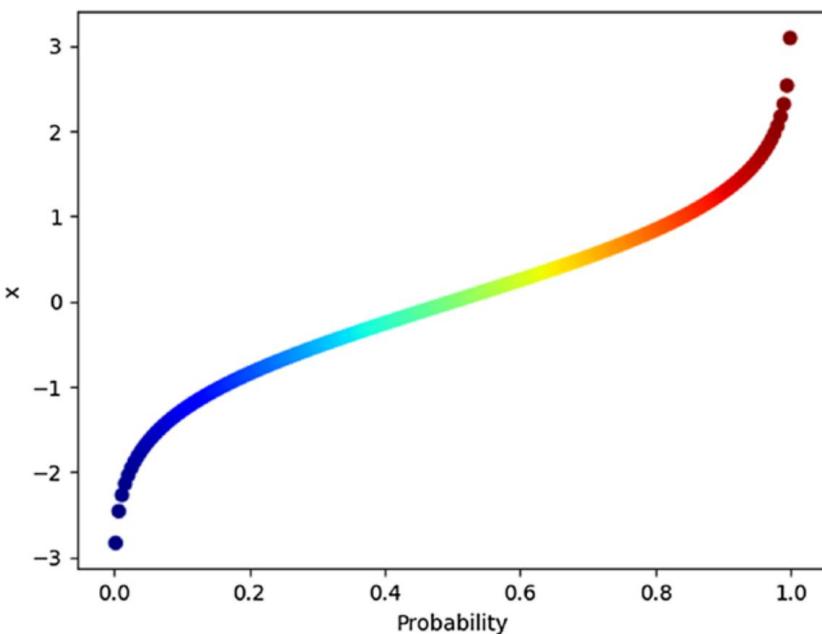


Hình 2-7. Trực quan hóa hàm mật độ xác suất bình thường



Hình 2-8. Trực quan hóa hàm phân phối tích lũy bình thường

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ



Hình 2-9. Trực quan hóa hàm phân phối tích lũy nghịch đảo bình thường

Mã bắt đầu bằng cách nhập ba thư viện – `matplotlib`, `scipy` và `numpy`. Khởi chính bắt đầu bằng cách tạo một chuỗi gồm  $1.000 \times$  giá trị trong khoảng từ 0,01 đến 0,99 (vì xác suất phải nằm trong khoảng từ 0 đến 1). Tiếp theo, một chuỗi các giá trị PDF y được tạo dựa trên các giá trị x. Mã tiếp tục bằng cách vẽ đồ thị PDF kết quả được hiển thị trong Hình 2-7. Tiếp theo, một chuỗi các giá trị CDF (Hình 2-8) và ICDF (Hình 2-9) được tạo và vẽ trên đồ thị. Từ hình ảnh trực quan, dễ dàng thấy rằng PDF đại diện cho tất cả các giá trị x có thể có (xác suất) tồn tại theo phân phối chuẩn. Việc hình dung CDF cũng dễ dàng hơn vì nó đại diện cho sự tích lũy của tất cả các xác suất có thể xảy ra. Cuối cùng, ICDF dễ hiểu hơn thông qua trực quan hóa (xem Hình 2-9) vì trực x biểu thị các xác suất, trong khi trực y biểu thị giá trị thực liên quan đến các xác suất đó.

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

Hãy áp dụng ICDF. Giả sử bạn là một nhà khoa học dữ liệu tại Apple và sếp của bạn yêu cầu bạn xác định tỷ lệ hỏng hóc của Apple iPhone 8 để cô ấy có thể phát triển một bản trình bày mô hình cho cấp trên của mình. Đối với ví dụ giả định này, sếp của bạn mong đợi bốn phép tính: thời gian cần 5% điện thoại hỏng, khoảng thời gian (phạm vi) khi 95% điện thoại hỏng, thời gian 5% điện thoại tồn tại (không hỏng) và khoảng thời gian nơi 95% điện thoại tồn tại. Trong mọi trường hợp, thời gian báo cáo tính bằng giờ. Từ việc khám phá dữ liệu, bạn xác định được thời gian hỏng hóc trung bình ( $\mu$ ) là 1.000 giờ và độ lệch chuẩn ( $\sigma$ ) là 300 giờ.

Ví dụ mã tính toán ICDF cho bốn tình huống và hiển thị kết quả ở định dạng dễ hiểu cho sếp của bạn:

```
từ định mức nhập scipy.stats
nhập numpy dưới dạng np

def np_rstrip(v):
    trả về np.char.rstrip(v.astype(str), '.0')

biến đổi xác định (t):
    môt, hai = vòng(t[0]), vòng(t[1])
    trả về (np_rstrip(môt), np_rstrip(hai))

nếu __name__ == "__main__":
    mu, sigma = 1000, 300
    print('Tỷ lệ thất bại dự kiến:')
    fail = np_rstrip(round(norm.ppf(0.05, loc=mu, scale=sigma)))
    print ('5% fail trong vòng', fail, 'giờ')
    fail_range = Norm.interval(0.95, loc=mu, scale=sigma)
    lo, hi = biến đổi (fail_range)
    print ('95% fail giữa', lo, 'and', hi, end=' ')
    print ('số giờ sử dụng')
    print ('\nTỷ lệ sống sót dự kiến:') last
    = np_rstrip(round(norm.ppf(0.95, loc=mu, scale=sigma)))
    print ('5% tồn tại đến', cuối cùng, 'giờ sử dụng')
```

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

```
last_range = Norm.interval(0.05, loc=mu, scale=sigma)
lo, hi = biến đổi (last_range)
print ('95% tồn tại giữa', lo, 'và', hi, 'giờ sử dụng')
```

Đầu ra:

```
Expected failure rates:
5% fail within 507 hours
95% fail between 412 and 1588 hours of usage

Expected survival rates:
5% survive up to 1493 hours of usage
95% survive between 981 and 1019 hours of usage
```

Ví dụ mã bắt đầu bằng cách nhập các thư viện `scipy` và `numpy`. Nó tiếp tục với hai chức năng. Hàm `np_rstrip()` chuyển đổi số float thành chuỗi và loại bỏ các ký tự không liên quan. Hàm `transform()` làm tròn và trả về một bộ. Cả hai chỉ được sử dụng để làm tròn số đến chữ số thập phân không để làm cho nó thân thiện với người dùng đối với ông chủ hư cấu của bạn. Khoi chính bắt đầu bằng cách khởi tạo `mu` và `sigma` thành `1.000` (`lỗi`) và `300` (`biến thể`). Tức là, trung bình, điện thoại thông minh của chúng tôi bị lỗi trong vòng `1.000` giờ và các lỗi khác nhau trong khoảng từ `700` đến `1.300` giờ. Tiếp theo, tìm giá trị ICDF cho tỷ lệ thất bại `5%` và khoảng thời gian mà `95%` thất bại với `Norm.ppf()`. Vì vậy, `5%` tổng số điện thoại được cho là sẽ hỏng trong vòng `507` giờ, trong khi `95%` sẽ hỏng trong khoảng thời gian từ `412` đến `1.588` giờ sử dụng. Tiếp theo, tìm giá trị ICDF cho tỷ lệ sống sót `5%` và khoảng thời gian mà `95%` sống sót. Vì vậy, `5%` tổng số điện thoại tồn tại 1.493 giờ, trong khi `95%` tồn tại trong khoảng từ `981` đến `1.019` giờ sử dụng.

Đơn giản, ICDF cho phép bạn làm việc ngược từ xác suất đã biết để tìm giá trị `x`! Vui lòng tham khảo <http://support.minitab.com/en-us/minitab-express/1/help-and-how-to/basic-statistics/phân-phối-xác-suất/chủ-de-hỗ-trợ/cơ-bản/sử-dụng-nghịch-đảo-tích-lũy-phân-phối-hàm-icdf/#what-is-an-inverse-tích-lũy-phân-phối-chức-năng-icdf> để biết thêm thông tin.

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

Hãy thử phân tích What-if. Điều gì sẽ xảy ra nếu chúng tôi giảm tỷ lệ lỗi ( $\sigma$ ) từ 300 đến 30?

**Expected failure rates:**  
**5% fail within 951 hours**  
**95% fail between 941 and 1059 hours of usage**

**Expected survival rates:**  
**5% survive up to 1049 hours of usage**  
**95% survive between 998 and 1002 hours of usage**

Giờ đây, 5% tổng số điện thoại được cho là sẽ hỏng trong vòng 951 giờ, trong khi 95% thất bại giữa 941 và 1.059 giờ sử dụng. Và, 5% trong số tất cả điện thoại tồn tại tới 1.049 giờ, trong khi 95% tồn tại trong khoảng từ 998 đến 1.002 giờ sử dụng. Điều đó có nghĩa là gì? Biến thể ít hơn (lỗi) cho thấy các giá trị gần với mức trung bình hơn cho cả tỷ lệ thất bại và tỷ lệ sống sót. Điều này có ý nghĩa vì biến thể được tính từ giá trị trung bình là 1.000.

Hãy chuyển sang một ví dụ mô phỏng. Giả sử sép của bạn yêu cầu bạn tìm số lượng đặt hàng hàng tháng tối ưu cho một loại ô tô với điều kiện là nhu cầu được phân phối chuẩn (điều đó phải xảy ra, vì PDF dựa trên giả định này), nhu cầu trung bình (mu) là 200 và độ biến thiên ( $\sigma$ ) là 30. Mỗi chiếc ô tô có giá 25.000 đô la, được bán với giá 45.000 đô la và một nửa số ô tô không được bán với giá đầy đủ có thể được bán với giá 30.000 đô la. Giống như các thử nghiệm MCS khác, bạn có thể sửa đổi thuật toán lợi nhuận để nâng cao tính hiện thực. Theo nhà cung cấp, bạn bị giới hạn số lượng đặt hàng là 160, 180, 200, 220, 240, 260 hoặc 280.

MCS được sử dụng để tìm kiếm lợi nhuận cho mỗi đơn đặt hàng dựa trên thông tin cung cấp. Nhu cầu được tạo ngẫu nhiên cho mỗi lần lặp lại mô phỏng. Tính toán lợi nhuận theo đơn hàng được tự động hóa bằng cách chạy MCS cho mỗi đơn hàng.

```
nhập numpy dưới dạng np
nhập matplotlib.pyplot dưới dạng plt

def str_int(s):
    val = "%.2f" % lợi nhuận
    trả về float (val)
```

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

```

nếu __name__ == "__main__": đơn
    đặt hàng = [180, 200, 220, 240, 260, 280, 300] mu,
    sigma, n = 200, 30, 10000 chi phí,
    giá, chiết khấu = 25000, 45000, 30000 lợi nhuận_ls =
    [ ] cho thứ tự
    trong đơn đặt hàng:
        x = 1

        Profit_val = []
        inv_cost = đơn hàng * chi
        phí trong khi x <= n:
            nhu cầu = round(np.random.normal(mu, sigma)) nếu nhu
            cầu < thứ tự:
                diff = thứ tự - nhu cầu
                nếu khác biệt > 0:
                    damt = round(abs(diff) / 2) * lợi nhuận chiết
                    khấu = (nhu cầu * giá) - inv_cost + damt
                khác:
                    lợi nhuận = (đơn hàng * giá) - inv_cost
                khác:
                    lợi nhuận = (đơn hàng * giá) - lợi nhuận
                    inv_cost = str_int(đơn hàng)
                    Profit_val.append(lợi nhuận)
                    x += 1

        avg_profit = np.mean(profit_val)
        profit_ls.append(avg_profit) print
        ('${0:.2f}'.format(avg_profit), '(profit)', 'for order:',
         order) max_profit =
        max(Profit_ls) Profit_np =
        np.array(profit_ls) max_ind =
        np.where(profit_np == Profit_np.max()) print ('\nLợi nhuận
        tối đa', '${0:.2f}'.format(max_profit),

```

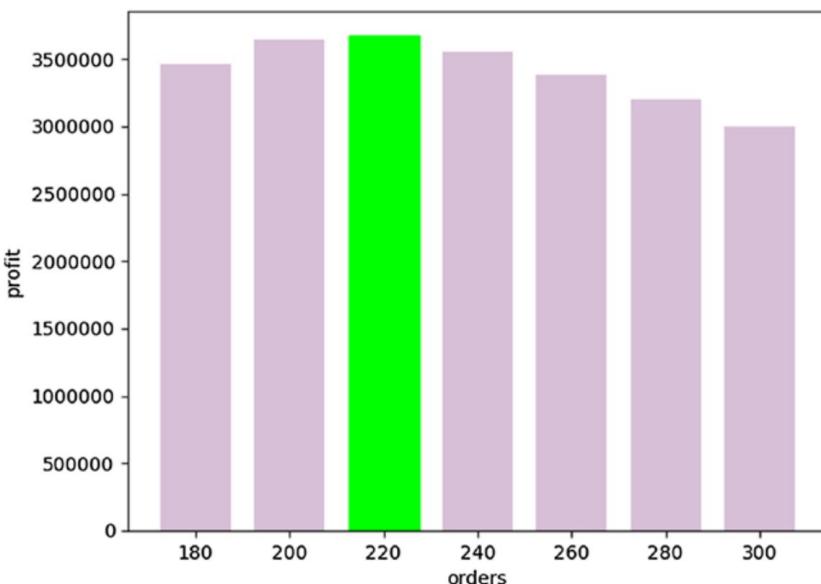
## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

```
'cho đơn hàng', đơn hàng[int(max_ind[0])])  
barlist = plt.bar(orders,profit_ls, width=15,  
color='thistle')  
barlist[int(max_ind[0])].set_color('violet')  
plt.title('Lợi nhuận theo số lượng đặt  
hàng') plt.xlabel('đơn  
đặt hàng')  
plt.ylabel('lợi  
nhuận') plt.tight_layout() plt.show()
```

Đầu ra:

```
$3,460,479.00 (profit) for order: 180  
$3,638,933.50 (profit) for order: 200  
$3,669,597.50 (profit) for order: 220  
$3,554,889.00 (profit) for order: 240  
$3,385,369.00 (profit) for order: 260  
$3,200,210.00 (profit) for order: 280  
$2,994,411.00 (profit) for order: 300  
  
Maximum profit $3,669,597.50 for order 220
```

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ



Hình 2-10. Lợi nhuận theo trực quan hóa số lượng đặt hàng

Mã bắt đầu bằng cách nhập numpy và matplotlib. Nó tiếp tục với một hàm (`str_int()`) chuyển đổi một chuỗi thành float. Khối chính bắt đầu bằng cách khởi tạo các đơn đặt hàng, mu, sigma, n, chi phí, giá cả, chiết khấu và danh sách lợi nhuận theo đơn đặt hàng. Nó tiếp tục bằng cách lặp qua từng số lượng đặt hàng và chạy MCS với 10.000 lần lặp. Xác suất nhu cầu được tạo ngẫu nhiên được sử dụng để tính toán lợi nhuận cho mỗi lần lặp mô phỏng. Kỹ thuật tính toán lợi nhuận khá đơn giản, nhưng bạn có thể thay thế thuật toán của riêng mình. Bạn cũng có thể sửa đổi bất kỳ thông tin đã cho nào dựa trên dữ liệu của riêng bạn. Sau khi tính toán lợi nhuận cho mỗi đơn đặt hàng thông qua MCS, mã tiếp tục bằng cách tìm số lượng đặt hàng có lợi nhuận cao nhất. Cuối cùng, mã này tạo ra một biểu đồ thanh để làm sáng tỏ các kết quả thông qua trực quan hóa được hiển thị trong Hình 2-10.

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

Ví dụ mã cuối cùng tạo ra một hình ảnh PDF:

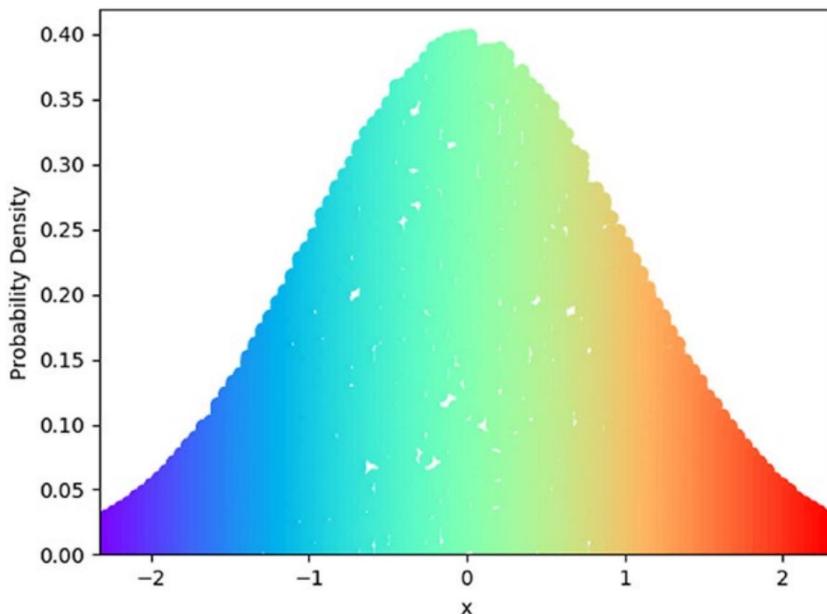
nhập matplotlib.pyplot dưới dạng plt, numpy dưới dạng np từ định mức nhập scipy.stats

```
nếu __name__ == '__main__':
    n = 100

    x = np.linspace(norm.ppf(0.01), Norm.ppf(0.99), num=n) y =
    Norm.pdf(x) dic
    = {} for
    i, row in enumerate(y):
        dic[x[ i]] = [np.random.uniform(0, row) for xs      _ trong phạm vi (n)]
    = [] ys
    = []
    for key, vals in dic.items(): for
        y in vals:
            xs.append(key)
            ys.append( y)
    plt.xlim(min(xs), max(xs))
    plt.ylim(0, max(ys)+0.02)
    plt.title('PDF bình
    thường')
    plt.xlabel('x') plt.ylabel( 'Mật độ
    xác suất') plt.scatter(xs, ys, c=xs,
    cmap='rainbow') plt.show()
```

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

Đầu ra:

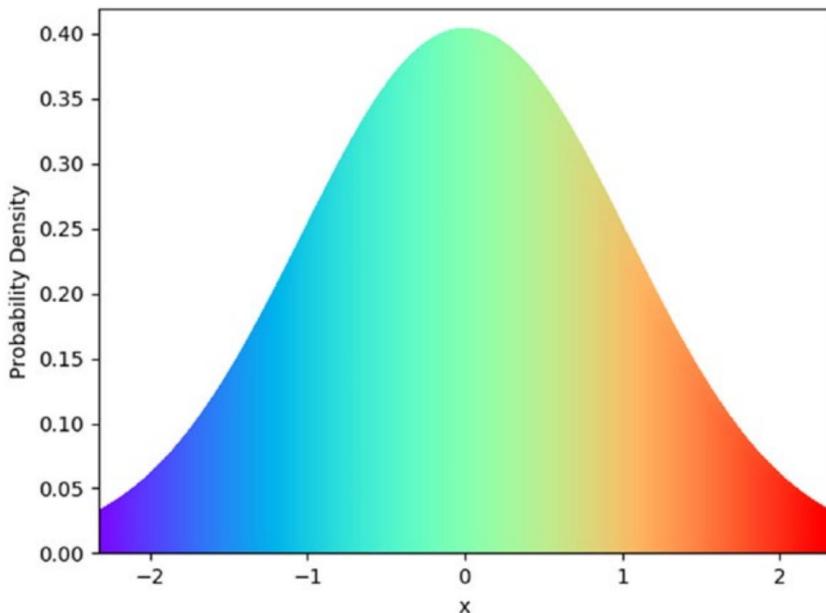


Hình 2-11. Tất cả các xác suất PDF với 100 mô phỏng

Mã bắt đầu bằng cách nhập các thư viện matplotlib, numpy và scipy. Khởi chính bắt đầu bằng cách khởi tạo số điểm bạn muốn vẽ, giá trị PDF x và y và từ điển. Để vẽ tất cả các xác suất PDF, một tập hợp các giá trị được tạo ngẫu nhiên cho từng điểm trên trục x được tạo. Để hoàn thành tác vụ này, đoạn mã gán 100 giá trị ( $n = 100$ ) cho x từ 0,01 đến 0,99. Nó tiếp tục bằng cách gán 100 giá trị PDF cho y. Tiếp theo, một phần tử từ điển được điền bằng một cặp (khóa, giá trị) bao gồm mỗi giá trị x làm khóa và danh sách 100 ( $n = 100$ ) số được tạo ngẫu nhiên trong khoảng từ 0 đến  $\text{pdf}(x)$  làm giá trị được liên kết với x. Mặc dù mã tạo từ điển rất đơn giản, nhưng hãy suy nghĩ cẩn thận về những gì đang xảy ra vì nó khá trừu tượng. Mã tiếp tục bằng cách xây dựng các cặp  $(x, y)$  từ từ điển. Kết quả là 10.000 ( $100 \times 100$ )  $(x, y)$  cặp, trong đó mỗi 100 giá trị x có 100 giá trị y liên kết được hiển thị trong Hình 2-11.

## Chương 2 Mô phỏng Monte Carlo và Hàm mật độ

Để làm mượt hình ảnh, hãy tăng n lên 1.000 ( $n = 1000$ ) ở đầu khôi chính:



Hình 2-12. Tất cả các xác suất PDF với 1.000 mô phỏng

Bằng cách tăng n lên 1000, các cặp 1.000.000 (1.000 X 1.000) ( $x, y$ ) được vẽ như hình 2-12!

## CHƯƠNG 3

# Đại số tuyến tính

Đại số tuyến tính là một nhánh của toán học liên quan đến không gian vectơ và ánh xạ tuyến tính giữa các không gian đó. Đơn giản, nó khám phá các mối quan hệ giống như dòng. Trên thực tế, mọi lĩnh vực của khoa học hiện đại đều xấp xỉ mô hình hóa các phương trình với đại số tuyến tính. Cụ thể, khoa học dữ liệu dựa vào đại số tuyến tính để học máy, lập mô hình toán học và giải quyết vấn đề phân bố chiều.

### không gian véc tơ

Không gian vectơ là một tập hợp các vectơ. Vectơ là bất kỳ đại lượng nào có độ lớn và hướng xác định vị trí của một điểm trong không gian so với điểm khác. Độ lớn là kích thước của một vật thể được đo bằng chuyển động, chiều dài và hoặc vận tốc. Các vectơ có thể được thêm và nhân (bằng vô hướng) để tạo thành các vectơ mới. Vô hướng là bất kỳ đại lượng nào có độ lớn (kích thước). Trong ứng dụng, vectơ là điểm trong không gian hữu hạn.

Các ví dụ về vectơ bao gồm thở, đi bộ và dịch chuyển.

Thở đòi hỏi cơ hoành tác dụng một lực có cường độ và hướng. Đi bộ đòi hỏi phải di chuyển theo một số hướng.

Độ dịch chuyển do khoảng cách mà một vật thể di chuyển theo một hướng nhất định.

## Chương 3 Đại số tuyến tính

## Toán Véc Tơ

Trong toán học vectơ, một vectơ được mô tả dưới dạng một đoạn thẳng có hướng có chiều dài là vectơ độ lớn của nó với một mũi tên chỉ hướng từ đuôi đến đầu. Đầu là vị trí bắt đầu của đoạn thẳng và đầu là vị trí kết thúc (mũi tên). Các vectơ bằng nhau nếu chúng có cùng độ lớn và hướng.

Để cộng hai vectơ  $a$  và  $b$ , hãy bắt đầu  $b$  ở vị trí  $a$  kết thúc và hoàn thành tam giác. Trực quan, bắt đầu tại một số điểm gốc, vẽ  $a$  (Hình 3-1), bắt đầu  $b$  (Hình 3-2) từ đầu của  $a$  và kết quả  $c$  (Hình 3-3) là một đường thẳng từ đuôi của  $a$  đến đầu của  $B$ . Ví dụ đầu tiên minh họa phép cộng vectơ cũng như mô tả đồ họa của quy trình:

nhập matplotlib.pyplot dưới dạng plt, numpy dưới dạng np

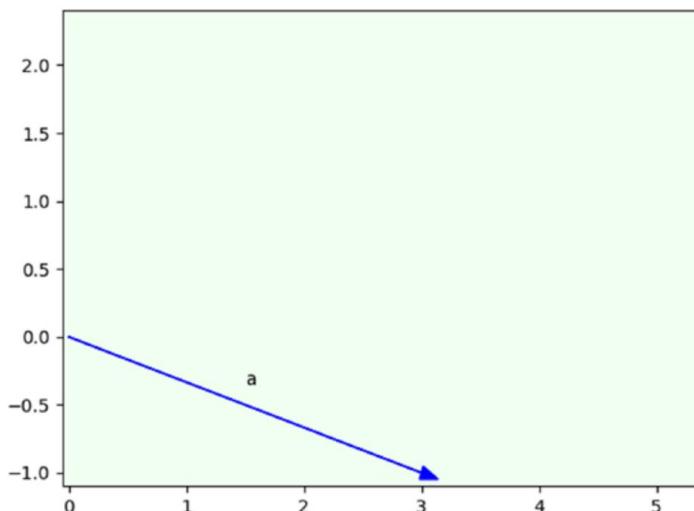
```
def vector_add(a, b):
    trả về np.add(a, b)

def set_up():
    plt.figure()
    plt.xlim(-.05, add_vectors[0]+0.4)
    plt.ylim(-1.1, add_vectors[1]+0.4)

nếu __name__ == "__main__":
    v1, v2 = np.array([3, -1]), np.array([2, 3])
    add_vector = vector_add(v1, v2)
    cài đặt()
    ax = plt.axes()
    ax.arrow(0, 0, 3, -1, head_width=0.1, fc='b', ec='b')
    ax.text(1.5, -0.35, 'a')
    ax.set_facecolor('honeydew')
    cài đặt()
    ax = plt.axes()
    ax.arrow(0, 0, 3, -1, head_width=0.1, fc='b', ec='b')
    ax.arrow(3, -1, 2, 3, head_width=0.1, fc='crimson', ec='crimson')
```

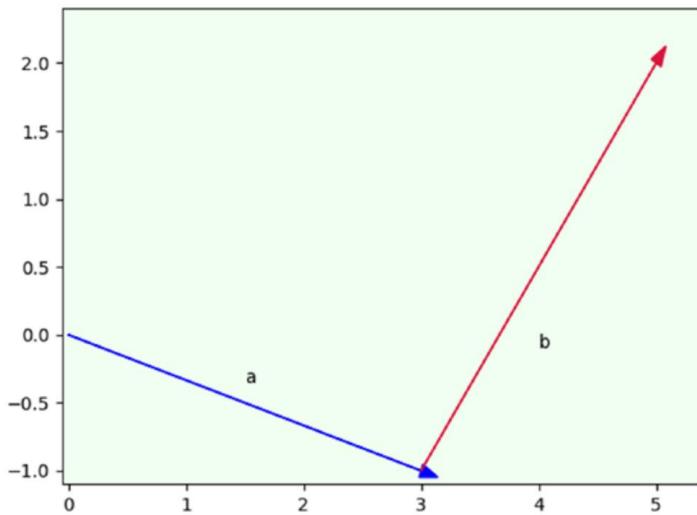
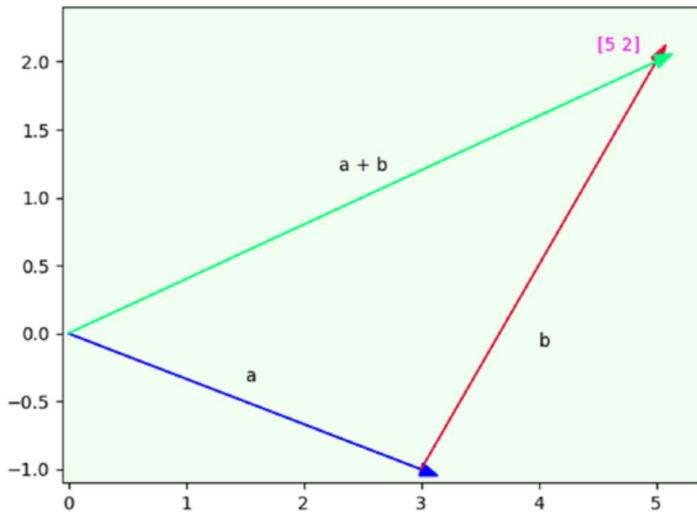
```
ax.text(1.5, -0.35, 'a')
ax.text(4, -0.1, 'b')
ax.set_facecolor('honeydew')
set_up()
ax = plt.axes()
ax.arrow(0, 0, 3, -1, head_width=0.1, fc='b', ec='b')
ax.arrow(3, -1, 2, 3, head_width=0.1, fc='crimson',
ec='crimson
') ax.arrow(0, 0, 5, 2, head_width=0.1, fc='springgreen',
ec='springgreen')
ax.text(1.5, -0.35, 'a')
ax.text(4, -0.1, 'b')
ax.text(2.3, 1.2, 'a + b')
ax.text(4.5, 2.08, add_vectors, color='fuchsia')
ax.set_facecolor('honeydew')
plt.show()
```

Đầu ra:



Hình 3-1. Vectơ a từ gốc tọa độ  $(0, 0)$  đến  $(3, -1)$

## Chương 3 Đại số tuyến tính

Hình 3-2. Vectơ  $b$  từ  $(3, -1)$  đến  $(5, 2)$ Hình 3-3. Vectơ  $c$  từ  $(0, 0)$  đến  $(5, 2)$

Mã bắt đầu bằng cách nhập các thư viện matplotlib và numpy.

Thư viện matplotlib là một thư viện vẽ sơ đồ được sử dụng để trực quan hóa chất lượng cao.

Thư viện numpy là gói cơ bản cho tính toán khoa học. Nó là một thư viện tuyệt vời để làm việc với vectơ và ma trận. Đoạn mã tiếp tục với hai hàm–vector\_add() và set\_up(). Hàm vector\_add() thêm hai vectơ. Hàm set\_up() thiết lập hình để vẽ đồ thị. Khối chính bắt đầu bằng cách tạo hai vectơ và thêm chúng.

Phần còn lại của mã trình bày bằng đồ họa cách hoạt động của phép cộng vectơ. Đầu tiên, nó tạo một đối tượng axis() với một mũi tên đại diện cho vectơ a bắt đầu tại gốc tọa độ  $(0, 0)$  và kết thúc tại  $(3, -1)$ . Nó tiếp tục bằng cách thêm văn bản và màu nền. Tiếp theo, nó tạo một đối tượng axis() thứ 2 có cùng mũi tên a, nhưng thêm mũi tên b (vectơ b) bắt đầu từ  $(3, -1)$  và tiếp tục đến  $(2, 3)$ . Cuối cùng, nó tạo một đối tượng axis() thứ 3 có cùng mũi tên a và b, nhưng thêm mũi tên c ( $a + b$ ) bắt đầu từ  $(0, 0)$  và kết thúc ở  $(5, 2)$ .

Ví dụ thứ 2 sửa đổi ví dụ trước bằng cách sử dụng các ô con (Hình 3-4). Các ô con chia một hình thành lưới  $m \times n$  để có trải nghiệm trực quan hóa khác.

nhập matplotlib.pyplot dưới dạng plt, numpy dưới dạng np

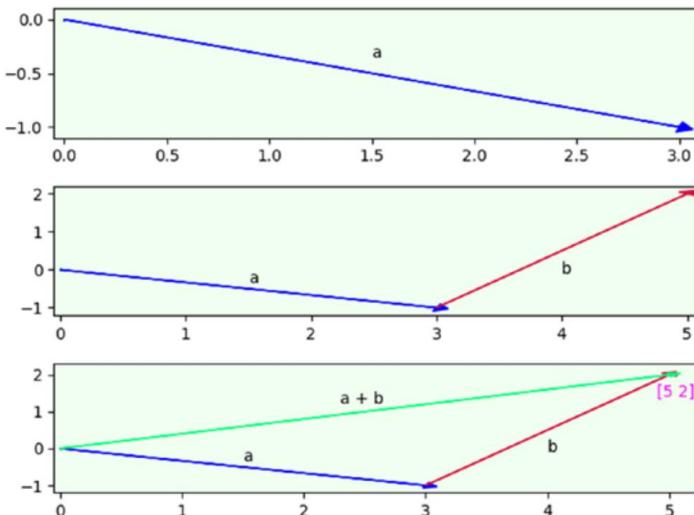
```
def vector_add(a, b):
    trả về np.add(a, b)

nếu __name__ == "__main__":
    v1, v2 = np.array([3, -1]), np.array([2, 3])
    add_vector = vector_add(v1, v2)
    f, ax = plt.subplots(3)
    x, y = [0, 3], [0, -1]
    ax[0].set_xlim([-0.05, 3.1])
    ax[0].set_ylim([-1.1, 0.1])
    ax[0].scatter(x,y,s=1)
    ax[0].arrow(0, 0, 3, -1, head_width=0.1, head_length=0.07,
               fc='b', ec='b')
```

## Chương 3 Đại số tuyến tính

```
ax[0].text(1.5, -0.35, 'a')
ax[0].set_facecolor('honeydew')
x, y = ([0, 3, 5]), ([0, -1, 2] )
ax[1].set_xlim([-0.05, 5.1])
ax[1].set_ylim([-1.2, 2.2])
ax[1].scatter(x,y,s=0.5)
ax[1].arrow (0, 0, 3, -1, head_width=0.2, head_length=0.1,
             fc='b', ec='b')
ax[1].arrow(3, -1, 2, 3, head_width=0.16, head_length=0.1,
            fc='crimson', ec='crimson')
ax[1].text(1.5, -0.35, 'a')
ax[1].text(4, -0.1, 'b')
ax[1].set_facecolor ('honeydew')
x, y = ([0, 3, 5]), ([0, -1, 2])
ax[2].set_xlim([-0.05, 5.25])
ax[2].set_ylim( [-1.2, 2.3])
ax[2].scatter(x,y,s=0.5)
ax[2].arrow(0, 0, 3, -1, head_width=0.15, head_length=0.1,
            fc='b ', ec='b')
ax[2].arrow(3, -1, 2, 3, head_width=0.15, head_length=0.1,
            fc='đỏ thẫm', ec='đỏ thẫm')
ax[2].arrow(0, 0, 5, 2, head_width=0.1, head_length=0.1,
            fc='springgreen', ec='springgreen')
ax[2].text(1.5, -0.35, 'a')
ax[2].text(4, -0.1, 'b')
ax[2].text (2.3, 1.2, 'a + b')
ax[2].text(4.9, 1.4, add_vectors, color='fuchsia')
ax[2].set_facecolor('honeydew')
plt. chăt_layout()
plt.show( )
```

Đầu ra:



Hình 3-4. Subplot Visualization của Vector Addition

Mã bắt đầu bằng cách nhập các thư viện matplotlib và numpy. Nó tiếp tục với cùng chức năng `vector_add()`. Khối chính tạo ba ô con với `plt.subplots(3)` và gán cho `f` và `ax`, trong đó `f` đại diện cho hình và `ax` đại diện cho từng ô con (`ax[0]`, `ax[1]` và `ax[2]`).

Thay vì làm việc với một hình, đoạn mã này xây dựng từng ô con bằng cách lập chỉ mục `ax`. Mã này sử dụng `plt. chăt_layout()` để tự động căn chỉnh từng ô con.

Ví dụ thứ 3 cộng trừ véc tơ. Phép trừ hai vectơ là phép cộng có số đối (phủ định) của một vectơ. Vì vậy, vectơ  $a$  trừ vectơ  $b$  bằng  $a + (-b)$ . Ví dụ mã minh họa phép cộng và phép trừ vectơ cho cả vectơ 2 và 3 chiều:

nhập numpy dưới dạng np

```
def vector_add(a, b):
    trả về np.add(a, b)

def vector_sub(a, b):
    trả về np.subtract(a, b)
```

## Chương 3 Đại số tuyến tính

```

nếu __name__ == "__main__":
    v1, v2 = np.array([3, -1]), np.array([2, 3])
    thêm = vector_add(v1, v2)
    phụ = vector_sub(v1, v2)
    print('Vectơ 2D:')
    in (v1, '+', v2, '=', thêm)
    in (v1, '-', v2, '=', phụ)
    v1 = np.array([1, 3, -5])
    v2 = np.array([2, -1, 3])
    thêm = vector_add(v1, v2)
    phụ = vector_sub(v1, v2)
    in ('\nVectơ 3D:')
    in (v1, '+', v2, '=', thêm)
    in (v1, '-', v2, '=', phụ)

```

Đầu ra:

```

2D vectors:
[ 3 -1] + [2 3] = [5 2]
[ 3 -1] - [2 3] = [ 1 -4]

3D vectors:
[ 1 3 -5] + [ 2 -1 3] = [ 3 2 -2]
[ 1 3 -5] - [ 2 -1 3] = [-1 4 -8]

```

Mã bắt đầu bằng cách nhập thư viện numpy. Nó tiếp tục với các hàm vector\_add() và vector\_subtract(), lần lượt cộng và trừ các vectơ. Khối chính bắt đầu bằng cách tạo hai vectơ 2 chiều, cộng và trừ chúng. Nó tiếp tục bằng cách cộng và trừ hai vectơ 3-D. Bất kỳ n chiều nào cũng có thể được cộng và trừ theo cách tương tự.

Độ lớn được đo bằng công thức khoảng cách. Độ lớn của một vectơ duy nhất được đo từ gốc tọa độ (0, 0) đến vectơ. Độ lớn giữa hai vectơ được đo từ vectơ thứ nhất đến vectơ thứ 2 . Khoảng cách công thức là căn bậc hai của (( giá trị thứ nhất từ vectơ thứ 2 trừ giá trị thứ nhất từ bình phương vectơ thứ nhất ) cộng với ( giá trị thứ 2 từ vectơ thứ 2 trừ giá trị thứ 2 từ bình phương vectơ thứ nhất )).

## Toán ma trận

Một ma trận là một mảng các số. Nhiều phép toán có thể được thực hiện trên một ma trận như cộng, trừ, phủ định, nhân và chia. Kích thước của một ma trận là kích thước của nó tính theo số hàng và cột theo thứ tự đó. Tức là ma trận  $2 \times 3$  có hai hàng và ba cột. Nói chung, một ma trận  $m \times n$  có  $m$  hàng và  $n$  cột. Một phần tử là một mục trong một ma trận. Cụ thể, một phần tử trong hàng  $i$  và cột  $j$  của ma trận  $A$  được ký hiệu là  $a_{i,j}$ . Cuối cùng, một vectơ trong ma trận thường được xem như một cột. Vì vậy, ma trận  $2 \times 3$  có ba vectơ (cột) mỗi vectơ có hai phần tử. Đây là một khái niệm rất quan trọng cần hiểu khi thực hiện phép nhân ma trận và/hoặc sử dụng ma trận trong các thuật toán khoa học dữ liệu.

Ví dụ mã đầu tiên tạo ra một ma trận có nhiều mảng, nhân nó với một đại lượng vô hướng, tính toán có nghĩa là theo hàng và cột, tạo một ma trận có nhiều mảng từ các mảng có nhiều mảng và hiển thị nó theo hàng và phần tử:

nhập numpy dưới dạng np

```
def mult_scalar(m, s):
    ma trận = np.empty(m.shape)
    m_shape = m.shape
    cho i, v trong liệt kê (phạm vi (m_shape [0])):
        kết quả = [x * s cho x trong m[v]]
        x = np.array(kết quả[0])
        ma trận[i] = x
    ma trận trả về
```

màn hình xác định ( $m$ ):

```
s = np.shape(m)
cols = s[1]
```

## Chương 3 Đại số tuyến tính

đối với tôi, hàng trong liệt kê (m):

```

print ('row', str(i) + ':', row, 'elements:', end=' ')
for col in range(cols):
    print (row[col], end=' ')
print ()

nếu __name__ == "__main__":
    v1, v2, v3 = [1, 7, -4], [2, -3, 10], [3, 5, 6]
    A = np.matrix([v1, v2, v3])
    print ('ma trận A:\n', A)
    vô hướng =
    0,5 B = mult_scalar(A, vô
    hướng) print ('\nma trận B:
    \n', B) mu_col = np.mean(A, axis=0,
    dtype=np.float64) print ('\nmean A (column-
    wise):\n', mu_col) mu_row = np.mean(A, axis=1,
    dtype= np.float64) print ('\nmean A (row-
    khôn ngoan):\n', mu_row) print ('\nmatrix C:')
    C = np.array([[2, 14, -8], [4, -6, 20], [6, 10, 12]])
    print (C)
    print ('\nnhiều thị mỗi hàng và phần tử:')
    màn hình (C)

```

Đầu ra:

```

matrix A:
[[ 1  7 -4]
 [ 2 -3 10]
 [ 3  5  6]]

matrix B:
[[ 0.5  3.5 -2. ]
 [ 1.   -1.5  5. ]
 [ 1.5  2.5  3. ]]

mean A (column-wise):
[[ 2.  3.  4.]]

mean A (row-wise):
[[ 1.33333333]
 [ 3.        ]
 [ 4.66666667]]

matrix C:
[[ 2 14 -8]
 [ 4 -6 20]
 [ 6 10 12]]

display each row and element:
row 0: [ 2 14 -8] elements: 2 14 -8
row 1: [ 4 -6 20] elements: 4 -6 20
row 2: [ 6 10 12] elements: 6 10 12

```

Mã bắt đầu bằng cách nhập numpy. Nó tiếp tục với hai hàm `mult_scalar()` và `display()`. Hàm `mult_scalar()` nhân một ma trận với một đại lượng vô hướng. Chức năng `display()` hiển thị một ma trận theo hàng và mỗi phần tử của một hàng. Khối chính tạo ba vectơ và thêm chúng vào ma trận numpy A. B được tạo bằng cách nhân đại lượng vô hướng 0,5 với A. Tiếp theo, giá trị trung bình của A được tính theo cột và hàng. Cuối cùng, ma trận numpy C được tạo từ ba mảng numpy và được hiển thị theo hàng và phần tử.

Ví dụ về mã thứ 2 tạo ra một ma trận gọn gàng A, tính tổng các cột của nó và các hàng, tích tích vô hướng của hai vectơ và tích tích vô hướng của hai ma trận. Tích vô hướng nhân hai vectơ để có được độ lớn có thể được sử dụng để tính toán độ dài của vectơ và góc giữa các vectơ. Cụ thể, tích vô hướng của hai vectơ a và b là  $ax \times bx + ay \times by$ .

## Chương 3 Đại số tuyến tính

Đối với phép nhân ma trận, tích vô hướng tạo ra ma trận C từ hai ma trận A và B. Tuy nhiên, không thể nhân hai vectơ khi cả hai được xem là ma trận cột. Để khắc phục sự cố này, hãy chuyển vị trí vectơ thứ nhất từ A, biến nó thành ma trận hàng  $1 \times n$  để có thể nhân nó với vectơ thứ nhất từ B và tính tổng. Tích bây giờ được xác định rõ vì tích của ma trận  $1 \times n$  với ma trận  $n \times 1$  là ma trận  $1 \times 1$  (một đại lượng vô hướng). Để lấy tích vô hướng, hãy lặp lại quy trình này cho các vectơ còn lại từ A và B. Numpy bao gồm một hàm hữu ích giúp tính tích vô hướng cho bạn, giúp đơn giản hóa rất nhiều phép nhân ma trận.

nhập numpy dưới dạng np

```
def sum_cols(ma trận):
    trả về np.sum(ma trận, trục=0)

def sum_rows(ma trận):
    trả về np.sum(ma trận, trục=1)

dưới châm chắc chắn (v, w):
    trả về np.dot(v, w)

nếu __name__ == "__main__":
    v1, v2, v3 = [1, 7, -4], [2, -3, 10], [3, 5, 6]
    A = np.matrix([v1, v2, v3])
    in ('ma trận A:\n', A)
    v_cols = sum_cols(A)
    print ('\nsum A theo cột:\n', v_cols)
    v_rows = sum_rows(A)
    in ('\nsum A theo hàng:\n', v_rows)
    dot_product = dot(v1, v2)
    in ('\nvectơ 1:', v1)
    in ('vectơ 2:', v2)
    print ('\ndot product v1 va v2:')
    in (dot_product)
```

```
v1, v2, v3 = [-2, 5, 4], [1, 2, 9], [10, -9, 3]
B = np.matrix([v1, v2, v3])
print ('\nmatrix B:\n', B)
C = A.dot(B)
print ('\nma trận C (chấm tích A và B):\n', C)
print ('\nC theo
hàng:') for i, row in
    enumerate(C): print ('row', str(i) +
': ', end='') cho v trong
        np.nditer(row):
    print (v, end=' ') print()
```

Đầu ra:

```
matrix A:
[[ 1  7 -4]
 [ 2 -3 10]
 [ 3  5  6]]

sum A by column:
[[ 6  9 12]]

sum A by row:
[[ 4]
 [ 9]
 [14]]

vector 1: [1, 7, -4]
vector 2: [2, -3, 10]

dot product v1 and v2:
-59

matrix B:
[[-2  5  4]
 [ 1  2  9]
 [10 -9  3]]

matrix C (dot product A and B):
[[-35  55  55]
 [ 93 -86  11]
 [ 59 -29  75]]

C by row:
row 0: -35 55 55
row 1: 93 -86 11
row 2: 59 -29 75
```

### Chương 3 Đại số tuyến tính

Mã bắt đầu bằng cách nhập numpy. Nó tiếp tục với ba chức năng—`sum_cols()`, `sum_rows()` và dấu chấm(). Hàm `sum_cols()` tính tổng từng cột và trả về một hàng có các giá trị này. Hàm `sum_rows()` tính tổng mỗi hàng và trả về một cột có các giá trị này. Hàm `dot()` tính toán tích vô hướng. Khối chính bắt đầu bằng cách tạo ba vectơ, sau đó được sử dụng để tạo ma trận A. Các cột và hàng được tính tổng cho A. Tích chấm sau đó được tính cho hai vectơ (`v1` và `v2`). Tiếp theo, ba vectơ mới được tạo sẽ được sử dụng để tạo ma trận B. Ma trận C được tạo bằng cách tính tích vô hướng cho A và B. Cuối cùng, mỗi hàng của C được hiển thị.

Ví dụ mã thứ 3 làm sáng tỏ một kịch bản thực tế. Giả sử một công ty bán ba loại bánh nướng - thịt bò, thịt gà và rau. Mỗi chiếc bánh nướng thịt bò có giá 3 đô la, bánh nướng gà có giá 4 đô la mỗi chiếc và bánh nướng rau có giá 2 đô la mỗi chiếc. Biểu diễn véc-tơ cho chi phí chiếc bánh là [3, 4, 2]. Bạn cũng biết doanh số bán hàng theo chiếc bánh từ thứ Hai đến thứ Năm. Doanh số bán thịt bò là 13 cho Thứ Hai, 9 cho Thứ Ba, 7 cho Thứ Tư và 15 cho Thứ Năm. Do đó, vector bán thịt bò là [13, 9, 7, 15]. Sử dụng logic tương tự, các vectơ cho doanh số bán gà lần lượt là [8, 7, 4, 6] và [6, 4, 0, 3]. Mục tiêu là tính tổng doanh số bán hàng trong bốn ngày (Thứ Hai-Thứ Năm).

nhập numpy dưới dạng np

dấu chấm chắc chắn (`v, w`):

trả về `np.dot(v, w)`

màn hình xác định (`m`):

đối với tôi, hàng trong liệt kê (`m`):

```
print ('tổng doanh thu theo ngày:\n', end=' ')
cho v trong np.nditer(hàng):
    in (v, kết thúc = ' ')
in()
```

```

nếu __name__ == "__main__":
    mёт = [3, 4, 2]
    A = np.matrix([a])
    print ('ma trận chi phí A:\n', A)
    v1, v2, v3 = [13, 9, 7, 15], [8, 7, 4, 6], [6, 4, 0, 3]
    B = np.matrix([v1, v2, v3])
    print ('\Doanh thu hàng ngày theo ma trận mặt hàng B:\n', B)
    C = A.dot(B)
    print ('\ndot product matrix C:\n', C, '\n')
    màn hình (C)

```

Đầu ra:

```

cost matrix A:
[[3 4 2]]

daily sales by item matrix B:
[[13 9 7 15]
 [ 8 7 4 6]
 [ 6 4 0 3]]

dot product matrix C:
[[83 63 37 75]]

total sales by day:
83 63 37 75

```

Mã bắt đầu bằng cách nhập numpy. Nó tiếp tục với hàm dot() tính toán tích vô hướng và hàm display() hiển thị các phần tử của ma trận, theo từng hàng. Khối chính bắt đầu bằng cách tạo một vectơ chứa chi phí của từng loại bánh. Nó tiếp tục bằng cách chuyển đổi vectơ thành ma trận A. Tiếp theo, ba vectơ được tạo đại diện cho doanh số bán hàng cho từng loại bánh từ thứ Hai đến thứ Sáu. Mã tiếp tục bằng cách chuyển đổi ba vectơ thành ma trận B. Ma trận C được tạo bằng cách tìm tích vô hướng của A và B. Kịch bản này minh họa cách sử dụng tích vô hướng để giải quyết các vấn đề kinh doanh.

## Chương 3 Đại số tuyến tính

Ví dụ mã thứ 4 tính toán độ lớn (khoảng cách) và hướng (góc) với một vectơ duy nhất và giữa hai vectơ:

nhập toán, numpy như np

```
def sqrt_sum_squares(ls):
    trả về math.sqrt(sum(map(lambda x:x*x,ls)))

def mag(v):
    trả về np.linalg.norm(v)

def a_tang(v):
    trả về math.degrees(math.atan(v[1]/v[0]))

def dist(v, w):
    return math.sqrt(((w[0]-v[0])** 2) + ((w[1]-v[1])** 2))

def mags(v, w):
    return np.linalg.norm(v - w)

def a_tangs(v, w):
    val = (w[1] - v[1]) / (w[0] - v[0])
    trả về math.degrees(math.atan(val))

if __name__ == "__main__":
    v = np.array([3, 4])
    print ('vectơ đơn', str(v) + ':') print
    ('độ lớn:', sqrt_sum_squares(v)) print
    ('Độ lớn NumPY:', mag(v)) print
    ('hướng:', round(a_tang(v)), 'độ\n') v1, v2 =
    np.array([2, 3]), np.array ([5, 8]), print,
    ('hai vectơ', str(v1) + + str(v2) + Và') print ('độ lớn',
    round(dist(v1, v2),2)) print (' Độ lớn
    NumPY:', round(mags(v1, v2),2)) print ('hướng:',
    round(a_tangs(v1, v2)), 'độ\n') v1, v2 = np.array([0 , 0]),
    np.array([3, 4])
```

```
print('dùng gốc tọa độ (0,0) làm vec tơ thứ nhất:')
print("hai vecto", str(v1) + + str(v2) + '')
in ('độ lớn:', tròn(mags(v1, v2),2))
print ('hướng:', round(a_tangs(v1, v2)), 'độ')
```

Đầu ra:

```
single vector [3 4]:
magnitude: 5.0
NumPY magnitude: 5.0
direction: 53 degrees

two vectors [2 3] and [5 8]:
magnitude 5.83
NumPY magnitude: 5.83
direction: 59 degrees

use origin (0,0) as 1st vector:
"two vectors [0 0] and [3 4]"
magnitude: 5.0
direction: 53 degrees
```

Mã bắt đầu bằng cách nhập các thư viện toán học và numpy. Nó tiếp tục với sáu chức năng. Hàm `sqrt_sum_squares()` tính toán độ lớn cho một vectơ từ đầu. Hàm `mag()` cũng làm như vậy nhưng sử dụng numpy. Hàm `a_tang()` tính toán `arctan` của một vectơ, là hướng (góc) của một vectơ tính từ gốc tọa độ  $(0,0)$ . Hàm `dist()` tính toán độ lớn giữa hai vectơ từ đầu. Hàm `mags()` cũng làm như vậy nhưng sử dụng numpy. Hàm `a_tangs()` tính toán `arctan` của hai vectơ. Khối chính tạo ra một vectơ, tính toán độ lớn và hướng và hiển thị. Tiếp theo, cưỡng độ và hướng được tính toán và hiển thị cho hai vectơ. Cuối cùng, độ lớn và hướng của một vectơ duy nhất được tính bằng hai công thức vectơ. Điều này được thực hiện bằng cách sử dụng gốc tọa độ  $(0,0)$  làm vectơ thứ nhất. Vì vậy, các hàm tính toán độ lớn và hướng cho một vectơ đơn lẻ là không cần thiết, bởi vì bất kỳ vectơ đơn lẻ nào luôn bắt đầu từ gốc tọa độ  $(0,0)$ . Do đó, một vectơ chỉ đơn giản là một điểm trong không gian được đo từ gốc tọa độ  $(0,0)$  hoặc liên quan đến một vectơ khác theo độ lớn và hướng.

## Chương 3 Đại số tuyến tính

## Phép biến đổi ma trận cơ bản

Ví dụ mã đầu tiên giới thiệu ma trận đơn vị, là một ma trận vuông với các số 1 trên đường chéo chính và các số 0 ở những nơi khác. Tích của ma trận A và ma trận đơn vị của nó là A, điều này rất quan trọng về mặt toán học vì thuộc tính đơn vị của phép nhân cho biết rằng bất kỳ số nào nhân với 1 đều bằng chính nó.

nhập numpy dưới dạng np

```
def slice_row(M, i): trả
    về M[i,:]

def slice_col(M, j): trả
    lại M[:, j]

def to_int(M):
    trả về M.astype(np.int64)

nếu __name__ == "__main__":
    A =
        [[1, 9, 3, 6, 7], [4, 8,
            6, 2, 1], [9, 8, 7,
            1, 2], [1, 1, 9, 2,
            4], [9, 1, 1, 3, 5]]

    A = np.matrix(A)
    print ('A:\n', A)
    print ('\nhàng thứ nhất: ', slice_row(A, 0))
    print ('\ncột thứ 3:\n', slice_col(A, 2)) shapeA =
    np.shape(A)
    Tôi = np.identity(np.shape(A)[0])
    I = to_int(I)
    print ('\nI:\n', I)
    dot_product = np.dot(A, I)
```

```
in ('`nA * I = A`\n', dot_product)
in ('`nA`\n', AI)
A_by_Ainv = np.round(np.dot(A, AI), số thập phân=0, ngoài=Không)
A_by_Ainv = to_int(A_by_Ainv)
in ('`nA * A`\n', A_by_Ainv)
```

Đầu ra:

```
A:
[[1 9 3 6 7]
 [4 8 6 2 1]
 [9 8 7 1 2]
 [1 1 9 2 4]
 [9 1 1 3 5]]

1st row: [[1 9 3 6 7]]

3rd column:
[3]
[6]
[7]
[9]
[1]

I:
[[1 0 0 0 0]
 [0 1 0 0 0]
 [0 0 1 0 0]
 [0 0 0 1 0]
 [0 0 0 0 1]]

A * I = A:
[[1 9 3 6 7]
 [4 8 6 2 1]
 [9 8 7 1 2]
 [1 1 9 2 4]
 [9 1 1 3 5]]

A':
[[-6.12745098e-02   6.90144479e-02  -8.77192982e-03  -2.97987616e-02
  9.93292054e-02]
 [ 8.82352941e-02  -1.09907121e-01   1.57894737e-01  -6.65634675e-02
 -1.11455108e-01]
 [-5.63725490e-02   9.50722394e-02  -4.388596491e-02   1.01006192e-01
 -3.35397317e-03]
 [-1.61764706e-01   8.24303406e-01  -6.84210526e-01  -7.73993800e-04
 3.35913313e-01]
 [ 2.00980392e-01  -6.15841073e-01   4.03508772e-01   4.72136223e-02
 -1.57378741e-01]]

A * A':
[[1 0 0 0 0]
 [0 1 0 0 0]
 [0 0 1 0 0]
 [0 0 0 1 0]
 [0 0 0 0 1]]
```

Mã bắt đầu bằng cách nhập numpy. Nó tiếp tục với ba chức năng. Hàm slice\_row() cắt một hàng từ ma trận. Hàm slice\_col() cắt một cột từ ma trận. Hàm to\_int() chuyển đổi ma trận

## Chương 3 Đại số tuyến tính

phần tử thành số nguyên. Khối chính bắt đầu bằng cách tạo ma trận A. Nó tiếp tục bằng cách tạo ma trận đồng nhất cho A. Cuối cùng, nó tạo ma trận đồng nhất cho A bằng cách sử dụng tích vô hướng của A với A' (nghịch đảo của A).

Ví dụ mã thứ 2 chuyển đổi một danh sách các danh sách thành một ma trận có nhiều mảng và đi qua nó:

nhập numpy dưới dạng np

```
nếu __name__ == "__main__":
    = [ [41,
        72, 180], [27, 66, 140], [18, 59,
        101], [57, 72, 160], [21, 59, 112] ,
        [29, 77, 250], [55, 60, 120], [28,
        72, 110], [19, 59, 99], [32, 68,
        125], [31, 79, 322], [ 36, 69, 111] ]
```

```
A = np.matrix(data)
print ('traversal thủ công: ')
for p in range(A.shape[0]):
    cho q trong dãy(A.shape[1]): print
        (A[p,q], end=' ') print ()
```

Đầu ra:

```
manual traversal:
41 72 180
27 66 140
18 59 101
57 72 160
21 59 112
29 77 250
55 60 120
28 72 110
19 59 99
32 68 125
31 79 322
36 69 111
```

Mã bắt đầu bằng cách nhập numpy. Khối chính bắt đầu bằng cách tạo một danh sách các danh sách, chuyển đổi nó thành ma trận có nhiều mảng A và duyệt qua A. Mặc dù tôi đã trình bày một số phương pháp để duyệt qua một ma trận có nhiều mảng, nhưng đây là phương pháp yêu thích của tôi.

Ví dụ mã thứ 3 chuyển đổi danh sách các danh sách thành ma trận numpy A. Sau đó, nó cắt và xác xác A:

nhập numpy dưới dạng np

```

nếu __name__ == "__main__":
    điểm_3D_space = [ [0, 0,
                      0], [1, 2,
                            3], [2, 2,
                                  2], [9, 9,
                                        9] ]

    A = np.matrix(điểm_3D_space) print ('cắt
    toàn bộ A:') print (A[:]) print
    ('\nslice cột
    thứ 2:') print (A[0:4, 1]) print
    ('\nslice Cột thứ 2
    (phương thức thay thế):') print (A[:, 1]) print ('\nslice
    Giá trị thứ 2 & 3
    Cột thứ 3:') print (A[1:3, 2]) print ('\nslice hàng cuối
    cùng :') print (A[-1])
    print ('\nslice hàng cuối cùng
    (phương thức thay
    thế):') print (A[3]) print ('\nslice hàng đầu tiên:')
    print (A[0, :])
    print ('\nslice hàng thứ 2; giá
    trị thứ 2 & thứ 3:')
    print (A[1, 1:3])

```

## Chương 3 Đại số tuyến tính

Đầu ra:

```

slice entire A:
[[0 0 0]
 [1 2 3]
 [2 2 2]
 [9 9 9]]

slice 2nd column:
[[0]
 [2]
 [2]
 [9]]

slice 2nd column (alt method):
[[0]
 [2]
 [2]
 [9]]

slice 2nd & 3rd value 3rd column:
[[3]
 [2]]

slice last row:
[[9 9 9]]

slice last row (alt method):
[[9 9 9]]

slice 1st row:
[[0 0 0]]

slice 2nd row; 2nd & 3rd value:
[[2 3]]
```

Mã bắt đầu bằng cách nhập numpy. Khối chính bắt đầu bằng cách tạo một danh sách các danh sách và chuyển đổi nó thành ma trận A. Mã tiếp tục bằng cách cắt và chia khối ma trận.

## Ứng dụng ma trận gấu trúc

Thư viện pandas cung cấp các công cụ phân tích và cấu trúc dữ liệu hiệu suất cao, dễ sử dụng. Đối tượng gấu trúc được sử dụng phổ biến nhất là DataFrame (df). df là cấu trúc 2-D với các trục (hàng và cột) được gắn nhãn thuộc các loại có khả năng khác nhau. Các phép toán căn chỉnh trên cả nhãn hàng và nhãn cột. Một df có thể được khai niêm hóa theo cột hoặc hàng. Để xem bởi

cột, hãy sử dụng `axis = 0` hoặc `axis = 'index'`. Để xem theo hàng, hãy sử dụng `axis = 1` hoặc `axis = 'columns'`. Điều này có vẻ phản trực giác khi làm việc với các hàng, nhưng đây là cách giao trúc triển khai tính năng này.

Pandas df dễ làm việc hơn nhiều so với ma trận numpy, nhưng nó cũng kém hiệu quả hơn. Nghĩa là, cần nhiều tài nguyên hơn để xử lý pandas df. Thư viện numpy được tối ưu hóa để xử lý lượng lớn dữ liệu và tính toán số.

Ví dụ đầu tiên tạo một danh sách các danh sách, đặt nó vào pandas df và hiển thị một số dữ liệu:

nhập giao trúc dưới dạng pd

```
nếu __name__ == "__main__":
    data =
        [ [41, 72,
            180], [27, 66, 140], [18, 59, 101],
            [57, 72, 160], [21, 59, 112] , [29,
            77, 250], [55, 60, 120], [28, 72, 110],
            [19, 59, 99], [32, 68, 125], [31, 79,
            322], [ 36, 69, 111] ]
```

```
tiêu đề = ['tuổi', 'chiều cao', 'cân nặng']
df = pd.DataFrame(dữ liệu, cột=tiêu đề)
n = 3
print ('Hàng đầu tiên', n, ''df'':\n', df.head(n)) print ('\nHàng
"df" đầu tiên:') print (df[0:1])
print ('\nHàng 2
đến 4') print (df[2:5]) print
('First', n, 'hàng
"tuổi" cột') print (df[['age']].head(n)) print
('Last', n, 'cột "trọng lượng" và
"tuổi" của hàng') print (df[['weight', 'age']].tail(n))
```

## Chương 3 Đại số tuyến tính

```
print ('\nHàng từ 3 đến 6 cột "trọng lượng" và "tuổi"')
in (df.ix[3:6, ['trọng lượng', 'tuổi']])
```

Đầu ra:

```
First 3 "df" rows:
   age  height  weight
0    41       72     180
1    27       66     140
2    18       59     101

First "df" row:
   age  height  weight
0    41       72     180

Rows 2 through 4
   age  height  weight
2    18       59     101
3    57       72     160
4    21       59     112

First 3 rows "age" column
   age
0    41
1    27
2    18

Last 3 rows "weight" and "age" columns
   weight  age
9      125  32
10     322  31
11     111  36

Rows 3 through 6 "weight" and "age" columns
   weight  age
3      160  57
4      112  21
5      250  29
6      120  55
```

Mã bắt đầu bằng cách nhập pandas. Khối chính bắt đầu bằng cách tạo một danh sách các danh sách và thêm nó vào pandas df. Bạn nên tạo các tiêu đề của riêng mình như chúng tôi làm ở đây. Phương thức head() và tail() tự động hiển thị năm bản ghi đầu tiên và năm bản ghi cuối cùng tương ứng trừ khi một giá trị được đưa vào. Trong trường hợp này, chúng tôi hiển thị ba bản ghi đầu tiên và cuối cùng. Sử dụng head() và tail() rất hữu ích, đặc biệt với df lớn. Lưu ý cách dễ dàng để cắt và xúc xác df. Ngoài ra, hãy lưu ý việc hiển thị dữ liệu cột bạn chọn dễ dàng như thế nào.

## Chương 3 Đại số tuyến tính

Ví dụ thứ 2 tạo một danh sách các danh sách, đặt nó vào ma trận numpy A và đặt A vào pandas df. Khả năng này rất quan trọng vì nó cho thấy việc tạo một df từ một ma trận khó hiểu dễ dàng như thế nào. Vì vậy, bạn có thể làm việc với ma trận numpy để đạt được độ chính xác và hiệu suất, sau đó chuyển đổi thành pandas để cắt, thái hạt lựu và các thao tác khác.

nhập giao trúc dưới dạng pd, numpy dưới dạng np

```
nếu __name__ == "__main__": data =
    [ [41, 72,
        180], [27, 66, 140], [18, 59, 101], [57,
        72, 160], [21, 59, 112] , [29, 77, 250],
        [55, 60, 120], [28, 72, 110], [19, 59,
        99], [32, 68, 125], [31, 79, 322], [ 36,
        69, 111] ]
```

```
A = np.matrix(data) headers
= ['tuổi', 'chiều cao', 'trọng lượng'] df =
pd.DataFrame(A, cột=tiêu đề) print ('Toàn bộ "df":')
print (df, '\n') print ('Được
cắt theo "tuổi" và
"chiều cao":') print (df[['tuổi', 'chiều cao']])
```

## Chương 3 Đại số tuyến tính

Đầu ra:

```
Entire "df":
   age  height  weight
0    41      72     180
1    27      66     140
2    18      59     101
3    57      72     160
4    21      59     112
5    29      77     250
6    55      60     120
7    28      72     110
8    19      59      99
9    32      68     125
10   31      79     322
11   36      69     111

Sliced by "age" and "height":
   age  height
0    41      72
1    27      66
2    18      59
3    57      72
4    21      59
5    29      77
6    55      60
7    28      72
8    19      59
9    32      68
10   31      79
11   36      69
```

Mã bắt đầu bằng cách nhập pandas và numpy. Khối chính bắt đầu bằng cách tạo một danh sách các danh sách, chuyển đổi nó thành ma trận gọn gàng A, sau đó thêm A vào pandas df.

Ví dụ thứ 3 tạo một danh sách các danh sách, đặt nó vào một danh sách các thành phần từ điển và đặt nó vào một pandas df. Khả năng này cũng rất quan trọng vì từ điển là cấu trúc dữ liệu rất hiệu quả khi làm việc với các ứng dụng khoa học dữ liệu.

nhập giao trúc dưới dạng pd

```
nếu __name__ == "__main__": data =
[ [41, 72,
  180], [27, 66, 140], [18, 59, 101],
  [57, 72, 160], [21, 59, 112] , [29,
  77, 250], [55, 60, 120], [28, 72, 110],
  [19, 59, 99], [32, 68, 125], [31, 79,
  322], [ 36, 69, 111] ] d = {} dls =
[] key = ['tuổi', 'chiều cao', 'cân
```

nặng']

cho hàng

trong dữ liệu:

```
cho i, num trong liệt kê(hàng):
    d[key[i]] = num
dls.append(d) d
= {} df
= pd.DataFrame(dls) print
('dict các phần tử từ danh sách:') cho hàng
trong dls:
    print (hàng)
print ('\nhigh từ phần tử dict đầu tiên là:', end=' ') print (dls[0]
['height']) print ('\n"df" được
chuyển đổi từ danh sách dict:\n' , df) print ('\nhigh phần tử df
thứ nhất:\n', df[['height']].head(1))
```

## Chương 3 Đại số tuyến tính

Đầu ra:

```

dict elements from list:
{'age': 41, 'height': 72, 'weight': 180}
{'age': 27, 'height': 66, 'weight': 140}
{'age': 18, 'height': 59, 'weight': 101}
{'age': 57, 'height': 72, 'weight': 160}
{'age': 21, 'height': 59, 'weight': 112}
{'age': 29, 'height': 77, 'weight': 250}
{'age': 55, 'height': 60, 'weight': 120}
{'age': 28, 'height': 72, 'weight': 110}
{'age': 19, 'height': 59, 'weight': 99}
{'age': 32, 'height': 68, 'weight': 125}
{'age': 31, 'height': 79, 'weight': 322}
{'age': 36, 'height': 69, 'weight': 111}

height from 1st dict element is: 72

"df" converted from dict list:
   age  height  weight
0    41      72     180
1    27      66     140
2    18      59     101
3    57      72     160
4    21      59     112
5    29      77     250
6    55      60     120
7    28      72     110
8    19      59      99
9    32      68     125
10   31      79     322
11   36      69     111

height 1st df element:
    height
0      72

```

Ví dụ mã thứ 4 tạo hai danh sách gồm các danh sách-dữ liệu và điểm số.

Danh sách dữ liệu chứa tuổi, chiều cao và cân nặng của 12 vận động viên. Danh sách điểm gồm 3 điểm thi của 12 học sinh. Danh sách dữ liệu được đưa trực tiếp vào df1 và danh sách điểm được đưa trực tiếp vào df2. Trung bình được tính toán và hiển thị.

nhập gấu trúc dưới dạng pd, numpy dưới dạng np

```

nếu __name__ == "__main__":
    dữ liệu = [
        [41, 72, 180], [27, 66, 140],
        [18, 59, 101], [57, 72, 160],
        [21, 59, 112], [29, 77, 250],

```

```
[55, 60, 120], [28, 72, 110],  
[19, 59, 99], [32, 68, 125],  
[31, 79, 322], [36, 69,  
  
111] ]  
điểm = [ [99, 90, 88], [77, 66, 81], [78,  
77, 83], [75, 72, 79], [88, 77, 93], [88,  
77, 94], [ 100, 99, 93], [94, 74, 90], [98,  
97, 99], [73, 68, 77], [55, 50, 68], [36,  
77, 90] ]  
n =  
3 key1 = ['tuổi', 'chiều cao', 'cân  
nặng'] df1 = pd.DataFrame(dữ liệu,  
cột=key1) print ('df1 lát:\n',  
df1.head(n)) avg_cols =  
df1 .apply(np.mean, axis=0)  
print ('\naverage  
theo cột:') print (avg_cols) avg_wt =  
df1[['weight']].apply(np.mean,  
axis='index')  
print ( '\naverage weight') print  
(avg_wt) key2 = ['exam1', 'exam2',  
'exam3'] df2 = pd.DataFrame(điểm,  
cột=key2) print ('\ndf2 lát:\n', df2.  
head(n)) avg_scores = df2.apply(np.mean, axis=1) print  
('\naverage score for 1st',  
n, 'students ( rows):') print (avg_scores.head(n))  
avg_slice =  
df2[['exam1','exam3']].apply(np.mean, axis='columns')  
print  
('\naverage "exam1" & "exam3" 1st', n, 'students ( rows):') in (avg_s
```

## Chương 3 Đại số tuyến tính

Đầu ra:

```

df1 slice:
   age  height  weight
0    41      72     180
1    27      66     140
2    18      59     101

average by columns:
age            32.833333
height         67.666667
weight        152.500000
dtype: float64

average weight
weight       152.5
dtype: float64

df2 slice:
   exam1  exam2  exam3
0      99      90      88
1      77      66      81
2      78      77      83

average scores for 1st 3 students (rows):
0    92.333333
1    74.666667
2    79.333333
dtype: float64

average "exam1" & "exam3" 1st 3 students (rows):
0    93.5
1    79.0
2    80.5
dtype: float64

```

Mã bắt đầu bằng cách nhập pandas và numpy. Khởi chính tạo danh sách dữ liệu và điểm số và đặt chúng vào df1 và df2 tương ứng.

Với df1 (dữ liệu), chúng tôi tính trung bình theo cột vì mục tiêu của chúng tôi là trả về tuổi, chiều cao và cân nặng trung bình cho tất cả các vận động viên. Với df2 (điểm), chúng tôi tính trung bình theo hàng vì mục tiêu của chúng tôi là trả về điểm bài kiểm tra tổng thể trung bình cho mỗi học sinh. Chúng tôi có thể tính trung bình theo cột cho df2 nếu mục tiêu là tính tổng điểm trung bình cho một trong các bài kiểm tra. Hãy thử điều này nếu bạn muốn.

## CHƯƠNG 4

# Xuống dốc

Độ dốc gốc (GD) là một thuật toán giảm thiểu (hoặc tối đa hóa) các chức năng. Để áp dụng, hãy bắt đầu từ tập hợp ban đầu các giá trị tham số của hàm và lặp đi lặp lại di chuyển về phía tập hợp các giá trị tham số thu nhỏ hàm. Giảm thiểu lặp đi lặp lại đạt được bằng cách sử dụng phép tính bằng cách thực hiện các bước theo hướng âm của độ dốc của hàm. GD rất quan trọng vì tối ưu hóa là một phần quan trọng của máy học. Ngoài ra, GD dễ thực hiện, chung chung và hiệu quả (nhanh).

## Tối thiểu hóa chức năng đơn giản (và tối đa hóa)

GD là thuật toán tối ưu hóa lặp bậc 1 để tìm giá trị cực tiểu của hàm  $f$ . Một hàm có thể được ký hiệu là  $f$  hoặc  $f(x)$ . Đơn giản, GD tìm lỗi nhỏ nhất bằng cách giảm thiểu (hoặc tối đa hóa) hàm chi phí. Một chức năng chi phí là một cái gì đó mà bạn muốn giảm thiểu.

Hãy bắt đầu với một ví dụ tối thiểu hóa. Để tìm giá trị cực tiểu địa phương của  $f$ , thực hiện các bước tỷ lệ với âm của gradient của  $f$  tại điểm hiện tại. Độ dốc là đạo hàm (tốc độ thay đổi) của  $f$ . Điểm yếu nhất của GD là nó tìm giá trị cực tiểu cục bộ thay vì giá trị cực tiểu cho toàn hàm.

## Chương 4 Đồ thị độ dốc

Quy tắc lũy thừa được sử dụng để phân biệt các hàm có dạng  $f(x) = x^n$  :

$$\frac{d}{dx} x^n = nx^{n-1}$$

Vì vậy, đạo hàm của  $x^n$  bằng  $nx^{n-1}$ . Đơn giản, đạo hàm là tích của số mũ nhân với  $x$  với số mũ giảm đi 1. Để giảm thiểu  $f(x) = x^4 - 3x^3 + 2$ , hãy tìm đạo hàm là  $f'(x) = 4x^3 - 9x^2$ . Vì vậy, bước đầu tiên luôn là tìm đạo hàm  $f'(x)$ . Bước thứ 2 là vẽ đồ thị hàm ban đầu để hình dung về hình dạng của nó. Bước thứ 3 là chạy GD. Bước thứ 4 là vẽ đồ thị mức tối thiểu cục bộ.

Ví dụ đầu tiên tìm điểm cực tiểu cục bộ của  $f(x)$  và hiển thị  $f(x)$ ,  $f'(x)$ , và tối thiểu trong ô con như trong Hình 4-1:

nhập `matplotlib.pyplot` dưới dạng `plt`, `numpy` dưới dạng `np`

```
def f(x):
    trả về x**4 - 3 * x**3 + 2

def df(x):
    trả về 4 * x**3 - 9 * x**2

nếu __name__ == "__main__":
    x = np.arange(-5, 5, 0.2)
    y_dx = f(x), df(x)
    axarr = plt.subplots(3, sharex=True)
    axarr[0].plot(x, y_dx, color='mediumspringgreen')
    axarr[0].set_xlabel('x')
    axarr[0].set_ylabel('f(x)')
    axarr[0].set_title('f(x)')
    axarr[1].plot(x, y_dx, color='salmon')
    axarr[1].set_xlabel('x')
    axarr[1].set_ylabel('dy/dx(x)')
    axarr[1].set_title('đạo hàm của f(x)')
    axarr[2].set_xlabel('x')
    axarr[2].set_ylabel('GD')
```

```

axarr[2].set_title('local minimum')
lặp lại, cur_x, gamma, precision = 0, 6, 0,01, 0,00001
previous_step_size = cur_x
trong khi previous_step_size > precision:
    trước_x = cur_x
    cur_x += -gamma * df(trước_x)
    previous_step_size = abs(cur_x - prev_x)
    lần lặp += 1
    axarr[2].plot(prev_x, cur_x, "o")
f.subplots_adjust(hspace=0.3)
f.tight_layout()
plt.show()
print ('minimum:', cur_x, '\niterations:', iterations)

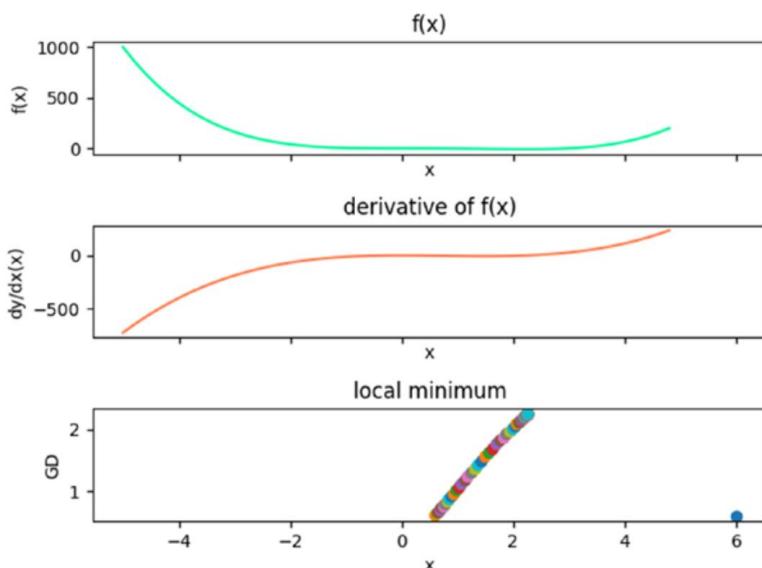
```

Đầu ra:

```

minimum: 2.2499646074278457
iterations: 70

```



Hình 4-1. Trục quan hóa ô con của  $f(x)$ ,  $f'(x)$  và điểm cực tiểu cục bộ

## Chương 4 Đốc độ dốc

Ví dụ mã bắt đầu bằng cách nhập `matplotlib` và `numpy`. Nó tiếp tục với hàm  $f(x)$  được sử dụng để vẽ đồ thị của hàm gốc và hàm  $df(x)$  được sử dụng để vẽ đạo hàm. Khối chính bắt đầu bằng cách tạo các giá trị cho  $f(x)$ . Nó tiếp tục bằng cách tạo một subplot. GD bắt đầu bằng cách khởi tạo các biến. Biến `cur_x` là điểm bắt đầu cho mô phỏng. Biến `gamma` là kích thước bước. Độ chính xác thay đổi là dung sai. Dung sai nhỏ hơn chuyển thành độ chính xác cao hơn, nhưng yêu cầu nhiều lần lặp lại (tài nguyên). Quá trình mô phỏng tiếp tục cho đến khi `previous_step_size` lớn hơn độ chính xác. Mỗi lần lặp lại nhân `-gamma` (`step_size`) với độ dốc (đạo hàm) tại điểm hiện tại để di chuyển nó đến mức tối thiểu cục bộ.

Biến `previous_step_size` sau đó được gán chênh lệch giữa `cur_x` và `prev_x`. Mỗi điểm được vẽ. Giá trị nhỏ nhất để giải  $f(x)$  tìm  $x$  là xấp xỉ 2,25. Tôi biết kết quả này là chính xác vì tôi đã tính toán bằng tay. Kiểm tra <http://www.dummies.com/education/math/calculus/cach-tim-dia-phuong-cuc-voi-thu-thu-dao-thi-dau-tien/> cho một bài học hay về cách tính toán bằng tay.

Ví dụ thứ 2 tìm giá trị cực đại và cực tiểu cục bộ của  $f(x) = x^3 - 6x^2 + 9x + 15$ . Đầu tiên, tìm  $f'(x)$ , là  $3x^2 - 12x + 9$ . Tiếp theo, tìm cực tiểu cục bộ, đồ thị, cực đại cục bộ, và âm mưu. Tôi không sử dụng ô con trong trường hợp này vì hình ảnh không phong phú bằng. Nghĩa là, sẽ dễ dàng hơn nhiều để thấy mức tối thiểu và mức tối đa cục bộ gần đúng bằng cách xem biểu đồ của  $f(x)$  và dễ dàng hơn để xem quy trình GD hoạt động kỳ diệu như thế nào.

nhập `matplotlib.pyplot` dưới dạng `plt`, `numpy` dưới dạng `np`

xác định  $f(x)$ :

trả về  $x^{**3} - 6 * x^{**2} + 9 \quad * \quad x + 15$

xác định  $df(x)$ :

trở lại  $3 * x^{**2} - 12 * \quad x + 9$

nếu `__name__ == "__main__"`:

`x = np.arange(-0,5, 5, 0,2)`

`y = f(x)`

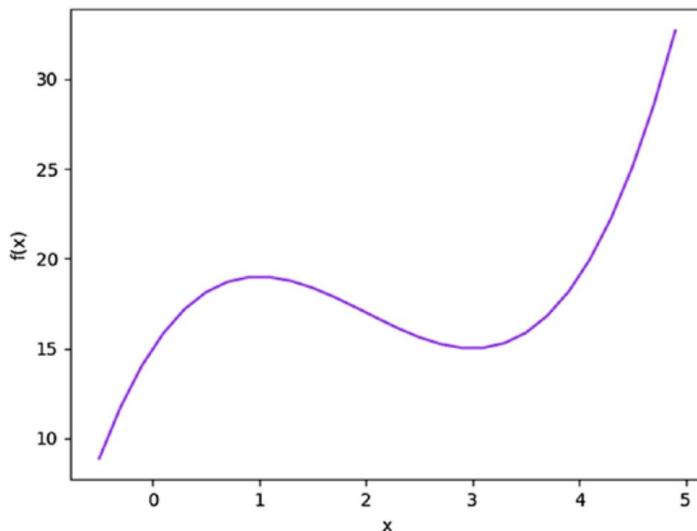
`plt.figure('f(x)')`

```
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('f(x)')
plt.plot(x, y, color='blueviolet')
plt.figure('local tối thiểu')
plt.xlabel('x')
plt.ylabel('GD')
plt.title('tối thiểu cục bộ') lặp lại, cur_x, gamma, precision = 0, 6, 0,01, 0,00001
previous_step_size = cur_x
trong khi previous_step_size > precision :
    trước_x = cur_x
    cur_x += -gamma * df(trước_x)
    previous_step_size = abs(cur_x - prev_x)
    lần lặp += 1
    plt.plot(prev_x, cur_x, "o")
local_min = cur_x
print ('tối thiểu:', local_min, 'số lần lặp:', số lần lặp)
plt.figure('tối đa cục bộ')
plt.xlabel('x')
plt. lặp lại
ylabel('GD') plt.title('tối đa cục bộ'), cur_x, gamma, precision = 0, 0.5, 0.01, 0.00001
previous_step_size = cur_x
while previous_step_size > precision:
    trước_x = cur_x
    cur_x += -gamma * -df(prev_x)
    previous_step_size = abs(cur_x - prev_x)
    lần lặp += 1
    plt.plot(prev_x, cur_x, "o")
local_max = cur_x
print ('maximum:', local_max, 'iterations:', iterations)
plt.show()
```

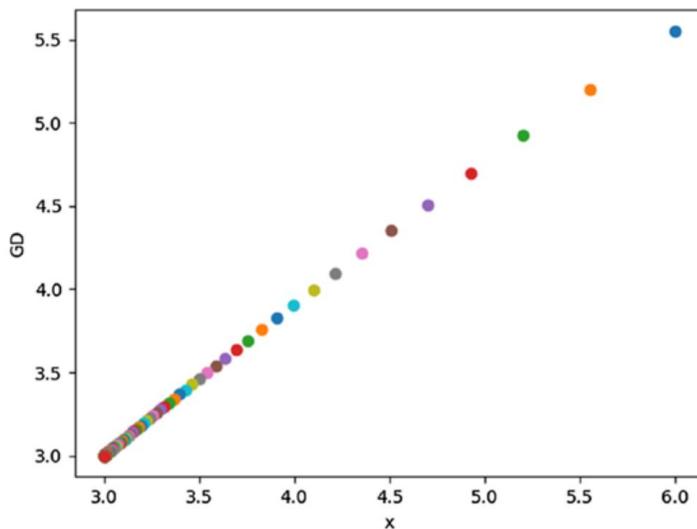
## Chương 4 Dốc độ dốc

Đầu ra:

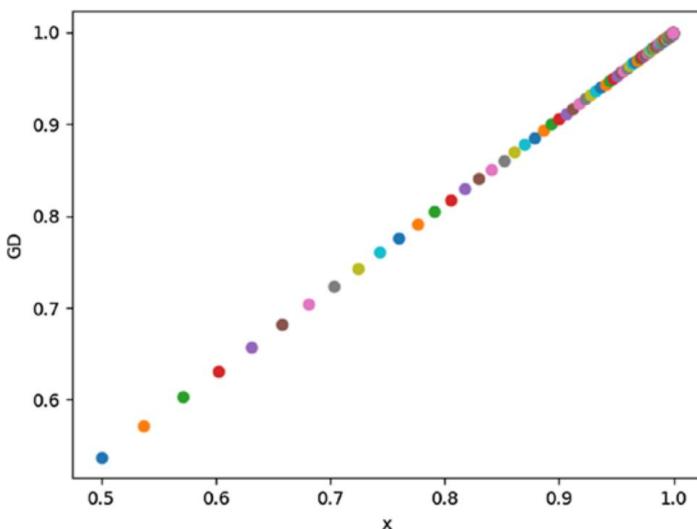
```
minimum: 3.0001526323101704 iterations: 144
maximum: 0.9998475518984531 iterations: 127
```



Hình 4-2. Hàm  $f(x)$



Hình 4-3. Cực tiểu cục bộ của hàm  $f(x)$

Hình 4-4. Cực đại cục bộ của hàm  $f(x)$ 

Mã bắt đầu bằng cách nhập các thư viện `matplotlib` và `numpy`. Nó tiếp tục với các hàm  $f(x)$  và  $df(x)$ , đại diện cho hàm gốc và đạo hàm của nó theo thuật toán. Khối chính bắt đầu bằng cách tạo dữ liệu cho  $f(x)$  và vẽ đồ thị cho nó. Nó tiếp tục bằng cách tìm cực tiểu và cực đại cục bộ, và vẽ chúng. Lưu ý `circ_x` (điểm bắt đầu) cho mức tối thiểu cục bộ là 6, trong khi đó là 0,5 cho mức tối đa cục bộ.

Đây là nơi khoa học dữ liệu mang tính nghệ thuật hơn là khoa học, bởi vì tôi đã tìm ra những điểm này bằng cách thử và sai. Cũng lưu ý rằng GD cho cực đại cục bộ là phủ định của đạo hàm. Một lần nữa, tôi biết rằng kết quả là chính xác vì tôi đã tính toán cả mức tối thiểu và tối đa cục bộ bằng tay. Lý do chính mà tôi sử dụng các ô riêng biệt thay vì ô phụ cho ví dụ này là để chứng minh tại sao việc vẽ đồ thị  $f(x)$  lại quan trọng đến vậy. Chỉ cần nhìn vào đồ thị, bạn có thể biết rằng cực đại cục bộ của  $x$  đối với  $f(x)$  gần bằng 1 và cực tiểu cục bộ của  $x$  đối với  $f(x)$  gần bằng 3. Ngoài ra, bạn có thể thấy rằng hàm có tổng mức tối đa lớn hơn 1 từ biểu đồ này. Hình 4-2, [4-3](#) và [4-4](#) cung cấp các hình ảnh trực quan.

Chương 4 Đốc độ đốc

## Giảm thiểu chúc năng sigmoid (và tối đa hóa)

Hàm sigmoid là một hàm toán học có đường cong hình chữ S hoặc sigmoid. Nó rất quan trọng trong khoa học dữ liệu vì nhiều lý do. Đầu tiên, nó có thể dễ dàng phân biệt đối với các tham số mạng, là yếu tố then chốt trong việc đào tạo mạng lưới thần kinh. Thứ hai, các hàm phân phối tách lũy cho nhiều phân phối xác suất phổ biến là sigmoidal. Thứ ba, nhiều quá trình tự nhiên (ví dụ: các đường cong học tập phức tạp) tuân theo một đường cong hình sigma theo thời gian. Vì vậy, một hàm sigmoid thường được sử dụng nếu không có sẵn mô hình toán học cụ thể.

Ví dụ đầu tiên tìm giá trị cực tiểu cục bộ của hàm sigmoid:

nhập matplotlib.pyplot dưới dạng plt, numpy dưới dạng np

```
def sigmoid(x):
    trả về 1/(1 + np.exp(-x))

    xác định df(x):
        trả lại x * (1-x)

nếu __name__ == "__main__":
    x = np.arange(-10., 10., 0.2)
    y, y_dx = sigmoid(x), df(x)

    f, axarr = plt.subplots(3, sharex=True)
    axarr[0].plot(x, y, color='lime')
    axarr[0].set_xlabel('x')
    axarr[0].set_ylabel('f(x)')
    axarr[0].set_title('Hàm Sigmoid')

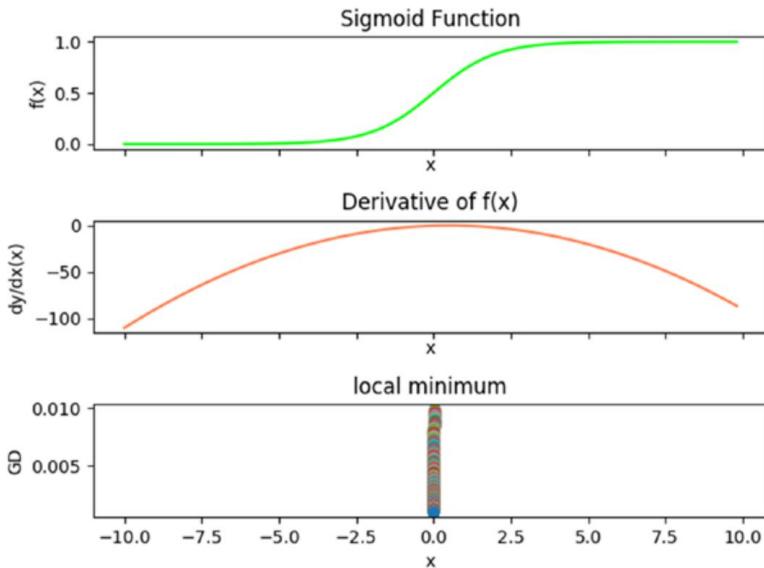
    axarr[1].plot(x, y_dx, color='san hô')
    axarr[1].set_xlabel('x')
```

```
axarr[1].set_ylabel('dy/dx(x)')
axarr[1].set_title('Derivative of f(x)')
axarr[2].set_xlabel('x')
axarr[2].set_ylabel( 'GD')
axarr[2].set_title('local minimum')
iterations, cur_x, gamma, precision = 0, 0,01, 0,01, 0,00001
previous_step_size = cur_x
while previous_step_size > precision:
    trước_x = cur_x
    cur_x += -gamma * df(trước_x)
    previous_step_size = abs(cur_x - prev_x)
    lần lặp += 1
    plt.plot(prev_x, cur_x, "o")
f.subplots_adjust(hspace=0.3)
f.tight_layout()
print ('minimum:', cur_x, '\niterations:', iterations)
plt.show()
```

## Chương 4 Dốc độ dốc

Đầu ra:

```
minimum: 0.0009901574660713482
iterations: 231
```



Hình 4-5. Ô con của  $f(x)$ ,  $f'(x)$  và cực tiểu địa phương

Mã bắt đầu bằng cách nhập `matplotlib` và `numpy`. Nó tiếp tục với các hàm `sigmoid(x)` và `df(x)`, đại diện cho hàm sigmoid và đạo hàm của nó theo thuật toán. Khối chính bắt đầu bằng cách tạo dữ liệu cho  $f(x)$  và  $f'(x)$ . Nó tiếp tục bằng cách tạo các ô con cho  $f(x)$ ,  $f'(x)$  và điểm cực tiểu cục bộ. Trong trường hợp này, việc sử dụng các ô con là tốt để trực quan hóa. Để dàng thấy từ các đồ thị  $f(x)$  và  $f'(x)$  (Hình 4-5) rằng cực tiểu cục bộ gần bằng 0. Tiếp theo, đoạn mã chạy GD để tìm cực tiểu cục bộ và vẽ đồ thị cho nó.

Một lần nữa, điểm khởi đầu cho GD,  $cur\_x$ , được tìm thấy bằng phương pháp thử và sai. Nếu bạn bắt đầu  $cur\_x$  xa hơn từ mức tối thiểu cục bộ (Bạn có thể ước tính điều này bằng cách xem ô con của  $f'(x)$ ), số lần lặp tăng lên vì thuật toán GD mất nhiều thời gian hơn để hội tụ về mức tối thiểu cục bộ. Như mong đợi, mức tối thiểu cục bộ xấp xỉ bằng 0.

Ví dụ thứ 2 tìm cực đại cục bộ của hàm sigmoid:

nhập matplotlib.pyplot dưới dạng plt, numpy dưới dạng np

```
def sigmoid(x):
    trả về 1 / (1 + np.exp(-x))

def df(x):
    trả về x    * (1-x)

nếu __name__ == "__main__":
    np.arange(-10., 10., 0.2) y, y_dx =
        sigmoid(x), df(x) f, axarr =
        plt.subplots(3, sharex=True) axarr[0].plot(x, y,
            color='lime') axarr[0].set_xlabel('x')
        axarr[0].set_ylabel('f(x)')
        axarr[0].set_title('Hàm Sigmoid')
        axarr[1].plot(x, y_dx, color='coral')
        axarr[1].set_xlabel('x') axarr[1].set_ylabel('dy/
            dx(x)')
        axarr[1].set_title('Derivative of
            f(x)') axarr[2].set_xlabel('x')
        axarr[2].set_ylabel('GD')
        axarr[2].set_title('local
            maximum') lặp lại, cur_x, gamma, độ chính
        xác = 0, 0,01, 0,01, 0,00001 trước_step_size = cur_x
```

## Chương 4 Dốc độ dốc

trong khi `previous_step_size > độ chính xác:`

```

trước_x = cur_x
cur_x += -gamma      * -df(prev_x)
previous_step_size = abs(cur_x - prev_x) lần
lặp += 1

plt.plot(prev_x, cur_x, "o")
f.subplots_adjust(hspace=0.3)
f.tight_layout()
print ('maximum:', cur_x, '\nitations:', iterations) plt.show()

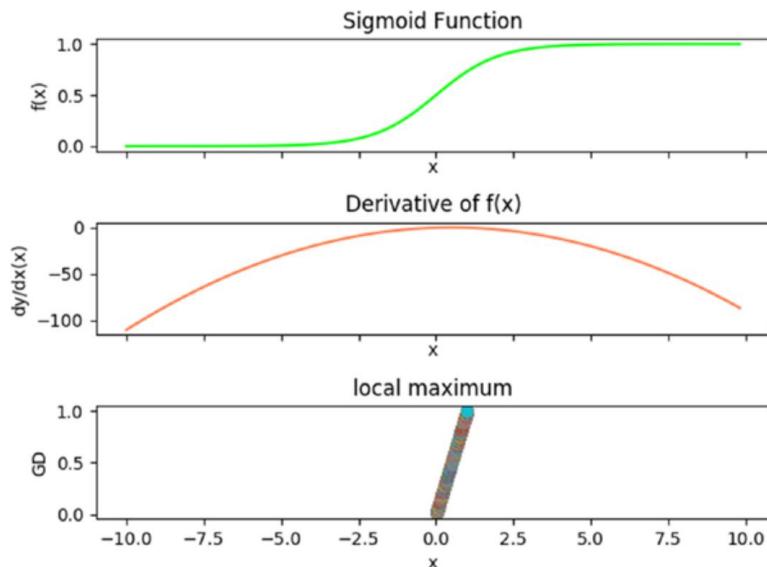
```

Đầu ra:

```

maximum: 0.9990096316387825
iterations: 1150

```



Hình 4-6. Ô con của  $f(x)$ ,  $f'(x)$  và cực đại cục bộ

Mã bắt đầu bằng cách nhập `matplotlib` và `numpy`. Nó tiếp tục với các hàm `sigmoid(x)` và `df(x)`, đại diện cho hàm sigmoid và đạo hàm của nó theo thuật toán. Khối chính bắt đầu bằng cách tạo dữ liệu cho  $f(x)$  và  $f'(x)$ . Nó tiếp tục bằng cách tạo các ô con cho  $f(x)$ ,  $f'(x)$  và cực đại cục bộ (Hình 4-6).  
 Đễ dàng thấy từ đồ thị  $f(x)$  rằng cực đại cục bộ gần bằng 1. Tiếp theo, đoạn mã chạy GD để tìm cực đại cục bộ và vẽ đồ thị cho nó. Một lần nữa, điểm khởi đầu cho GD, `cur_x`, được tìm thấy bằng phương pháp thử và sai. Nếu bạn bắt đầu `cur_x` xa hơn từ mức tối đa cục bộ (Bạn có thể ước tính điều này bằng cách xem ô con của  $f(x)$ ), số lần lặp sẽ tăng lên vì thuật toán GD mất nhiều thời gian hơn để hội tụ về mức tối đa cục bộ. Theo dự kiến, mức tối đa cục bộ là khoảng 1.

## Tối thiểu hóa khoảng cách Euclidean

### Kiểm soát kích thước bước

Khoảng cách Euclidean là khoảng cách đường thẳng thông thường giữa hai điểm trong không gian Euclidean. Với khoảng cách này, không gian Euclidean trở thành không gian metric. Định mức liên quan là chuẩn Euclidean (EN). EN gắn cho mỗi vectơ độ dài mũi tên của nó. Vì vậy, EN thực sự chỉ là độ lớn của một vectơ. Một không gian vectơ mà trên đó một chuẩn được xác định là không gian vectơ chuẩn.

Để tìm cực tiểu cục bộ của  $f(x)$  trong không gian ba chiều (3-D), bước đầu tiên là tìm giá trị nhỏ nhất cho tất cả các vectơ 3 chiều. Bước thứ 2 là tạo một vectơ 3 chiều ngẫu nhiên  $[x, y, z]$ . Bước thứ 3 là chọn một điểm bắt đầu ngẫu nhiên, sau đó thực hiện các bước nhỏ theo hướng ngược lại của độ dốc  $f'(x)$  cho đến khi đạt đến một điểm tại đó độ dốc rất nhỏ. Mỗi bước nhỏ (từ vectơ hiện tại sang vectơ tiếp theo) được đo bằng chỉ số ED. Số liệu ED là khoảng cách giữa hai điểm trong không gian Euclidean. Số liệu này là bắt buộc vì chúng ta cần biết cách di chuyển cho từng bước nhỏ. Vì vậy, số liệu ED bổ sung cho GD để tìm mức tối thiểu cục bộ trong không gian 3 chiều.

## Chương 4 Đốc độ đốc

Ví dụ mã tìm giá trị cực tiểu cục bộ của hàm sigmoid trong

Không gian 3 chiều:

```

nhập matplotlib.pyplot dưới dạng
plt từ mpl_toolkits.mplot3d nhập Axes3D nhập
ngẫu nhiên, gọn gàng dưới dạng
np từ khoảng cách nhập scipy.spatial

def step(v, direction, step_size):
    return [v_i + step_size * direction_i
            cho v_i, direction_i trong zip(v, direction)]

def sigmoid_gradient(v):
    trả về [v_i * (1-v_i) cho v_i trong v]

def mod_vector(v):
    for i, v_i in enumerate(v): if
        v_i == float("inf") or v_i == float("-inf"): v[i] =
            random.randint(-1, 1) trả lại v

nếu __name__ == "__main__":
    v = [random.randint(-10, 10) for i in range(3)] dung
    sai = 0,0000001
    lần lặp = 1
    fig = plt.figure('Euclidean') ax
    = fig.add_subplot(111, projector='3d') trong khi
    True:
        gradient = sigmoid_gradient(v)
        next_v = step(v, gradient, -0,01) xs
        = gradient[0] ys
        = gradient[1] zs
        = gradient[2]
        ax.scatter(xs, ys, zs, c='lime' , điểm đánh dấu='o')

```

```

v = mod_vector(v)
next_v = mod_vector(next_v)
test_v = distance.euclidean(v, next_v) if
test_v < dung sai: break

v = next_v
lần lặp += 1

print ('tối thiểu:', test_v, '\nitations:', iterations)
ax.set_xlabel('Trục X')
ax.set_ylabel('Trục Y')
ax.set_zlabel('Trục Z')
plt. chăt_layout()
plt .trình diễn()

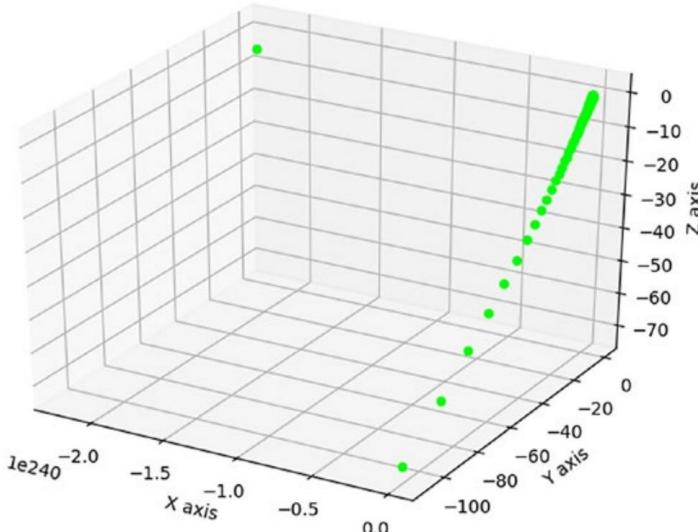
```

Đầu ra:

```

minimum: 9.980323358143592e-08
iterations: 1184

```



Hình 4-7. Hiển thị 3-D của mức tối thiểu cục bộ

## Chương 4 Đốc độ dốc

Mã bắt đầu bằng cách nhập các thư viện `matplotlib`, `mpl_toolkits`, `random`, `numpy` và `scipy`. Hàm `step()` di chuyển một vectơ theo hướng (dựa trên độ dốc), theo kích thước bước. Hàm `sigmoid_gradient()` là  $f'(sigmoid)$  được trả về dưới dạng một điểm trong không gian 3 chiều. Hàm `mod_vector()` đảm bảo rằng một vectơ sai được tạo bởi mô phỏng được xử lý đúng cách. Khối chính bắt đầu bằng cách tạo một vectơ 3 chiều được tạo ngẫu nhiên  $[x, y, z]$  làm điểm bắt đầu cho mô phỏng. Nó tiếp tục bằng cách tạo ra một dung sai (độ chính xác). Dung sai nhỏ hơn dẫn đến kết quả chính xác hơn. Một ô con được tạo để giữ kết xuất 3-D ở mức tối thiểu cục bộ (Hình 4-7). Mô phỏng GD tạo ra một tập hợp các vectơ 3-D chịu ảnh hưởng của gradient sigmoid cho đến khi gradient rất nhỏ. Kích thước (độ lớn) của gradient được tính bằng chỉ số ED. Mức tối thiểu cục bộ, như mong đợi là gần bằng 0.

## Ôn định giảm thiểu khoảng cách Euclidean với mô phỏng Monte Carlo

Thí nghiệm khoảng cách Euclidean trong ví dụ trước được cố định bởi một quá trình ngẫu nhiên. Cụ thể, vectơ bắt đầu  $v$  được tạo ngẫu nhiên bởi `Randomint()`. Do đó, mỗi lần chạy thử nghiệm GD tạo ra một kết quả khác nhau cho số lần lặp lại. Từ Chương 2, chúng ta đã biết rằng mô phỏng Monte Carlo (MCS) mô hình hiệu quả các quá trình ngẫu nhiên (ngẫu nhiên). Tuy nhiên, MCS cũng có thể ổn định các thí nghiệm ngẫu nhiên.

Ví dụ về mã đầu tiên kết thúc thử nghiệm GD trong một vòng lặp chạy  $n$  số lần mô phỏng. Với  $n$  mô phỏng, số lần lặp trung bình được tính toán. Mã kết quả sau đó được bao bọc trong một vòng lặp khác chạy  $m$  thử nghiệm. Với  $m$  thử nghiệm, khoảng cách trung bình giữa mỗi số lần lặp lại trung bình sẽ được tính toán. Khoảng cách được tính bằng cách trừ đi mức tối thiểu từ lần lặp trung bình tối đa. Khoảng cách càng nhỏ, kết quả càng ổn định (chính xác). Để tăng độ chính xác, hãy tăng

mô phỏng ( $n$ ). Hạn chế duy nhất là sức mạnh tính toán. Nghĩa là, chạy 1.000 mô phỏng cần nhiều sức mạnh tính toán hơn 100. Kết quả ổn định (chính xác) cho phép so sánh với các thử nghiệm thay thế.

nhập ngẫu nhiên, gọn gàng dưới dạng np  
từ khoảng cách nhập `scipy.spatial`

bước xác định ( $v$ , hướng, bước\_size):

```
trả về [v_i + step_size * direction_i
          cho v_i, direction_i trong zip(v, direction)]
```

`def sigmoid_gradient(v):`

```
trả lại [v_i * (1-v_i) cho v_i trong v]
```

`def mod_vector(v):`

```
cho i, v_i trong liệt kê (v):
```

```
nếu v_i == float("inf") hoặc v_i == float("-inf"):
```

```
v[i] = ngẫu nhiên.randint(-1, 1)
```

```
trả lại v
```

`nếu __name__ == "__main__":`

```
thử nghiệm = 10
```

```
sims = 10
```

```
avg_its = []
```

```
cho trong phạm vi (thử nghiệm):
```

```
nó = []
```

```
cho trong phạm vi (sims):
```

```
v = [random.randint(-10, 10) cho tôi trong phạm vi (3)]
```

```
dung sai = 0,0000001
```

```
lặp lại = 0
```

```
trong khi Đúng:
```

```
độ dốc = sigmoid_gradient(v)
```

```
next_v = bước (v, độ dốc, -0,01)
```

```
v = mod_vector(v)
```

## Chương 4 Dốc độ dốc

```

next_v = mod_vector(next_v)
test_v = distance.euclidean(v, next_v)
nếu test_v < dung sai:
    phá vỡ

v = next_v
lặp lại += 1

it.append(lặp đi lặp lại)
a = round(np.mean(its))
avg_its.append(a)

khoảng cách = np.max(avg_its) - np.min(avg_its)
print (thử nghiệm, 'thử nghiệm với', sims, 'từng mô phỏng:')
in ('khoảng cách', khoảng cách)
in ('số lần lặp trung bình', round(np.mean(avg_its)))

```

Đầu ra:

```

10 trials with 10 simulations each:
gap 243.0
avg iterations 1031.0

10 trials with 100 simulations each:
gap 97.0
avg iterations 1087.0

10 trials with 1000 simulations each:
gap 13.0
avg iterations 1089.0

```

Đầu ra dành cho 10, 100 và 1.000 mô phỏng. Bằng cách chạy 1.000 mô phỏng mười lần (thử nghiệm), khoảng cách giảm xuống còn 13. Vì vậy, độ tin cậy cao là số lần lặp lại cần thiết để giảm thiểu chừn năng là gần 1.089. Chúng ta có thể ổn định hơn nữa bằng cách gói mã trong một vòng lặp khác để giảm sự thay đổi về khoảng cách và số lần lặp lại. Tuy nhiên, thời gian xử lý máy tính trở thành một vấn đề. Tận dụng MCS cho loại thử nghiệm này tạo nên một trường hợp mạnh mẽ cho điện toán đám mây. Có thể khó hiểu về ứng dụng MCS này, nhưng nó là một công cụ rất mạnh để làm việc và giải quyết các vấn đề về khoa học dữ liệu.

## Thay thế một phương pháp NumPy để tăng tốc Tối thiểu hóa khoảng cách Euclidean

Vì các mảng có nhiều mảng nhanh hơn các danh sách Python, nên việc sử dụng một phương thức có nhiều mảng sẽ hiệu quả hơn để tính khoảng cách Euclidean. Ví dụ mã thay thế `np.linalg.norm()` cho `distance.euclidean()` để tính khoảng cách Euclidean cho thử nghiệm GD.

```
nhập matplotlib.pyplot dưới dạng plt
từ mpl_toolkits.mplot3d nhập Axes3D nhập ngẫu
nhiên, numpy dưới dạng np

def step(v, direction, step_size): return
    [v_i + step_size * direction_i
        cho v_i, direction_i trong zip(v, direction)]

def sigmoid_gradient(v): trả
    về [v_i * (1-v_i) cho v_i trong v]

def round_v(v):
    trả về np.round(v, số thập phân=3)

nếu __name__ == "__main__":
    v = [random.randint(-10, 10) for i in range(3)] dung sai
    = 0,0000001
    lần lặp = 1
    fig = plt.figure('norm') ax
    = fig.add_subplot(111, projector='3d') trong khi
    True:
        gradient = sigmoid_gradient(v)
        next_v = step(v, gradient, -0.01)
        round_gradient = round_v(gradient) xs =
        round_gradient[0] ys =
        round_gradient[1]
```

## Chương 4 Dốc độ dốc

```

zs = round_gradient[2]
ax.scatter(xs, ys, zs, c='lime', marker='o')
Norm_v = np.linalg.norm(v)
Norm_next_v = np.linalg.norm(next_v)
test_v = Norm_v - Norm_next_v
nếu test_v < dung sai:
    phá vỡ
v = next_v
lần lặp += 1
print ('tối thiểu:', test_v, '\niterations:', iterations)
ax.set_xlabel('trục X')
ax.set_ylabel('trục Y')
ax.set_zlabel('trục Z')
plt.show()

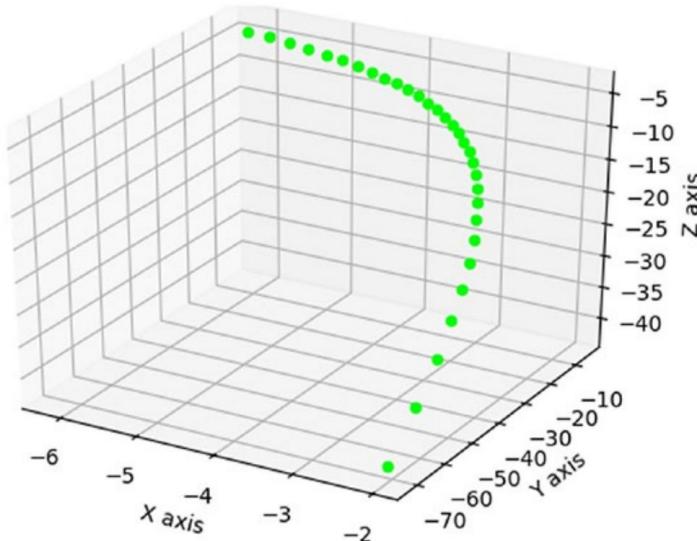
```

Đầu ra:

```

minimum: -0.0046610878817
iterations: 31

```



Hình 4-8. Hiển thị 3-D Numpy ở mức tối thiểu cục bộ

## Chương 4 Dốc độ dốc

Số lần lặp thấp hơn nhiều ở mức 31 (Hình 4-8). Tuy nhiên, do thử nghiệm GD là ngẫu nhiên, chúng ta có thể sử dụng MCS để so sánh khách quan.

Sử dụng cùng một phương pháp MCS, ví dụ mã đầu tiên kết thúc thử nghiệm GD trong một vòng lặp chạy n số lần mô phỏng. Mã kết quả sau đó được bao bọc trong một vòng lặp khác chạy m thử nghiệm.

nhập ngẫu nhiên, gọn gàng dưới dạng np

bước xác định ( $v$ , hướng, bước\_size):

```
trả về [v_i + step_size * direction_i
          cho v_i, direction_i trong zip(v, direction)]
```

def sigmoid\_gradient(v):

```
trả lại [v_i * (1-v_i) cho v_i trong v]
```

xác định round\_v( $v$ ):

```
trả về np.round(v, số thập phân=3)
```

nếu \_\_name\_\_ == "\_\_main\_\_":

```
thử nghiệm = 10
```

```
sim = 10
```

```
avg_its = []
```

```
cho trong phạm vi (thử nghiệm):
```

```
    nó = []
```

```
    cho trong phạm vi (sims):
```

```
        v = [random.randint(-10, 10) cho tôi trong phạm vi (3)]
```

```
        dung sai = 0,0000001
```

```
        lặp lại = 0
```

```
        trong khi Đúng:
```

```
            độ dốc = sigmoid_gradient(v)
```

```
            next_v = bước (v, độ dốc, -0,01)
```

```
            Norm_v = np.linalg.norm(v)
```

```
            Norm_next_v = np.linalg.norm(next_v)
```

## Chương 4 Đốc độ đốc

```

test_v = Norm_v - Norm_next_v
nếu test_v < dung sai:
    phá vỡ
v = next_v
lặp lại += 1
it.append(lặp đi lặp lại)
a = round(np.mean(its))
avg_its.append(a)
khoảng cách = np.max(avg_its) - np.min(avg_its)
print ('thử nghiệm', 'thử nghiệm với', sims, 'từng mô phỏng:')
in ('khoảng cách', khoảng cách)
in ('số lần lặp trung bình', round(np.mean(avg_its)))

```

Đầu ra:

```

10 trials with 10 simulations each:
gap 235.0
avg iterations 164.0

10 trials with 100 simulations each:
gap 141.0
avg iterations 200.0

10 trials with 1000 simulations each:
gap 27.0
avg iterations 193.0

```

Xử lý nhanh hơn nhiều khi sử dụng numpy. Số lượng trung bình của số lần lặp lại gần bằng 193. Như vậy, sử dụng phương án thay thế gọn gàng để tính khoảng cách Euclidean nhanh hơn gấp năm lần!

## Tối thiểu hóa và tối đa hóa giảm dần độ đốc ngẫu nhiên

Cho đến thời điểm này trong chương, các thử nghiệm tối ưu hóa đã sử dụng GD hàng loạt. Batch GD tính toán độ đốc bằng cách sử dụng toàn bộ tập dữ liệu. Stochastic GD tính toán độ đốc bằng cách sử dụng một mẫu duy nhất, do đó, nó được tính toán

## Chương 4 Dốc độ dốc

nhanh hơn nhiều. Nó được gọi là GD ngẫu nhiên vì độ dốc được xác định ngẫu nhiên. Tuy nhiên, không giống như GD hàng loạt, GD ngẫu nhiên là một giá trị gần đúng. Nếu yêu cầu độ dốc chính xác, GD ngẫu nhiên không phải là tối ưu. Một vấn đề khác với GD ngẫu nhiên là nó có thể dao động quanh mức tối thiểu mãi mãi mà không thực sự hội tụ. Vì vậy, điều quan trọng là vẽ sơ đồ tiến trình mô phỏng để xem điều gì đang xảy ra.

Hãy đổi hướng và tối ưu hóa một chức năng quan trọng khác—tổng bình phương còn lại (RSS). Hàm RSS là một kỹ thuật thống kê để đo lường lỗi (phương sai) còn lại giữa hàm hồi quy và tập dữ liệu. Phân tích hồi quy là một thuật toán ước tính mối quan hệ giữa các biến. Nó được sử dụng rộng rãi để dự đoán và dự báo. Nó cũng là một thuật toán dự đoán và lập mô hình phổ biến cho các ứng dụng khoa học dữ liệu.

Ví dụ mã đầu tiên tạo một mảng, chạy thử nghiệm GD n lần và xử lý mảng một cách ngẫu nhiên:

```
nhập matplotlib.pyplot dưới dạng plt
nhập ngẫu nhiên, gọn gàng dưới dạng np

chắc chắn rnd():
    trả về [random.randint(-10,10) cho tôi trong phạm vi (3)]

def Random_vectors(n):
    ls = []
    cho v trong phạm vi (n):
        ls.append(rnd())
    trả lại ls

chắc chắn sos(v):
    trả lại tổng (v_i ** 2 cho v_i trong v)

def sos_gradient(v):
    trả lại [2 * v_i cho v_i trong v]
```

## Chương 4 Đốc độ đốc

```

def in_random_order(data):
    indexes = [i for i, _ in enumerate(data)]
    random.shuffle(indexes)
    for i in indexes:
        yield data[i]

nếu __name__ == "__main__":
    v, x, y = rnd(), random_vectors(3), random_vectors(3)
    data = list(zip(x, y))
    theta = v

    alpha, value = 0, 0
    min_theta, min_value = Không, float("inf")
    iterations_with_no_improvement = 0
    n, x = 30, 1
    _ trong liệt kê(phạm
vi(n)): for i, y =
    np.linalg.norm(theta) plt.scatter(x, y, c='r')
    x = x + 1

    s =
    [] cho x_i, y_i trong
    dữ liệu: s.extend([sos(theta), sos(x_i),
    sos(y_i)])
    value = sum(s) if
        value < min_value: min_theta,
        min_value = theta, value
    iterations_with_no_improvement = 0 alpha = 0,01
    khác:
        iterations_with_no_improvement += 1
        alpha *= 0,9
    g = []

```

```

    cho x_i, y_i trong in_random_order(dữ
    liệu): g.extend([sos_gradient(theta),
                    sos_gradient(x_i),
                    sos_gradient(y_i)]) cho v trong g: theta =
                    np.around(np.subtract(theta, alpha*
                    np.array(v)), 3) g = []
print ('minimum:', np.around(min_theta, 4),
      'with', i+1, 'iterations') print
('các lần lặp không cải thiện: ', iterations_with_no_improvement)
print ('độ lớn của vectơ tối thiểu:', np.linalg.norm(min_theta)) plt.sh

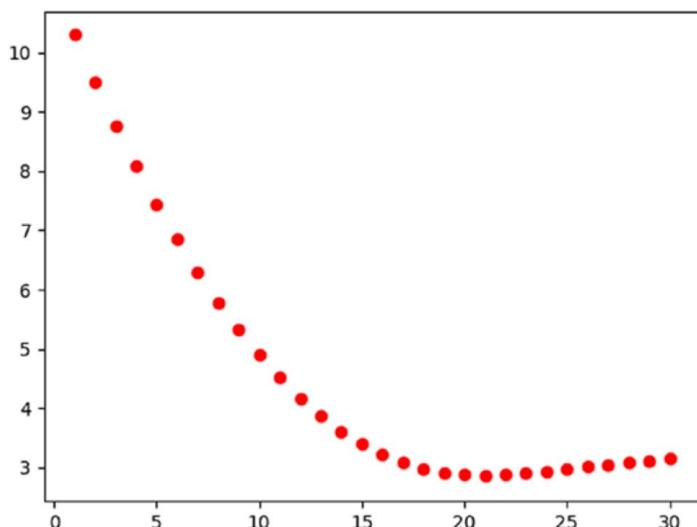
```

Đầu ra:

```

minimum: [ 0.609 -2.07   1.892] with 30 iterations
iterations with no improvement: 9
magnitude of min vector: 2.86974650448

```



Hình 4-9. giảm thiểu RSS

## Chương 4 Đốc độ dốc

Mã bắt đầu bằng cách nhập `matplotlib`, `ngau` nhiên và `numpy`. Nó tiếp tục với hàm `rnd()`, trả về danh sách các số nguyên ngẫu nhiên từ -10 đến 10. Hàm `random_vectors()` tạo danh sách (mẫu ngẫu nhiên) gồm n số. Hàm `sos()` trả về RSS cho một vectơ. Hàm `sos_`

`gradient()` trả về đạo hàm (độ dốc) của RSS cho một vectơ. Hàm `in_random_order()` tạo danh sách các chỉ mục được xáo trộn ngẫu nhiên. Hàm này thêm hương vị ngẫu nhiên vào thuật toán GD. Khối chính bắt đầu bằng cách tạo một vectơ ngẫu nhiên và làm điểm bắt đầu cho mô phỏng. Nó tiếp tục bằng cách tạo một mẫu vectơ x và y có kích thước 3.

Tiếp theo, vectơ được gán cho theta, đây là tên chung cho một vectơ của một số phân phối xác suất chung. Chúng ta có thể gọi vectơ là bất cứ thứ gì chúng ta muốn, nhưng một vấn đề phổ biến về khoa học dữ liệu là tìm (các) giá trị của theta. Mã tiếp tục với kích thước bước cố định alpha, giá trị theta tối thiểu, giá trị kết thúc tối thiểu, các lần lặp lại không cải thiện, số lần mô phỏng n và giá trị biểu đồ cho tọa độ x (Hình 4-9).

Mô phỏng bắt đầu bằng cách gán cho y độ lớn của theta. Tiếp theo, nó vẽ các tọa độ x và y hiện tại. Tọa độ x được tăng thêm 1 để biểu thị sự hội tụ ở mức tối thiểu cho mỗi tọa độ y. Khối mã tiếp theo tìm RSS cho mỗi theta và mẫu của các giá trị x và y. Giá trị này xác định xem mô phỏng có lờ lững quanh mức tối thiểu cục bộ thay vì hội tụ hay không. Phần cuối cùng của mã duyệt qua các điểm dữ liệu mẫu theo thứ tự ngẫu nhiên (ngẫu nhiên), tìm độ dốc của theta, x và y, đặt ba giá trị này vào danh sách g và duyệt qua vectơ này để tìm giá trị theta tiếp theo.

Chà! Điều này không đơn giản, nhưng đây là cách hoạt động của GD ngẫu nhiên. Lưu ý rằng mức tối thiểu được tạo là 2,87, không phải là mức tối thiểu thực sự là 0. Vì vậy, GD ngẫu nhiên yêu cầu một vài lần lặp lại nhưng không tạo ra mức tối thiểu thực sự.

Mô phỏng trước đó có thể được tinh chỉnh bằng cách điều chỉnh thuật toán để tìm theta tiếp theo. Trong ví dụ trước, theta tiếp theo được tính cho dài màu dựa trên theta hiện tại, giá trị x và giá trị y cho mỗi mẫu. Tuy nhiên, theta mới thực tế dựa trên điểm dữ liệu thứ 3 trong

vật mẫu. Vì vậy, ví dụ thứ 2 được tinh chỉnh bằng cách lấy theta tối thiểu từ toàn bộ mẫu thay vì điểm dữ liệu thứ 3:

```

nhập matplotlib.pyplot dưới dạng plt
nhập ngẫu nhiên, numpy dưới dạng np

def rnd():
    trả về [random.randint(-10,10) cho tôi trong phạm vi (3)]

def Random_vectors(n): ls
    = [] for
    v in range(n):
        ls.append([random.randint(-10,10) for i in range(3)]) return ls

def sos(v):
    trả về tổng(v_i ** 2 cho v_i trong v)

def sos_gradient(v): trả
    về [2 * v_i cho v_i trong v]

def in_random_order(data):
    indexes = [i for i, in enumerate(data)]
    random.shuffle(indexes) for
    i in indexes: yield
        data[i]

nếu __name__ == "__main__":
    v, x, y = rnd(), random_vectors(3), random_vectors(3) data =
    list(zip(x, y)) theta = v
    alpha,
    value = 0.01, 0 min_theta,
    min_value = Không, float("inf ")
    iterations_with_no_improvement = 0 n, x =
    60, 1

```

## Chương 4 Dốc độ dốc

```

cho i, _ trong liệt kê (phạm vi (n)):
    y = np.linalg.norm(theta)
    plt.scatter(x, y, c='r')
    x = x + 1

    s = []
    cho x_i, y_i trong dữ
        liệu: s.extend([sos(theta), sos(x_i), sos(y_i)])
    value = sum(s)
    if value < min_value:
        min_theta, min_value = theta, value
        iterations_with_no_improvement = 0 alpha
        = 0,01
    khác:
        iterations_with_no_improvement += 1 alpha
        *= 0,9 g, t,
    m = [], [], [] for x_i,
    y_i in in_random_order(data):
        g.extend([sos_gradient(theta), sos_gradient(x_i),
                  sos_gradient(y_i)]) m
        = np.around([np.linalg.norm(x) for x in g], 2) for v in g:

            theta = np.around(np.subtract(theta,alpha*np.
array(v)),3)
            t.append(np.around(theta,2)) mm =
            np.argmin(m) theta
            = t[mm] g, m,
            t = [], [], []
print ('tối thiểu:', np.around(min_theta, 4), 'với',
      i+1, 'lần lặp')

```

```

print('các lần lặp không cải tiến:',
      iterations_with_no_improvement)
print ('độ lớn của vectơ tối thiểu:', np.linalg.norm(min_theta))
plt.show()

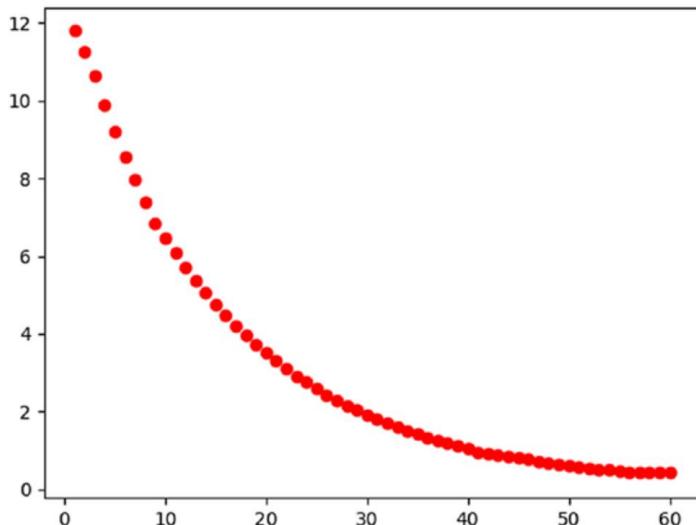
```

Đầu ra:

```

minimum: [ 0.26  0.26  0.26] with 60 iterations
iterations with no improvement: 3
magnitude of min vector: 0.450333209968

```



Hình 4-10. Giảm thiểu RSS đã sửa đổi

Sự khác biệt duy nhất trong mã là về phía dưới nơi mà theta tối thiểu được tính toán (Hình 4-10). Mặc dù phải mất 60 lần lặp lại, mức tối thiểu gần bằng 0 và ổn định hơn nhiều. Nghĩa là, ví dụ trước sai lệch nhiều hơn một chút mỗi khi chạy thử nghiệm.

## Chương 4 Dốc độ dốc

Ví dụ thứ 3 tìm thấy mức tối đa:

```

nhập matplotlib.pyplot dưới dạng plt
nhập ngẫu nhiên, numpy dưới dạng np

def rnd():
    trả về [random.randint(-10,10) cho tôi trong phạm vi (3)]

def Random_vectors(n): ls
    = [] for
    v in range(n):
        ls.append([random.randint(-10,10) for i in range(3)]) return ls

def sos_gradient(v): trả
    về [2 * v_i cho v_i trong v]

def negate(function): def
    new_function(*args, **kwargs): return
        np.negative(function(*args, **kwargs)) return new_function

def in_random_order(data):
    indexes = [i for i, in enumerate(data)]
    random.shuffle(indexes) for
    i in indexes: yield
        data[i]

nếu __name__ == "__main__":
    v, x, y = rnd(), random_vectors(3), random_vectors(3) data =
    list(zip(x, y)) theta,
    alpha = v, 0,01 neg_gradient
    = negate(sos_gradient) n, x = 100, 1

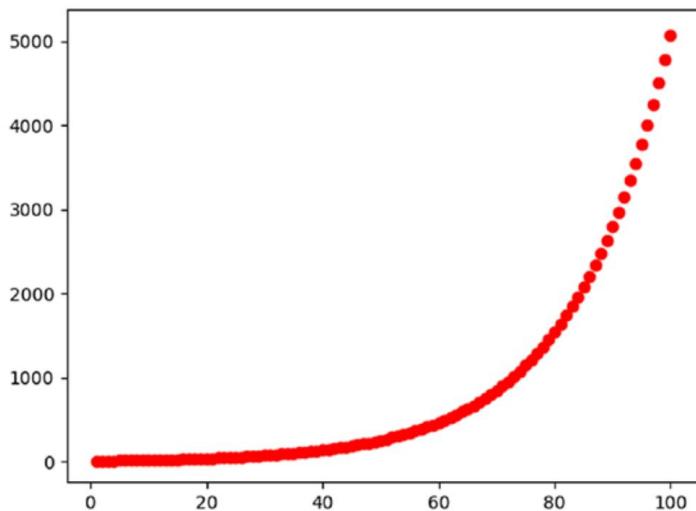
```

đối với i, hàng trong liệt kê(phạm  
vi(n)): y =  
np.linalg.norm(theta) plt.scatter(x, y, c='r')  
x = x + 1  
g = []  
cho x\_i, y\_i trong in\_random\_order(data):  
g.extend([neg\_gradient(theta), neg\_gradient(x\_i),  
neg\_gradient(y\_i)])  
cho v trong  
g: theta = np.around(np.subtract(theta, alpha\*np.  
array(v)),3)  
g = []  
print ('maximum:', np.around(theta, 4),  
'with', i+1, 'iterations')  
print ('độ lớn của vectơ tối đa:', np.linalg.norm(theta))  
plt.show()

Đầu ra:

```
maximum: [-1521.6    4178.212 -3038.379] with 100 iterations
magnitude of max vector: 5385.57972967
```

## Chương 4 Dốc độ dốc



Hình 4-11. tối đa hóa RSS

Sự khác biệt duy nhất trong mã từ ví dụ đầu tiên là phủ định() chức năng phủ nhận độ dốc để tìm giá trị lớn nhất. Vì mức tối đa của RSS là vô hạn (chúng ta có thể biết bằng hình ảnh trực quan trong Hình 4-11), chúng ta có thể dừng ở 100 lần lặp. Hãy thử 1.000 lần lặp lại và xem điều gì sẽ xảy ra.

## CHƯƠNG 5

# Làm việc với dữ liệu

Làm việc với dữ liệu chi tiết các quy trình sớm nhất để giải quyết vấn đề khoa học dữ liệu. Bước đầu tiên là xác định vấn đề, từ đó xác định tất cả những việc cần làm khác. Bước thứ 2 là thu thập dữ liệu. Bước thứ 3 là sắp xếp dữ liệu (*munge*), điều này rất quan trọng. Sắp xếp lại đang đưa dữ liệu vào một dạng hữu ích cho máy học và các vấn đề khoa học dữ liệu khác. Tất nhiên, dữ liệu lộn xộn có thể sẽ phải được làm sạch. Bước thứ 4 là trực quan hóa dữ liệu. Trực quan hóa giúp bạn tìm hiểu dữ liệu và hy vọng xác định được các mẫu.

## Ví dụ dữ liệu một chiều

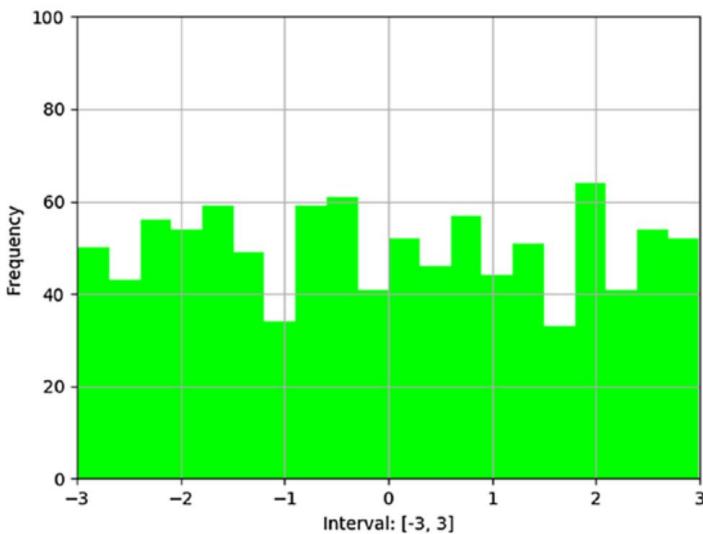
Ví dụ mã tạo trực quan hóa hai phân phối dữ liệu rất phổ biến–đồng nhất và bình thường. Phân bố đều có hằng số xác suất. Nghĩa là tất cả các biến có thuộc phân phối đều có xác suất như nhau. Phân phối bình thường đối xứng về trung tâm, có nghĩa là 50% giá trị của nó nhỏ hơn giá trị trung bình và 50% giá trị của nó lớn hơn giá trị trung bình. Hình dạng của nó giống như một đường cong chuông. Phân phối chuẩn cực kỳ quan trọng vì nó mô hình hóa nhiều sự kiện xảy ra tự nhiên.

## Chương 5 Làm việc với dữ liệu

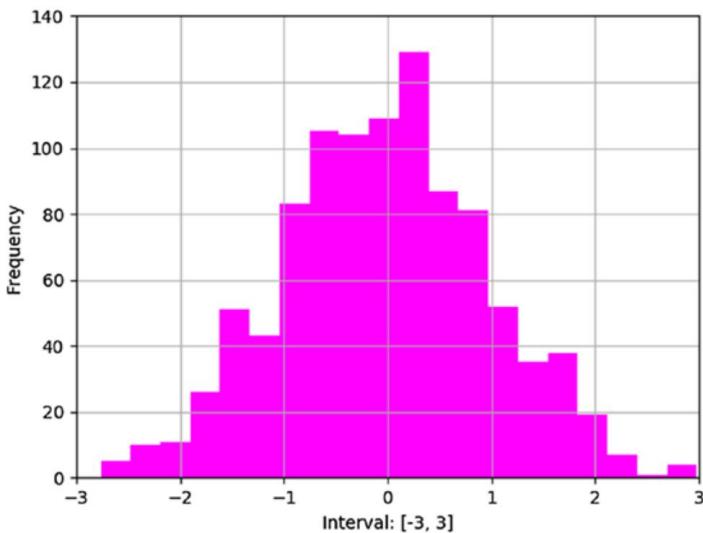
```
nhập matplotlib.pyplot dưới dạng plt
nhập numpy dưới dạng np

if __name__ == "__main__":
    plt.figure('Uniform Distribution') thống
    nhất = np.random.uniform(-3, 3, 1000) đếm, thùng,
    bỏ qua = plt.hist(uniform, 20, facecolor='lime') plt.xlabel('Khoảng thời
    gian: [-3, 3]') plt.ylabel('Tần suất')
    plt.title('Phân bố đồng
    đều') plt.axis([-3, 3, 0, 100]) plt.grid
    (Đúng) plt.figure('Phân
    phối bình
    thường') normal = np.random.normal(0,
    1, 1000) đếm, thùng, bị bỏ qua = plt.hist(bình
    thường, 20, facecolor='fuchsia') plt.xlabel( 'Khoảng
    thời gian: [-3, 3]')
    plt.ylabel('Tần suất') plt.title('Phân
    phối chuẩn')
    plt.axis([-3, 3, 0, 140]) plt.grid(True)
    plt.show ()
```

Đầu ra:



Hình 5-1. Phân bố đồng đều



Hình 5-2. Phân phối bình thường

## Chương 5 Làm việc với dữ liệu

Ví dụ mã bắt đầu bằng cách nhập `matplotlib` và `numpy`. Khởi chính bắt đầu bằng cách tạo một hình và dữ liệu để phân phối đồng đều. Tiếp theo, một biểu đồ được tạo và vẽ dựa trên dữ liệu. Một con số cho một phân phối bình thường sau đó được tạo ra và vẽ đồ thị. Xem Hình 5-1 và [5-2](#).

### Ví dụ về dữ liệu hai chiều

Mô hình hóa dữ liệu 2-D cung cấp một bức tranh chân thực hơn về các sự kiện xảy ra tự nhiên. Ví dụ mã so sánh hai bản phân phối thông thường của dữ liệu được tạo ngẫu nhiên có cùng giá trị trung bình và độ lệch chuẩn (SD). SD đo lượng biến thể (độ phân tán) của một tập hợp các giá trị dữ liệu. Mặc dù cả hai bộ dữ liệu thường được phân phối với cùng một giá trị trung bình và SD, nhưng mỗi bộ có một phân phối chung rất khác nhau (tương quan). Tương quan là sự phụ thuộc lẫn nhau của hai biến.

```

nhập matplotlib.pyplot dưới dạng plt
nhập matplotlib.gridspec dưới dạng gridspec
nhập numpy dưới dạng np, ngẫu nhiên
từ scipy.nhập đặc biệt ndtri

def inverse_normal_cdf(r):
    trả về ndtri(r)

def Random_normal():
    trả về inverse_normal_cdf(random.random())

def phân tán (loc):
    plt.scatter(xs, ys1, marker='.', color='black', label='ys1')
    plt.scatter(xs, ys2, marker='.', color='gray', label='ys2')
    plt.xlabel('xs')
    plt.ylabel('ys')
    plt.legend(loc=loc)
    plt.chặt_layout()

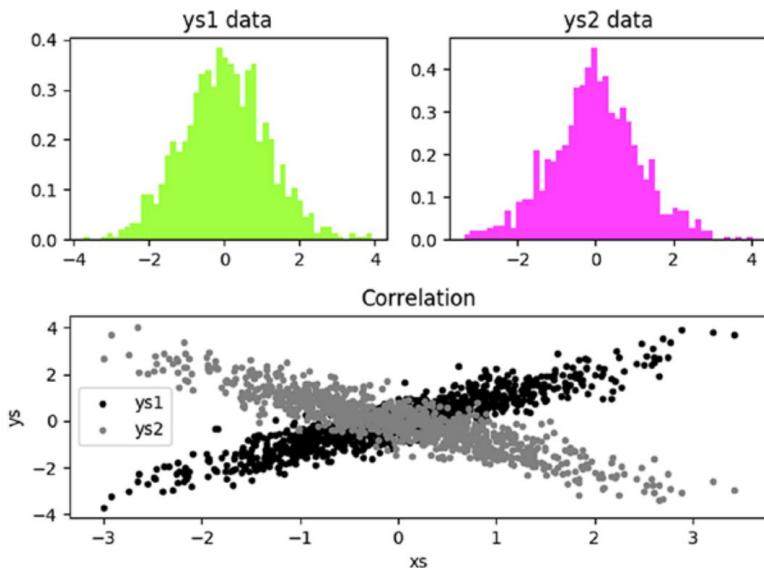
```

```
nếu __name__ == "__main__":
    xs = [random_normal() cho _ trong phạm vi (1000)]
    ys1 = [x + random_normal() / 2 cho x trong xs]
    ys2 = [-x + random_normal() / 2 cho x trong xs]
    gs = gridspec.GridSpec(2, 2)
    fig = plt.figure()
    ax1 = fig.add_subplot(gs[0,0])
    plt.title('dữ liệu
ys1') n, các thùng, bị bỏ qua = plt.hist(ys1, 50,
normed=1,
facecolor='chartreuse' ,
alpha=0,75) ax2 =
fig.add_subplot(gs[0,1]) plt.title('ys2 data') n,
bins, gone =
plt.hist(ys2,
50, normed=1, facecolor='fuchsia' ,
alpha=0,75) ax3 =
fig.add_subplot(gs[1,:])
plt.title('Tương quan') scatter(6)
print (np.corrcoef(xs, ys1)[0, 1]) print (np. corrcoef(xs, ys2)[0, 1]) p
```

## Chương 5 Làm việc với dữ liệu

Đầu ra:

```
0.907683439554
-0.896109957488
```



Hình 5-3. Subplot của phân phối bình thường và tương quan

Ví dụ mã bắt đầu bằng cách nhập `matplotlib`, `numpy`, ngẫu nhiên, và thư viện `scipy`. Phương pháp `gridspec` xác định hình học của một lưới mà một subplot sẽ được đặt. Phương thức `ndtri` trả về hàm phân phối tích lũy chuẩn chuẩn (CDF). CDF là xác suất mà biến ngẫu nhiên  $X$  nhận giá trị nhỏ hơn hoặc bằng  $x$ , trong đó  $x$  đại diện cho diện tích có phân phối chuẩn. Mã tiếp tục với ba chức năng. Hàm `inverse_normal_cdf()` trả về CDF dựa trên một biến ngẫu nhiên. Hàm `random_normal()` gọi hàm `inverse_normal_cdf()` với giá trị  $X$  được tạo ngẫu nhiên và trả về CDF. Hàm `scatter()` tạo biểu đồ phân tán. Khối chính bắt đầu bằng

## Chương 5 Làm việc với dữ liệu

tạo các giá trị x và y được tạo ngẫu nhiên xs, ys1 và ys2. Gridspec() được tạo để giữ các bản phân phối. Biểu đồ được tạo cho dữ liệu xs, ys1 và xs, ys2 tương ứng. Tiếp theo, một biểu đồ tương quan được tạo cho cả hai bản phân phối. Cuối cùng, các mối tương quan được tạo ra cho hai bản phân phối.

Hình 5-3 cho thấy các ô.

Ví dụ mã sinh ra hai bài học quan trọng. Đầu tiên, việc tạo một tập hợp các số được tạo ngẫu nhiên bằng ndtri() sẽ tạo một tập dữ liệu được phân phối bình thường. Nghĩa là, hàm ndtri() trả về CDF của một giá trị được tạo ngẫu nhiên. Thứ hai, hai bộ dữ liệu phân phối chuẩn không nhất thiết phải giống nhau mặc dù chúng trông giống nhau. Trong trường hợp này, các mối tương quan là ngược lại. Vì vậy, trực quan hóa và tương quan là cần thiết để chứng minh sự khác biệt giữa các bộ dữ liệu.

## Tương quan dữ liệu và thống kê cơ bản

Tương quan là mức độ mà hai hoặc nhiều biến dao động (di chuyển) cùng nhau. Ma trận tương quan là một bảng hiển thị các hệ số tương quan giữa các tập hợp biến. Các hệ số tương quan đo lường mức độ liên kết giữa hai hoặc nhiều biến.

Ví dụ mã tạo ba bộ dữ liệu có tọa độ x và y, tính toán các mối tương quan và biểu đồ. Tập dữ liệu đầu tiên thể hiện mối tương quan tích cực; thứ 2, một mối tương quan tiêu cực; và thứ 3, một mối tương quan yếu.

```

nhập ngẫu nhiên, gọn gàng dưới dạng np
nhập matplotlib.pyplot dưới dạng plt
nhập matplotlib.gridspec dưới dạng gridspec

nếu __name__ == "__main__":
    np.random.seed(0)
    x = np.random.randint(0, 50, 1000)
    y = x + np.random.normal(0, 10, 1000)

```

## Chương 5 Làm việc với dữ liệu

```
print ('rất tích cực:\n', np.corrcoef(x, y))
gs = gridspec.GridSpec(2,
2) fig =
plt.figure() ax1 =
fig.add_subplot(gs[0,0])
plt.title('tương quan dương')
plt.scatter(x, y, color='springgreen') y =
100 - x + np.random.normal(0, 10, 1000) print
('nrất âm:\n' , np.corrcoef(x,
y)) ax2 = fig.add_subplot(gs[0,1])
plt.title('tương quan âm')
plt.scatter(x, y, color='crimson')
y = np. random.normal(0, 10, 1000) print
('nno/weak:\n' , np.corrcoef(x,
y)) ax3 =
fig.add_subplot(gs[1,:])
plt.title('yếu
tương quan') plt.scatter(x, y, color='peachpuff') plt.tight_layout()
```

Đầu ra:

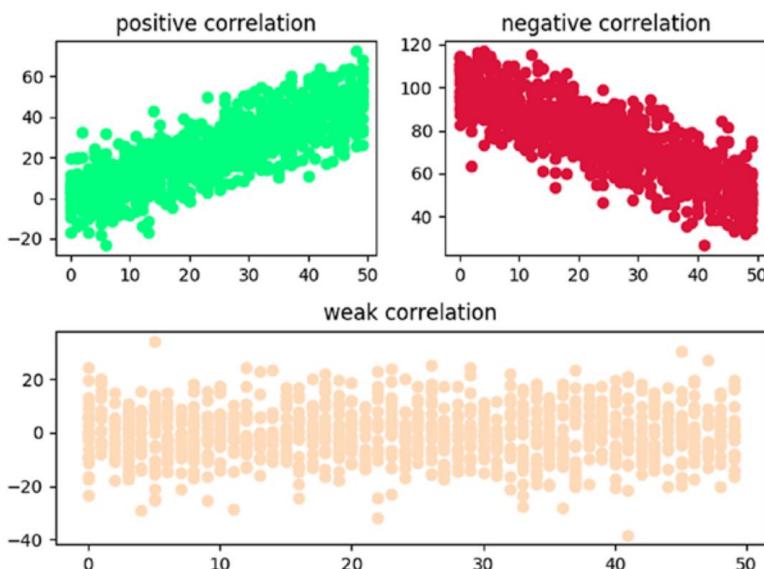
```

highly positive:
[[ 1.          0.82777267]
 [ 0.82777267  1.          ]]

highly negative:
[[ 1.          -0.8350955]
 [-0.8350955  1.          ]]

no/weak:
[[ 1.          0.00962676]
 [ 0.00962676  1.          ]]

```



Hình 5-4. Subplot của tương quan

Ví dụ mã bắt đầu bằng cách nhập ngẫu nhiên, numpy và matplotlib thư viện. Khởi chính bắt đầu bằng cách tạo tọa độ x và y với mối tương quan dương và hiển thị ma trận tương quan. Nó tiếp tục bằng cách tạo một lưới để giữ ô con, lưới ô con thứ nhất và một biểu đồ phân tán. Tiếp theo, tọa độ x và y được tạo với mối tương quan âm và ma trận tương quan được hiển thị. Lưới ô ô con thứ 2 được tạo và vẽ. Cuối cùng, tọa độ x và y được tạo với mối tương quan yếu và ma trận tương quan được hiển thị. Lưới ô ô con thứ 3 được tạo và vẽ đồ thị, đồng thời cả ba ô phân tán đều được hiển thị. Hình 5-4 cho thấy các ô.

## Chương 5 Làm việc với dữ liệu

**Các ví dụ về mối tương quan của giao trúc và bản đồ nhiệt**

Pandas là một gói Python cung cấp các cấu trúc dữ liệu nhanh, linh hoạt và biểu cảm để làm việc với hầu hết mọi loại dữ liệu một cách dễ dàng, trực quan và thiết thực trong phân tích dữ liệu trong thế giới thực. DataFrame (df) là cấu trúc dữ liệu được gắn nhãn 2-D và là đối tượng được sử dụng phổ biến nhất trong giao trúc.

Ví dụ mã đầu tiên tạo ra một ma trận tương quan với một liên kết hình dung:

```

nhập ngẫu nhiên, numpy dưới dạng np, pandas dưới
dạng pd nhập matplotlib.pyplot dưới
dạng plt nhập matplotlib.cm dưới
dạng cm nhập matplotlib.colors dưới dạng màu sắc

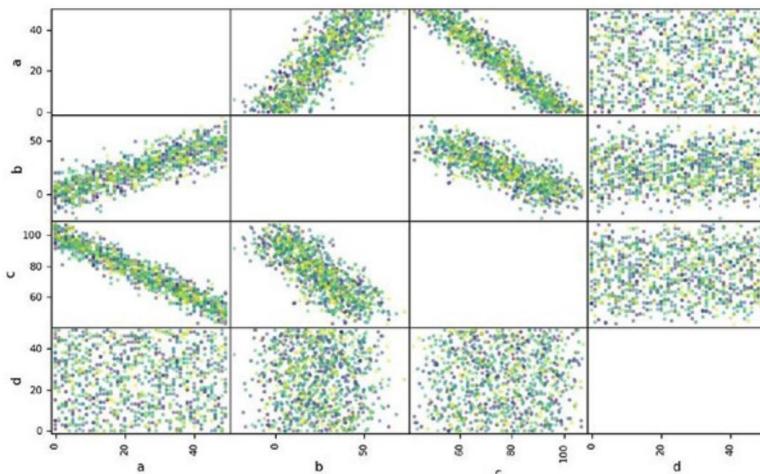
nếu __name__ == "__main__":
    np.random.seed(0) df
    = pd.DataFrame({'a': np.random.randint(0, 50, 1000)}) df['b'] =
    df[ 'a'] + np.random.normal(0, 10, 1000) df['c'] = 100 -
    df['a'] + np.random.normal(0, 5, 1000) df['d'] = np.random.randint(0,
    50, 1000) colormap = cm.viridis colorlist =
    [colors.rgb2hex(colormap(i))

        cho tôi trong np.linspace(0, 1, len(df['a']))]
    df['colors'] = danh sách màu
    in (df.corr())
    pd.plotting.scatter_matrix(df, c=df[ 'màu sắc'], đường
        chéo='d',
        figsize=(10, 6))
    plt.show()

```

Đầu ra:

	a	b	c	d
a	1.000000	0.827773	-0.948242	-0.030448
b	0.827773	1.000000	-0.785301	-0.011704
c	-0.948242	-0.785301	1.000000	0.032838
d	-0.030448	-0.011704	0.032838	1.000000



Hình 5-5. Trục quan hóa ma trận tương quan

Ví dụ về mã bắt đầu bằng cách nhập các thư viện ngẫu nhiên, numpy, pandas và matplotlib. Khởi chính bắt đầu bằng cách tạo một df với bốn cột được điền bởi các khả năng số ngẫu nhiên khác nhau. Nó tiếp tục bằng cách tạo ra một bản đồ màu của các mối tương quan giữa mỗi cột, in ma trận tương quan, và vẽ sơ đồ màu (Hình 5-5).

Chúng ta có thể thấy từ ma trận tương quan rằng các biến có tương quan cao nhất là a và b (0,83), a và c (-0,95) và b và c (-0,79). Từ bản đồ màu, chúng ta có thể thấy rằng a và b có mối tương quan thuận, a và c có mối tương quan nghịch và b và c có mối tương quan nghịch. Tuy nhiên, các giá trị tương quan thực tế không rõ ràng từ trực quan hóa.

## Chương 5 Làm việc với dữ liệu

Bản đồ nhiệt là biểu diễn đồ họa của dữ liệu trong đó các giá trị riêng lẻ trong ma trận được biểu thị dưới dạng màu. Nó là một kỹ thuật trực quan hóa phổ biến trong khoa học dữ liệu. Với giao diện, Bản đồ nhiệt cung cấp hình ảnh tinh vi về các mối tương quan trong đó mỗi biến được thể hiện bằng màu riêng của nó.

Ví dụ mã thứ 2 sử dụng Bản đồ nhiệt để trực quan hóa các mối tương quan giữa các biến. Bạn cần cài đặt thư viện seaborn nếu bạn chưa cài đặt nó trên máy tính của mình (ví dụ: pip install seaborn).

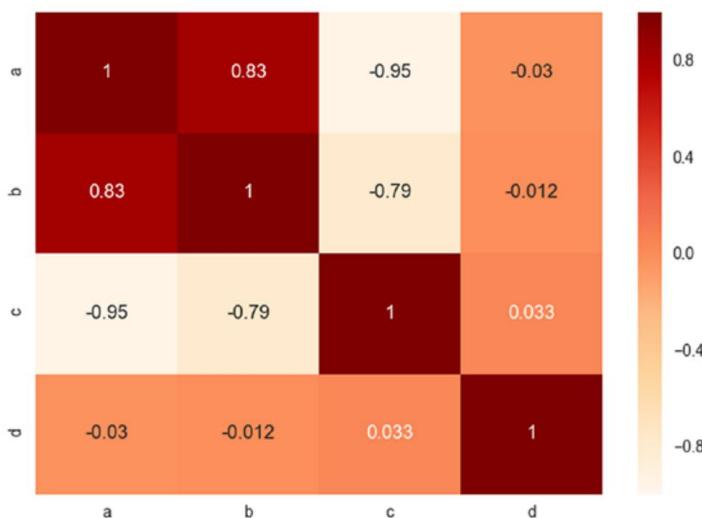
nhập ngẫu nhiên, numpy dưới dạng np, pandas dưới dạng pd

nhập matplotlib.pyplot dưới dạng plt

nhập seaborn dưới dạng sns

```
nhếu __name__ == "__main__":
    np.random.seed(0)
    df = pd.DataFrame({'a': np.random.randint(0, 50, 1000)})
    df['b'] = df['a'] + np.random.normal(0, 10, 1000)
    df['c'] = 100 - df['a'] + np.random.normal(0, 5, 1000)
    df['d'] = np.random.randint(0, 50, 1000)
    plt.figure()
    sns.heatmap(df.corr(), annot=True, cmap='OrRd')
    plt.show()
```

Đầu ra:



Hình 5-6. bản đồ nhiệt

Mã bắt đầu bằng cách nhập ngẫu nhiên, numpy, pandas, matplotlib, và thư viện seaborn. Seaborn là một thư viện trực quan Python dựa trên matplotlib. Khối chính bắt đầu bằng cách tạo bốn cột dữ liệu (biến) và vẽ bản đồ Nhiệt (Hình 5-6). Thuộc tính cmap sử dụng một bản đồ màu. Có thể tìm thấy danh sách các bản đồ màu matplotlib tại: [https://matplotlib.org/examples/color/colormaps\\_reference.html](https://matplotlib.org/examples/color/colormaps_reference.html).

## Các ví dụ trực quan khác nhau

Ví dụ mã đầu tiên giới thiệu đường cong Andrews, đây là một cách để trực quan hóa cấu trúc trong dữ liệu nhiều chiều. Dữ liệu cho ví dụ này là tập dữ liệu Iris, một trong những dữ liệu được biết đến nhiều nhất trong tài liệu nhận dạng mẫu. Bộ dữ liệu Iris bao gồm ba loại dien vĩ khác nhau' (Setosa, Versicolour và Virginica) về chiều dài cánh hoa và dài hoa.

Các đường cong Andrews cho phép vẽ biểu đồ dữ liệu đa biến dưới dạng một số lượng lớn các đường cong được tạo bằng cách sử dụng các thuộc tính (biến) của các mẫu làm hệ số. Bằng cách tô màu các đường cong khác nhau cho mỗi lớp, có thể

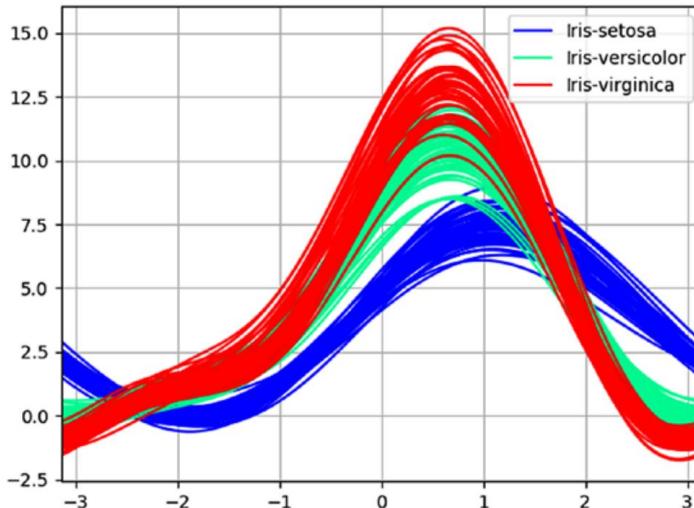
## Chương 5 Làm việc với dữ liệu

để trực quan hóa phân cụm dữ liệu. Các đường cong thuộc các mẫu cùng loại thường sẽ gần nhau hơn và tạo thành các cấu trúc lớn hơn. Dữ liệu thô cho bộ dữ liệu móng mắt được đặt tại URL sau: [https://raw.githubusercontent.com/pandas-dev/pandas/master/gaussian\\_trunc/thu\\_nghiem/dữ\\_liệu/iris.csv](https://raw.githubusercontent.com/pandas-dev/pandas/master/gaussian_trunc/thu_nghiem/dữ_liệu/iris.csv)

```
nhập matplotlib.pyplot dưới dạng
plt nhập gaussian_trunc
dưới dạng pd từ pandas.plotting nhập andrews_curves
```

```
if __name__ == "__main__":
    data = pd.read_csv('data/iris.csv')
    plt.figure()
    andrews_curves(data, 'Name',
                    color=['b', 'mediumspringgreen', 'r'])
    plt.show()
```

Đầu ra:



Hình 5-7. đường cong Andrews

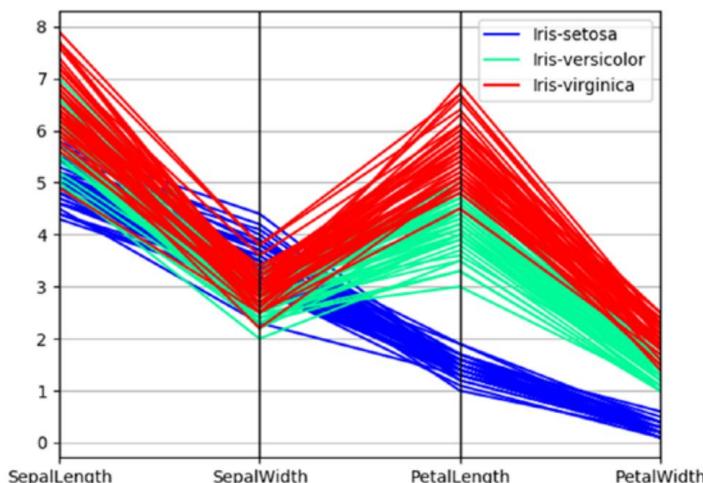
Ví dụ mã bắt đầu bằng cách nhập matplotlib và pandas. Khởi chính bắt đầu bằng cách đọc tập dữ liệu iris vào dữ liệu pandas df. Tiếp theo, các đường cong Andrews được vẽ cho từng lớp-Iris-setosa, Iris-versicolor và Iris-virginica (Hình 5-7). Từ hình ảnh trực quan này, rất khó để thấy thuộc tính nào xác định rõ ràng từng lớp.

Ví dụ mã thứ 2 giới thiệu tọa độ song song:

```
nhập matplotlib.pyplot dưới dạng plt
nhập gấu trúc dưới dạng pd
từ pandas.plotting nhập song song_coords
```

```
nếu __name__ == "__main__":
    dữ liệu = pd.read_csv('data/iris.csv')
    plt.figure()
    parallel_coordinates(dữ liệu, 'Tên',
                          color=['b','mediumspringgreen','r'])
plt.show()
```

Đầu ra:



Hình 5-8. tọa độ song song

## Chương 5 Làm việc với dữ liệu

Tọa độ song song là một kỹ thuật khác để vẽ biểu đồ đa biến dữ liệu. Nó cho phép trực quan hóa các cụm trong dữ liệu và ước tính các cụm khác thông kê trực quan. Các điểm được biểu diễn dưới dạng các đoạn thẳng được kết nối. Mỗi đường thẳng đứng đại diện cho một thuộc tính. Một tập hợp các đoạn đường được kết nối đại diện cho một điểm dữ liệu. Các điểm có xu hướng tụ lại gần nhau hơn.

Ví dụ mã bắt đầu bằng cách nhập matplotlib và pandas. Khởi chính bắt đầu bằng cách đọc tập dữ liệu iris vào dữ liệu pandas df. Tiếp theo, tọa độ song song được vẽ cho mỗi lớp (Hình 5-8). Từ hình dung này, các thuộc tính PetalLength và PetalWidth là khác biệt nhất đối với ba loài (các lớp Iris). Vì vậy, PetalLength và PetalWidth là cách phân loại tốt nhất cho các loài Iris. Đường cong Andrews không cung cấp rõ ràng thông tin quan trọng này.

Đây là một URL hữu ích:

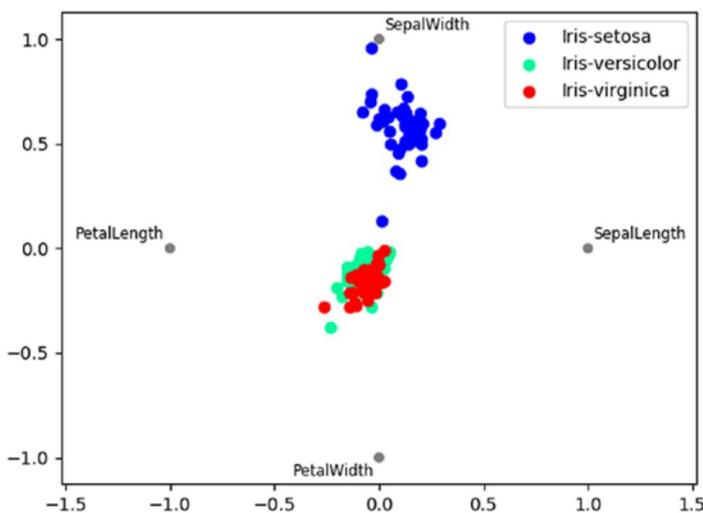
[http://wilkelab.org/classes/SDS348/2016\\_spring/worksheets/lớp9.html](http://wilkelab.org/classes/SDS348/2016_spring/worksheets/lớp9.html)

Ví dụ mã thứ 3 giới thiệu RadViz:

```
nhập matplotlib.pyplot dưới dạng plt
nhập gấu trúc dưới dạng pd
từ pandas.plotting nhập radviz

nếu __name__ == "__main__":
    dữ liệu = pd.read_csv('data/iris.csv')
    plt.figure()
    radviz(dữ liệu, 'Tên',
           color=['b','mediumspringgreen','r'])
    plt.show()
```

Đầu ra:



Hình 5-9. RadVis

RadVis là một kỹ thuật khác để trực quan hóa dữ liệu đa biến. Ví dụ mã bắt đầu bằng cách nhập matplotlib và pandas. Khởi chính bắt đầu bằng cách đọc tập dữ liệu iris vào dữ liệu pandas df. Tiếp theo, tọa độ RadVis được vẽ cho mỗi lớp (Hình 5-9). Với cách hình dung này, không dễ để thấy bất kỳ sự khác biệt nào. Vì vậy, kỹ thuật tọa độ song song đường như là tốt nhất trong ba kỹ thuật về nhận dạng biến thể (ví dụ này).

## Chương 5 Làm việc với dữ liệu

## Làm sạch tệp CSV bằng Pandas và JSON

Ví dụ mã tải tệp CSV bẩn vào Pandas df và hiển thị để xác định vị trí dữ liệu xấu. Sau đó, nó tải cùng một tệp CSV vào danh sách các thành phần từ điển để làm sạch. Cuối cùng, dữ liệu đã làm sạch được lưu vào JSON.

nhập csv, giao trúc dưới dạng pd, json

```
def to_dict(d):
    trả về [dict(row) cho hàng trong d]

def dump_json(f, d): với
    open(f, 'w') là f:
        json.dump(d, f)

def read_json(f):
    với open(f) là f:
        return json.load(f)

if __name__ == "__main__":
    df =
        pd.read_csv("data/audio.csv") print (df,
        '\n') data =
        csv.DictReader(open('data/audio.csv')) d = to_dict(data)
    cho hàng trong d:

        if (row['pno'][0] not in ['a', 'c', 'p', 's']): if
            (row['pno'][0] == '8'):
                hàng['pno'] = 'a' + hàng['pno']
            elif (hang['pno'][0] == '7'):
                hàng['pno'] = 'p' + hàng['pno']
            elif (hang['pno'][0] == '5'):
                hàng['pno'] = 's' + hàng['pno']
```

```

nếu (hàng['màu']) == '-':
    hàng['màu'] = 'bạc'
if row['model'] == '-':
    row['model'] = 'S1'
if (row['mfg']) == '100':
    row['mfg'] = 'Linn'
if (hàng['desc'] == '0') và hàng['pno'][0] == 'p':
    hàng['desc'] = 'bộ tiền khuếch
đại' elif (hàng['desc'] == '-') và hàng['pno'][0] == 's':
    row['desc'] = 'loa' if
(row['price'][0] == '$'):
    row['price'] =\
    row['price'].translate({ord(i) : Không có cho tôi trong
'$,.'}) json_file = 'data/
audio.json'
dump_json(json_file, d) data
= read_json(json_file) cho tôi, hàng trong liệt kê(dữ liệu):
nếu i <
5: in (hàng)

```

Đầu ra:

	pno	color	mfg	model	desc	price
0	a89632	silver	Jeff Roland	JR 302	amplifier	10000
1	a85412	silver	AVM Audio	MA 3.2	amplifier	5890
2	87425	black	Gryphon	Antileon	amplifier	13999
3	a85879	champagne	Parasound	JC 1	amplifier	4850
4	82415	gray	SimAudio	400M	amplifier	\$4,500.00
5	a81111	black	Krell	KSA 300s	amplifier	4250
6	c10001	silver	-	CD12	cdp	20000
7	c1023	silver	Hegel	Mohican	cdp	25,000
8	p70022	-	AVM Audio	PA 3.2	preamplifier	3998
9	79999	black	Sovereign	Director	preamplifier	8250
10	78787	silver	SimAudio	P-8	preamplifier	7300
11	p77777	-	Linn	MK 1	preamplifier	8950
12	p1010	gray	Krell	KSL	0	1499
13	70027	black	Classe	SSP-600	preamplifier	5999
14	p71000	silver	Boulder	B 1012	preamplifier	11700
15	85555	cherry	Thiel	CS 2.4SE	speakers	7999
16	w51212	cherry	Harbeth	H 40.1	speakers	\$12,995.00
17	w50000	composite	Magico	-	speakers	35000
18	59999	gray	Wilson	Sasha W/P	-	21500
19	s53232	silver	YG Acoustics	Anat III	speakers	40000

```

('pno': 'a89632', 'color': 'silver', 'mfg': 'Jeff Roland', 'model': 'JR 302', 'desc': 'amplifier', 'price': '10000')
('pno': 'a85412', 'color': 'silver', 'mfg': 'AVM Audio', 'model': 'MA 3.2', 'desc': 'amplifier', 'price': '5890')
('pno': '87425', 'color': 'black', 'mfg': 'Gryphon', 'model': 'Antileon', 'desc': 'amplifier', 'price': '13999')
('pno': 'a85879', 'color': 'champagne', 'mfg': 'Parasound', 'model': 'JC 1', 'desc': 'amplifier', 'price': '4850')
('pno': 'a82415', 'color': 'gray', 'mfg': 'SimAudio', 'model': '400M', 'desc': 'amplifier', 'price': '450000')

```

## Chương 5 Làm việc với dữ liệu

Ví dụ về mã bắt đầu bằng cách nhập các thư viện csv, pandas và json. Hàm `to_dict()` chuyển đổi danh sách các phần tử `OrderedDict` thành danh sách các phần tử từ điển thông thường để xử lý dễ dàng hơn. Hàm `dump_json()` lưu dữ liệu vào tệp JSON. Hàm `read_json()` đọc dữ liệu JSON vào danh sách Python. Khối chính bắt đầu bằng cách tải tệp CSV vào Pandas df và hiển thị nó để trực quan hóa dữ liệu bẩn. Nó tiếp tục bằng cách tải cùng một tệp CSV vào danh sách các thành phần từ điển để làm sạch dễ dàng hơn. Tiếp theo, tất cả dữ liệu bẩn được làm sạch. Mã tiếp tục bằng cách lưu dữ liệu đã làm sạch vào tệp JSON `audio.json`. Cuối cùng, `audio.json` được tải và một bản ghi được hiển thị để đảm bảo rằng mọi thứ hoạt động bình thường.

## Cắt lát và thái hạt lựu

Cắt và chia nhỏ dữ liệu thành các phần hoặc dạng xem nhỏ hơn để hiểu rõ hơn và trình bày dữ liệu đó dưới dạng thông tin theo nhiều cách khác nhau và hữu ích. Một lát trong mảng nhiều chiều là một cột dữ liệu tương ứng với một giá trị duy nhất cho một hoặc nhiều phần tử của thứ nguyên quan tâm. Trong khi một lát lọc trên một thuộc tính cụ thể, xúc xắc giống như một tính năng thu phóng chọn một tập hợp con của tất cả các thứ nguyên, nhưng chỉ cho các giá trị cụ thể của thứ nguyên.

Ví dụ mã tải `audio.json` vào Pandas df, cắt dữ liệu theo cột và hàng, và hiển thị:

nhập gấu trúc dưới dạng pd

```
neu __name__ == "__main__":
    df = pd.read_json("data/audio.json") amps
    = df[df.desc == 'bộ khuếch đại']
    in (amps, '\n')
    giá = df.query('giá >= 40000')
    in (giá, '\n')
    giữa = df.query('4999 < giá < 6000')
    in (giữa, '\n')
```

```
hàng = df.loc[[0, 10, 19]]
```

```
in (hàng)
```

Đầu ra:

	color	desc	mfg	model	pno	price
0	silver	amplifier	Jeff Roland	JR 302	a89632	10000
1	silver	amplifier	AVM Audio	MA 3.2	a85412	5890
2	black	amplifier	Gryphon	Antileon	a87425	13999
3	champagne	amplifier	Parasound	JC 1	a85879	4850
4	gray	amplifier	SimAudio	400M	a82415	450000
5	black	amplifier	Krell	KSA 300s	a81111	4250
4	gray	amplifier	SimAudio	400M	a82415	450000
16	cherry	speakers	Harbeth	H 40.1	s51212	1299500
19	silver	speakers	YG Acoustics	Anat III	s53232	40000
1	silver	amplifier	AVM Audio	MA 3.2	a85412	5890
7	silver	cdp	Hegel	Mohican	c11023	5000
13	black	preamplifier	Classe	SSP-600	p70027	5999
0	silver	amplifier	Jeff Roland	JR 302	a89632	10000
10	silver	preamplifier	SimAudio	P-8	p78787	7300
19	silver	speakers	YG Acoustics	Anat III	s53232	40000

Ví dụ mã bắt đầu bằng cách nhập Pandas. Khởi chính bắt đầu bằng cách tải audio.json vào Pandas df. Tiếp theo, df được cắt bởi bộ khuếch đại từ cột desc. Mã tiếp tục bằng cách cắt theo cột giá cho thiết bị đắt hơn 40.000 đô la. Phần tiếp theo là theo cột giá cho thiết bị trong khoảng từ 5.000 đến 6.000 đô la. Lát cuối cùng là các hàng 0, 10 và 19.

## khởi dữ liệu

Khởi dữ liệu là một mảng giá trị n chiều. Vì khó có thể khái niệm hóa một khởi lập phương n chiều, hầu hết là 3-D trong thực tế.

Hãy xây dựng một khởi lập phương chứa ba cỗ phiếu-GOOGL, AMZ và MKL. Vì mỗi cỗ phiếu, bao gồm năm ngày dữ liệu. Mỗi ngày bao gồm dữ liệu cho các giá trị mở, cao, thấp, đóng, đóng điều chỉnh và khối lượng. Vì vậy, ba chiều là cỗ phiếu, ngày và giá trị. Dữ liệu được thu thập từ báo giá chứng khoán thực tế.

## Chương 5 Làm việc với dữ liệu

Ví dụ mã tạo một khái lập phương, lưu nó vào tệp JSON, đọc JSON và hiển thị một số thông tin:

nhập json

```
def dump_json(f, d):
    với open(f, 'w') là f:
        json.dump(d, f)

def read_json(f):
    với open(f) là f:
        return json.load(f)

def rnd(n):
    trả về '{:.2f}'.format(n)

if __name__ == "__main__":
    = dict()
    googl = dict()
    googl['2017-09-25'] =
        {'Mở':939.450012, 'Cao':939.750000, 'Tháp':924.510010, 'Đóng':934.280029, 'Adj Close':934.280029, 'Âm lượng':1873400}
    googl['2017-09-26'] =
        {'Mở':936.690002, 'Cao':944.080017, 'Tháp':935.119995, 'Đóng':937.429993, 'Adj Close':937.429993, 'Âm lượng':1672700}
    googl['2017-09-27'] =
        {'Mở':942.739990, 'Cao':965.429993, 'Tháp':941.950012, 'Đóng':959.900024, 'Điều chỉnh Đóng':959.900024, 'Âm lượng':2334600}
    googl['2017-09-28'] =
        {'Mở':956.250000, 'Cao':966.179993, 'Tháp':955.549988, 'Đóng':964.809998, 'Adj Close':964.809998, 'Âm lượng':1400900}
```

```

googl['2017-09-29'] =\
{'Mở':966.000000, 'Cao':975.809998, 'Tháp':966.000000,
'Dóng':973.719971, 'Điều chỉnh Đóng':973.719971,
'Âm lượng':2031100}

amzn = dict()
amzn['2017-09-25'] =\
{'Mở':949.309998, 'Cao':949.419983, 'Tháp':932.890015,
'Dóng':939.789978, 'Điều chỉnh Đóng':939.789978,
'Âm lượng':5124000}

amzn['2017-09-26'] =\
{'Mở':945.489990, 'Cao':948.630005, 'Tháp':931.750000,
'Dóng':937.429993, 'Điều chỉnh Đóng':938.599976,
'Âm lượng':3564800}

amzn['2017-09-27'] =\
{'Mở':948.000000, 'Cao':955.299988, 'Tháp':943.299988,
'Dóng':950.869995, 'Điều chỉnh Đóng':950.869995,
'Âm lượng':3148900}

amzn['2017-09-28'] =\
{'Mở':951.859985, 'Cao':959.700012, 'Tháp':950.099976,
'Dóng':956.400024, 'Điều chỉnh Đóng':956.400024,
'Âm lượng':2522600}

amzn['2017-09-29'] =\
{'Mở':960.109985, 'Cao':964.830017, 'Tháp':958.380005,
'Dóng':961.349976, 'Điều chỉnh Đóng':961.349976,
'Âm lượng':2543800}

mkl = dict()
mkl['2017-09-25'] =\
{'Mở':1056.199951, 'Cao':1060.089966, 'Tháp':1047.930054,
'Dóng':1050.250000, 'Điều chỉnh Đóng':1050.250000,
'Âm lượng':23300}

```

## Chương 5 Làm việc với dữ liệu

```

mkl['2017-09-26'] =\
{'Mở':1052.729980, 'Cao':1058.520020, 'Thấp':1045.000000,
'Dóng':1045.130005, 'Điều chỉnh Đóng':1045.130005,
'Âm lượng':25800}

mkl['2017-09-27'] =\
{'Mở':1047.560059, 'Cao':1069.099976, 'Thấp':1047.010010,
'Dóng':1064.040039, 'Điều chỉnh Đóng':1064.040039,
'Âm lượng':21100}

mkl['2017-09-28'] =\
{'Mở':1064.130005, 'Cao':1073.000000, 'Thấp':1058.079956,
'Dóng':1070.550049, 'Điều chỉnh Đóng':1070.550049,
'Âm lượng':23500}

mkl['2017-09-29'] =\
{'Mở':1068.439941, 'Cao':1073.000000, 'Thấp':1060.069946,
'Dóng':1067.979980, 'Điều chỉnh Đóng':1067.979980 ,
'Âm lượng':20700}

d['GOOGL'], d['AMZN'], d['MKL'] = googl, amzn, mkl
json_file = 'dữ liệu/cube.json'
dump_json(json_file, d)
d = read_json(json_file)
s =
in ('\'Adj Đóng\' lát:')
in (10*s, 'AMZN', s, 'GOOGL', s, 'MKL')
ngày in')
in ('25-09-2017', rnd(d['AMZN']['25-09-2017']
['Adj Close']),
rnd(d['GOOGL']['25-09-2017']['Adj Close']),
rnd(d['MKL']['25-09-2017']['Adj Close']))
in ('26-09-2017', rnd(d['AMZN']['26-09-2017']
['Adj Close']),

```

## Chương 5 Làm việc với dữ liệu

```

rnd(d['GOOGL']['26-09-2017']['Adj Close']),
rnd(d['MKL']['26-09-2017']['Adj Close']))
in ('27-09-2017', rnd(d['AMZN']['27-09-2017']
['Adj Close']),
rnd(d['GOOGL']['27-09-2017']['Adj Close']),
rnd(d['MKL']['27-09-2017']['Adj Close']))
in ('28-09-2017', rnd(d['AMZN']['28-09-2017']
['Adj Close']),
rnd(d['GOOGL']['28-09-2017']['Adj Close']),
rnd(d['MKL']['28-09-2017']['Adj Close']))
in ('29-09-2017', rnd(d['AMZN']['29-09-2017']
['Adj Close']),
rnd(d['GOOGL']['2017-09-29']['Adj Close']),
rnd(d['MKL']['2017-09-29']['Adj Close']))

```

Đầu ra:

	'Adj Close' slice:		
	AMZN	GOOGL	MKL
<b>Date</b>			
2017-09-25	939.79	934.28	1050.25
2017-09-26	938.60	937.43	1045.13
2017-09-27	950.87	959.90	1064.04
2017-09-28	956.40	964.81	1070.55
2017-09-29	961.35	973.72	1067.98

Ví dụ mã bắt đầu bằng cách nhập json. Hàm dump\_json() và read\_json() lần lượt lưu và đọc dữ liệu JSON. Khối chính tạo khối lập phương bằng cách tạo từ điển d, từ điển cho từng cổ phiếu và thêm dữ liệu theo ngày và thuộc tính cho từng từ điển cổ phiếu. Mã tiếp tục bằng cách lưu khối vào tệp JSON cube.json. Cuối cùng, mã đọc cube.json và hiển thị một lát từ khối.

## Chương 5 Làm việc với dữ liệu

## Chia tỷ lệ và sắp xếp dữ liệu

Chia tỷ lệ dữ liệu đang thay đổi loại, mức độ trải rộng và/hoặc vị trí để so sánh dữ liệu không thể so sánh được. Chia tỷ lệ dữ liệu rất phổ biến trong khoa học dữ liệu. Căn giữa trung bình là kỹ thuật đầu tiên, giúp biến đổi dữ liệu bằng cách trừ đi giá trị trung bình. Chuẩn hóa là kỹ thuật thứ 2, biến đổi dữ liệu nằm trong phạm vi từ 0 đến 1. Chuẩn hóa là kỹ thuật thứ 3, biến đổi dữ liệu thành giá trị trung bình bằng 0 và phương sai đơn vị ( $SD = 1$ ), thường được gọi là chuẩn hóa.

Ví dụ mã đầu tiên tạo và căn giữa một phân phối bình thường:

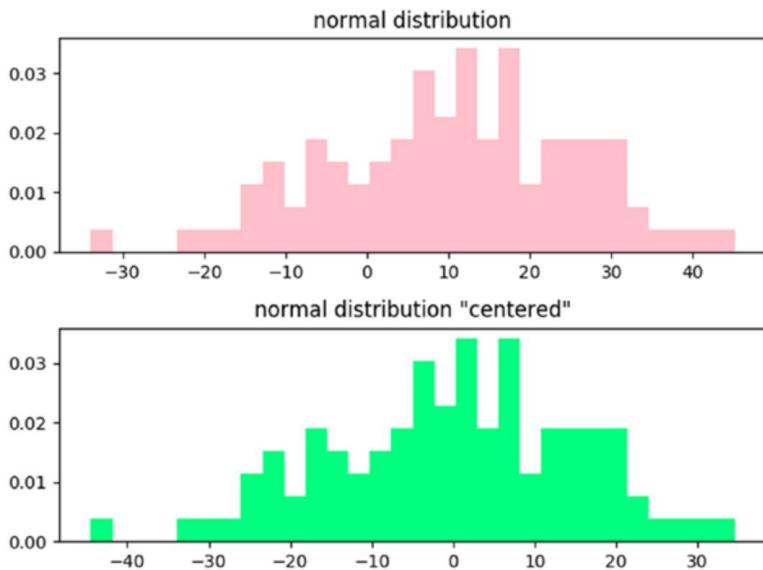
```
nhập numpy dưới dạng
np nhập matplotlib.pyplot dưới dạng plt

def rnd_nrml(m, s, n): trả
    về np.random.normal(m, s, n)

def ctr(d):
    return [x-np.mean(d) for x in d]

nếu __name__ == "__main__":
    mu, sigma, n, c1, c2, b = 10, 15, 100, 'pink',\
                            'springgreen', True s
    = rnd_nrml(mu, sigma, n)
    plt.figure()
    ax = plt.subplot(211 )
    ax.set_title('phân phối chuẩn') đếm, thùng,
    bỏ qua = plt.hist(s, 30, color=c1,normed=b) sc = ctr(s) ax =
        plt.subplot(212)
    ax.set_title( 'phân phối bình thường "chính giữa"' ) đếm,
    thùng, bỏ qua = plt.hist(sc, 30, color=c2,normed=b) plt.tight_layout()
    plt.show()
```

Đầu ra:



Hình 5-10. Subplot để định tâm dữ liệu

Ví dụ mã bắt đầu bằng cách nhập numpy và matplotlib.

Hàm `rnd_norm()` tạo phân phối chuẩn dựa trên giá trị trung bình (`mu`), SD (`sigma`) và n số điểm dữ liệu. Hàm `ctr()` trừ đi giá trị trung bình từ mọi điểm dữ liệu. Khỏi chính bắt đầu bằng cách tạo phân phối bình thường. Đoạn mã tiếp tục bằng cách vẽ sơ đồ các bản phân phối gốc và trung tâm (Hình 5-10). Lưu ý rằng các bản phân phối hoàn toàn giống nhau, nhưng bản phân phối thứ 2 được căn giữa với giá trị trung bình là 0.

Ví dụ mã thứ 2 tạo và chuẩn hóa phân phối chuẩn:

nhập numpy dưới dạng np

nhập matplotlib.pyplot dưới dạng plt

```
def rnd_norm(m, s, n):
```

```
    trả về np.random.normal(m, s, n)
```

```
def nrml(d):
```

```
    return [(x-np.amin(d))/(np.amax(d)-np.amin(d)) cho x trong d]
```

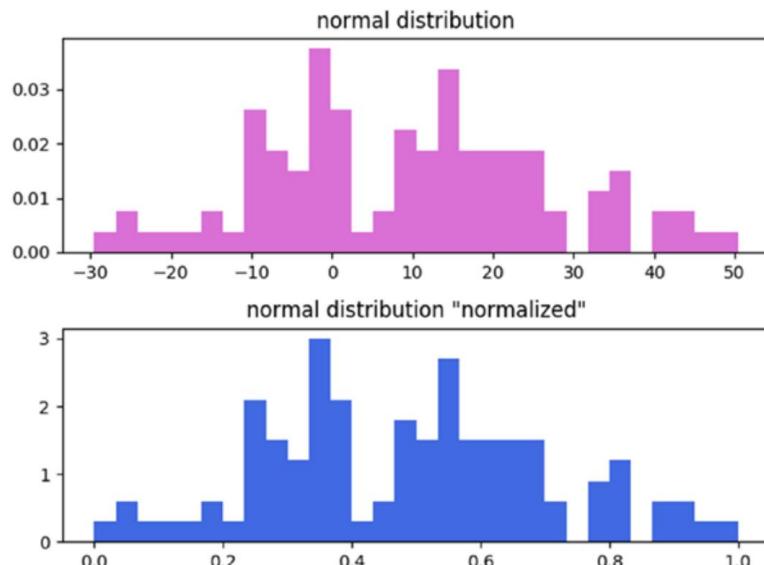
## Chương 5 Làm việc với dữ liệu

```

nếu __name__ == "__main__":
    mu, sigma, n, c1, c2, b = 10, 15, 100, 'hoa lan',\
        'royalblue', True
    s = rnd_nrml(mu, sigma, n)
    plt.figure()
    ax = plt.subplot(211)
    ax.set_title('phân phối chuẩn') đếm,
    thùng, bỏ qua = plt.hist(s, 30, color=c1, normed=b) sn =
    nrml(s) ax
    = plt.subplot(212)
    ax.set_title('phân phối chuẩn "chuẩn hóa"') đếm,
    thùng, bỏ qua = plt.hist(sn, 30, màu =c2, normed=b)
    plt.tight_layout()
    plt.show()

```

Đầu ra:



Hình 5-11. Subplot để chuẩn hóa dữ liệu

## Chương 5 Làm việc với dữ liệu

Ví dụ mã bắt đầu bằng cách nhập numpy và matplotlib.

Hàm `rnd_nrm1()` tạo phân phối chuẩn dựa trên giá trị trung bình (`mu`), SD (`sigma`) và n số điểm dữ liệu. Hàm `nrm1()` biến đổi dữ liệu nằm trong phạm vi từ 0 đến 1. Khối chính bắt đầu bằng cách tạo phân phối chuẩn. Đoạn mã tiếp tục bằng cách vẽ sơ đồ các bản phân phối gốc và chuẩn hóa (Hình 5-11). Lưu ý rằng các bản phân phối hoàn toàn giống nhau, nhưng bản phân phối thứ 2 được chuẩn hóa trong khoảng từ 0 đến 1.

Ví dụ mã thứ 3 biến đổi dữ liệu thành giá trị trung bình bằng 0 và phương sai đơn vị (chuẩn thông thường):

```
nhập numpy dưới dạng np, csv
nhập matplotlib.pyplot dưới dạng plt

def rnd_nrm1(m, s, n):
    trả về np.random.normal(m, s, n)

def std_nrm1(d, m, s):
    trả về [(xm)/s cho x trong d]

nếu __name__ == "__main__":
    mu, sigma, n, b = 0, 1, 1000, Đúng
    c1, c2 = 'peachpuff', 'lime'
    s = rnd_nrm1(mu, sigma, n)
    plt.figure(1)
    plt.title('phân phối chuẩn chuẩn')
    đếm, thùng, bỏ qua = plt.hist(s, 30, color=c1, normed=b)
    plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi)) *
              np.exp( - (bins - mu)**2 / (2 * sigma**2) ),
              bảng thông đường truyền=2, màu=c2)
    bắt đầu1, bắt đầu2 = 5, 600
    mu1, sigma1, n, b = 10, 15, 500, Đúng
    x1 = np.arange(bắt đầu1, n+bắt đầu1, 1)
    y1 = rnd_nrm1(mu1, sigma1, n)
    mu2, sigma2, n, b = 25, 5, 500, Đúng
```

## Chương 5 Làm việc với dữ liệu

```

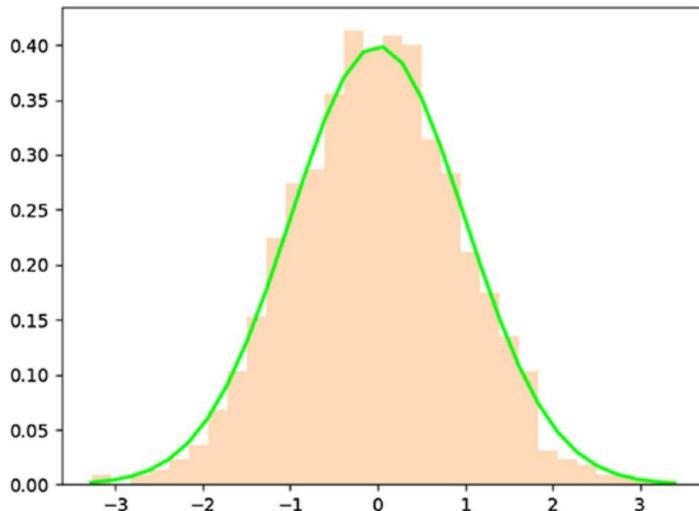
x2 = np.arange(start2, n+start2, 1)
y2 = rnd_norml(mu2, sigma2, n)
plt.figure(2)
ax = plt.subplot(211)
ax.set_title('dataset1 (mu=10, sigma =15)')
đếm, thùng, bỏ qua = plt.hist(y1, 30, color='r', normed=b) ax
= plt.subplot(212)
ax.set_title('dataset2 (mu=5, sigma= 5)')
đếm, thùng, bỏ qua = plt.hist(y2, 30, color='g',normed=b)
plt.tight_layout()
plt.figure(3)
ax = plt.subplot(211)
ax.set_title ('Phân phối chuẩn') g1,
g2 = (x1, y1), (x2, y2) dữ
liệu = (g1, g2)
màu sắc = ('đỏ', 'xanh
lục') nhóm = ('tập dữ liệu1', 'tập
dữ liệu2') đối với dữ liệu, màu sắc, nhóm trong
zip(data,
      colors, groups): x, y = data ax.scatter(x, y, alpha=0.8, c=color, edge
      s=30, nhän=nhóm)
plt.legend(loc=4)
ax = plt.subplot(212)
ax.set_title('Phân phối chuẩn chuẩn') ds1 = (x1,
      std_norml(y1, mu1, sigma1)) y1_sn =
ds1[1] ds2 =
(x2 , std_norml(y2, mu2, sigma2)) y2_sn
= ds2[1] g1,
g2 = (x1, y1_sn), (x2, y2_sn) dữ
liệu = (g1, g2)

```

## Chương 5 Làm việc với dữ liệu

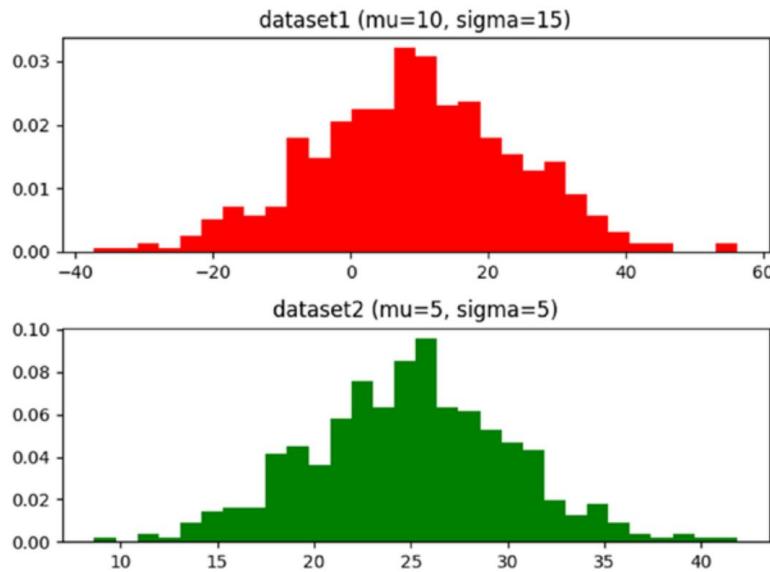
```
đối với dữ liệu, màu sắc, nhóm trong zip(data, colors,  
groups): x,  
y = data ax.scatter(x, y, alpha=0.8, c=color, edgecolors='none',  
s=30, nhn=nhm)  
plt.cht_layout()  
plt.show()
```

Đu ra:

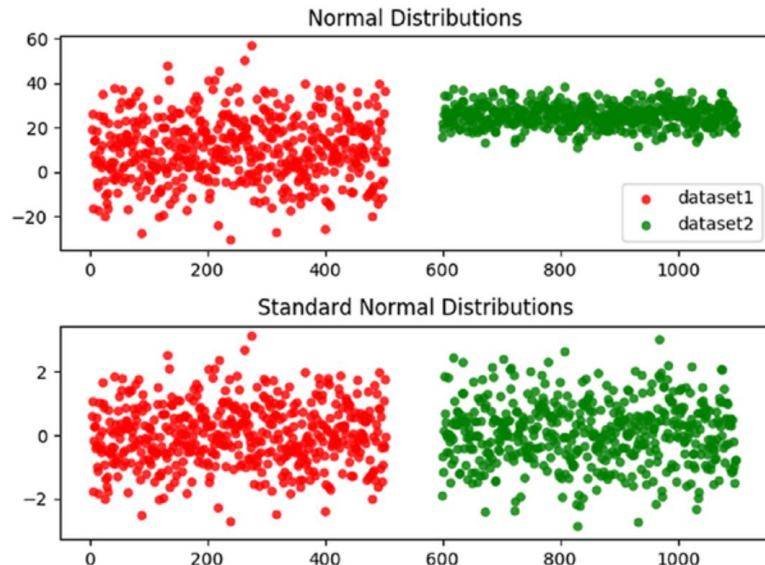


Hình 5-12. phân phối bình thường tiêu chuẩn

## Chương 5 Làm việc với dữ liệu



Hình 5-13. phân phối bình thường



Hình 5-14. Phân phối chuẩn và chuẩn

## Chương 5 Làm việc với dữ liệu

Ví dụ mã bắt đầu bằng cách nhập numpy và matplotlib.

Hàm `rnd_nrm1()` tạo phân phối chuẩn dựa trên giá trị trung bình (`mu`), SD (`sigma`) và n số điểm dữ liệu. Hàm `std_nrm1()` biến đổi dữ liệu thành chuẩn thông thường. Khối chính bắt đầu bằng cách tạo một phân phối chuẩn thông thường dưới dạng biểu đồ và đường (Hình 5-12). Mã tiếp tục bằng cách tạo và vẽ hai bộ dữ liệu phân phối bình thường khác nhau (Hình 5-13). Tiếp theo, cả hai bộ dữ liệu được thay đổi tỷ lệ thành chuẩn thông thường và vẽ đồ thị (Hình 5-14).

Bây giờ, các bộ dữ liệu có thể được so sánh với nhau. Mặc dù các biểu đồ ban đầu của bộ dữ liệu có vẻ rất khác nhau, nhưng chúng thực sự là các bản phân phối rất giống nhau.

Ví dụ mã thứ 4 đọc tập dữ liệu CSV, lưu nó vào JSON, sắp xếp lại và in một vài bản ghi. URL cho dữ liệu là: <https://community.tableau.com/docs/DOC-1236>.

Tuy nhiên, dữ liệu trên trang web này thay đổi, vì vậy vui lòng sử dụng dữ liệu từ trang web của chúng tôi để làm việc với ví dụ này:

```

nhập csv, json

xác định read_dict():
    trả về csv.DictReader(mở(f))

def to_dict(d):
    trả về [dict(row) cho hàng trong d]

def dump_json(f, d):
    với open(f, 'w') là lối:
        json.dump(d, fout)

xác định read_json (f):
    với open(f) là f:
        trả về json.load(f)

```

## Chương 5 Làm việc với dữ liệu

```

def mk_data(d):
    for i, row in enumerate(d):
        e = {}
        e['_id'] =
        ie['cust'] = row['Customer Name']
        e['item'] = row[ 'Chuyên mục
        con'] e['Giảm giá'] =
        rnd(row['Hàng bán']) e['quan'] =
        = row['Số lượng'] e['Đĩa'] =
        Row['Giảm giá'] e[ 'prof'] =
        rnd(row['Profit']) e['segm'] =
        = row['Segment'] yield e

def rnd(v):
    return str(round(float(v),2))

nếu __name__ == "__main__":
    f= 'data/superstore.csv'
    d = read_dict(f)
    data = to_dict(d)
    jsonf = 'data/superstore.json'
    dump_json(jsonf, data)
    print (Dữ liệu "siêu cửa hàng" được thêm vào
    JSON\n') json_data =
    read_json(jsonf) print ("{:20s} {:15s} {:10s} {:3s} {:5s}
    {:12s} {:10s} ". format('CUSTOMER', 'ITEM', ' BÁN
    HÀNG', 'Q', 'ĐĨA',
    'LỢI NHUẬN', 'ĐOẠN')) trình tạo
    = mk_data(json_data) cho i, hàng trong
    liệt kê (trình tạo): nếu tôi < 10:
        in ("{:20s} {:15s} ".format(row['cust'],
        row['item']),

```

```

    "{:10s} {:3s}".format(row['sale'],
    row['quan']),
    "{:5s} {:12s}".format(row['disc'],
    row['prof']),
    "{:10s}".format(row['segm']))

```

khác:

phá vỡ

Đầu ra:

```

"superstore" data added to JSON

CUSTOMER      ITEM        SALES     Q   DISC   PROFIT    SEGMENT
Claire Gute   Bookcases  261.96   2   0       41.91    Consumer
Claire Gute   Chairs     731.94   3   0       219.58   Consumer
Darrin Van Huff Labels    14.62    2   0       6.87    Corporate
Sean O'Donnell Tables    957.58   5   0.45   -383.03  Consumer
Sean O'Donnell Storage   22.37    2   0.2    2.52    Consumer
Brosina Hoffman Furnishings 48.86    7   0       14.17   Consumer
Brosina Hoffman Art       7.28     4   0       1.97    Consumer
Brosina Hoffman Phones   907.15   6   0.2    90.72   Consumer
Brosina Hoffman Binders  18.5     3   0.2    5.78    Consumer
Brosina Hoffman Appliances 114.9    5   0       34.47   Consumer

```

Ví dụ về mã bắt đầu bằng cách nhập các thư viện csv và json. Hàm `read_dict()` đọc tệp CSV dưới dạng `OrderedDict`. Hàm `to_dict()` chuyển đổi một `OrderedDict` thành một từ điển thông thường. Hàm `dump_json()` lưu tệp vào JSON. Hàm `read_json()` đọc tệp JSON. Hàm `mk_data()` tạo một đối tượng trình tạo bao gồm dữ liệu được sắp xếp lại từ tệp JSON.

Hàm `rnd()` làm tròn một số đến 2 chữ số thập phân. Khối chính bắt đầu bằng cách đọc tệp CSV và chuyển đổi nó thành JSON. Mã tiếp tục bằng cách đọc dữ liệu JSON mới được tạo. Tiếp theo, một đối tượng trình tạo được tạo từ dữ liệu JSON. Đối tượng trình tạo rất quan trọng vì nó tăng tốc độ xử lý các đơn đặt hàng có độ lớn nhanh hơn một danh sách. Vì tập dữ liệu gần 10.000 bản ghi nên tốc độ rất quan trọng. Để xác minh rằng dữ liệu đã được tạo chính xác, đối tượng trình tạo được lặp lại một vài lần để in một số bản ghi bị xáo trộn.

## Chương 5 Làm việc với dữ liệu

Ví dụ mã thứ 5 và cũng là mã cuối cùng đọc tệp JSON được tạo trong ví dụ trước, sắp xếp lại nó và lưu tập dữ liệu đã sắp xếp thành JSON: nhập json

```
def read_json(f):
    với open(f) là f:
        return json.load(f)

def mk_data(d):
    for i, row in enumerate(d):
        e = {}
        e['_id'] =
        ie['cust'] = row['Customer Name']
        e['item'] = row[ 'Chuyên mục
        con' ] e['Giảm giá'] =
        rnd(row['Hàng bán']) e['quan'] =
        = row['Số lượng'] e['Đĩa'] =
        Row['Giảm giá'] e[ 'prof' ] =
        rnd(row['Profit']) e['segm'] =
        = row['Segment'] yield e

def rnd(v):
    return str(round(float(v),2))

if __name__ == "__main__":
    jsonf = 'data/superstore.json'
    json_data = read_json(jsonf)
    l = len(list(mk_data(json_data)))
    generator = mk_data(json_data)
    jsonf= 'data/wrangled.json'
    với open(jsonf, 'w') là f:
        f.write('[')
        for i, row in enumerate(generator): j
            = json.dumps(row)
```

```

nếu tôi < l - 1:
    với open(jsonf, 'a') là f:
        f.write(j)
        f.write(',')
khác:
    với open(jsonf, 'a') là f:
        f.write(j)
        f.write(']')
json_data = read_json(jsonf)
đối với tôi, hàng trong liệt kê (json_data):
    nếu tôi < 5:
        in (hang ['khách hàng'], hàng ['mặt hàng'], hàng ['giảm giá'])
    khác:
        phá vỡ

```

Đầu ra:

```

Claire Gute Bookcases 261.96
Claire Gute Chairs 731.94
Darrin Van Huff Labels 14.62
Sean O'Donnell Tables 957.58
Sean O'Donnell Storage 22.37

```

Ví dụ mã nhập json. Hàm `read_json()` đọc tệp JSON. Hàm `mk_data()` tạo một đối tượng trình tạo bao gồm dữ liệu được sắp xếp lại từ tệp JSON. Hàm `rnd()` làm tròn một số đến hai chữ số thập phân. Khối chính bắt đầu bằng cách đọc tệp JSON. Một đối tượng trình tạo phải được tạo hai lần. Trình tạo đầu tiên cho phép chúng tôi tìm độ dài của tệp JSON. Trình tạo thứ 2 bao gồm dữ liệu được sắp xếp từ tệp JSON. Tiếp theo, trình tạo được duyệt qua để chúng tôi có thể tạo tệp JSON của dữ liệu được sắp xếp. Mặc dù đối tượng trình tạo được tạo và có thể được duyệt qua rất nhanh, nhưng sẽ mất một chút thời gian để tạo một tệp JSON bao gồm gần 10.000 bản ghi được sắp xếp. Trên máy của tôi, mất hơn 33 giây một chút, vì vậy hãy kiên nhẫn.

## CHƯƠNG 6

# khám phá dữ liệu

Khám phá thăm dò sâu hơn vào lĩnh vực dữ liệu. Một chủ đề quan trọng trong khoa học dữ liệu là giảm kích thước. Chương này muốn dữ liệu trộn lẫn từ Chương 5 để chứng minh cách thức hoạt động của nó. Một chủ đề khác là mô phỏng tốc độ. Khi làm việc với các tập dữ liệu lớn, tốc độ có tầm quan trọng rất lớn. Dữ liệu lớn được khám phá với một bộ dữ liệu phổ biến được sử dụng bởi các học giả và ngành công nghiệp. Cuối cùng, Twitter và Web scraping là hai nguồn dữ liệu quan trọng để khám phá.

## Bản đồ nhiệt

Bản đồ nhiệt đã được giới thiệu trong Chương 5, nhưng một bản đồ không được tạo cho bộ dữ liệu trộn lẫn. Vì vậy, chúng tôi bắt đầu bằng cách tạo trực quan hóa Bản đồ nhiệt của dữ liệu wrangled.json.

```

nhập json, gấu trúc dưới dạng pd
nhập matplotlib.pyplot dưới dạng plt
nhập seaborn dưới dạng sns

xác định read_json (f):
    với open(f) là f:
        trả về json.load(f)

def verify_keys(d, **kwargs):
    dữ liệu = d[0].items()
    k1 = set([tup[0] cho tup trong dữ liệu])

```

## Chương 6 Khám phá dữ liệu

```

s = kwargs.items()
k2 = set([tup[1] for tup in s])
danh sách trả về(k1.intersection(k2))

def build_ls(k, d):
    return [{k: row[k] for k in (keys)} for row in d]

def get_rows(d, n):
    [in(hàng) cho i, hàng trong liệt kê(d) nếu tôi < n]

def conv_float(d):
    trả về [dict([k, float(v)]) cho k, v trong hàng.items()] cho hàng
    trong d

if __name__ == "__main__":
    f= 'data/wrangle.json'
    data = read_json(f)
    keys = verify_keys(data, c1='sale', c2='quan', c3='disc', c4='
    prof')
    heat = build_ls(keys, data)
    print ('Hàng đầu tiên trong
    "heat":')
    get_rows(heat, 1)
    heat =
    conv_float(heat)
    print ('\nhàng đầu tiên trong "heat"
    được chuyển đổi
    thành float: ')
    get_rows(heat,
    1)
    df = pd.DataFrame(heat)
    plt.figure()
    sns.heatmap(df.corr(), annot=True, cmap='OrRd')
    plt.show()

Đầu ra:

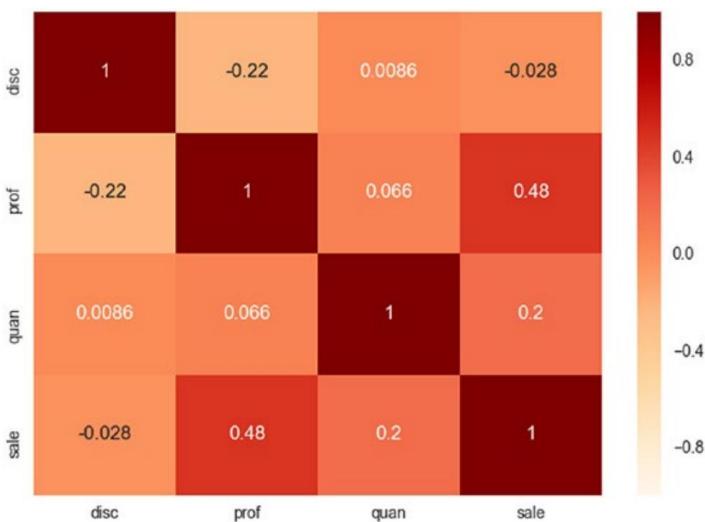
```

```

lst row in "heat":
{'prof': '41.91', 'disc': '0', 'quan': '2', 'sale': '261.96'}

lst row in "heat" converted to float:
{'prof': 41.91, 'disc': 0.0, 'quan': 2.0, 'sale': 261.96}

```



Hình 6-1. bản đồ nhiệt

Ví dụ mã bắt đầu bằng cách nhập các thư viện json, pandas, matplotlib và seaborn. Hàm `read_json()` đọc tệp JSON. Hàm `verify_keys()` đảm bảo rằng các khóa quan tâm tồn tại trong tệp JSON. Điều này rất quan trọng vì chúng tôi chỉ có thể tạo Bản đồ nhiệt dựa trên các biến số và các ứng cử viên duy nhất từ tệp JSON là doanh số, số lượng, chiết khấu và lợi nhuận. Hàm `build_ls()` xây dựng danh sách các phần tử từ điển dựa trên các biến số. Hàm `get_rows()` trả về n hàng từ một danh sách. Hàm `conv_float()` chuyển đổi các phần tử từ điển thành float.

Khối chính bắt đầu bằng cách đọc tệp JSON `wangled.json`. Nó tiếp tục bằng cách lấy khóa chỉ cho các biến số. Tiếp theo, nó xây dựng list một danh sách các phần tử từ điển (nhiệt) dựa trên các khóa thích hợp. Mã này hiển thị hàng đầu tiên dưới dạng nhiệt để minh chứng rằng tất cả các giá trị đều nổi. Vì chúng không phải như vậy, nên mã chuyển đổi chúng thành float. Đoạn mã sau đó tạo một df từ nhiệt và vẽ bản đồ Nhiệt (Hình 6-1).

## Chương 6 Khám phá dữ liệu

## Phân tích thành phần chính

Phân tích thành phần chính (PCA) tìm các thành phần chính của dữ liệu. Các thành phần chính đại diện cho cấu trúc cơ bản trong dữ liệu vì chúng phát hiện ra các hướng mà dữ liệu có nhiều phương sai nhất (được dàn trải nhiều nhất). PCA tận dụng các vectơ riêng và giá trị riêng để khám phá phương sai dữ liệu. Véc tơ riêng là một hướng, trong khi giá trị riêng là một số biểu thị phương sai (trong dữ liệu) theo hướng của véc tơ riêng. Vectơ riêng có giá trị riêng cao nhất là thành phần chính. Một bộ dữ liệu có thể được giải cấu trúc thành các vectơ riêng và giá trị riêng. Số lượng vectơ riêng (và giá trị riêng) trong tập dữ liệu bằng với số thứ nguyên. Vì tập dữ liệu wrangled.json có bốn thứ nguyên (biến), nên nó có bốn vectơ riêng/giá trị riêng.

Ví dụ mã đầu tiên chạy PCA trên tập dữ liệu wrangled.json. Tuy nhiên, PCA chỉ hoạt động với dữ liệu số, vì vậy bộ dữ liệu chỉ được chắt lọc thành những tính năng đó.

```

nhập matplotlib.pyplot dưới dạng plt, pandas dưới dạng pd
nhập numpy dưới dạng np, json, ngẫu nhiên dưới dạng rnd
từ sklearn.tiền xử lý nhập StandardScaler
từ pandas.plotting nhập song song_coords

xác định read_json (f):
    với open(f) là f:
        trả về json.load(f)

def unique_features(k, d):
    danh sách trả về (set([dic[k] for dic in d]))

def sire_features(k, d):
    return [{k: row[k] for k in (k)} for row in d]

def sire_numeric(k, d):
    s = conv_float(sire_features(k, d))
    trả lại s

```

```

def sire_sample(k, v, d, m):
    indices = np.arange(0, len(d), 1) s =
    [d[i] for i trong indices if d[i][k] == v] n =
    len(s)
    num_keys = ['sale', 'quan', 'disc', 'prof'] for
    i, row in enumerate(s): for k
        in num_keys: row[k]
        = float(row[k]) s =
    rnd_sample(m, len(s), s) return
    (s, n)

def rnd_sample(m, n, d):
    indices = sorted(rnd.sample(range(n), m))
    return [d[i] for i in index]

def conv_float(d):
    trả về [dict([k, float(v)]) cho k, v trong hàng.items()] cho hàng
    trong d]

nếu __name__ == "__main__":
    f = 'data/wrangle.json'
    data = read_json(f)
    segm = unique_features('segm', data)
    print ('các lớp trong tính năng "segm":')
    print (segm)
    keys = ['sale', 'quan ', 'disc', 'prof', 'segm']
    features = sire_features(keys, data)
    num_keys = ['sale', 'quan', 'disc', 'prof']
    numeric_data = sire_numeric(num_keys, features) k , v
    = "segm", "Văn phòng chính"
    m = 100

    s_home = sire_sample(k, v, features, m) v =
    "Người tiêu dùng"
    s_cons = sire_sample(k, v, features, m)

```

## Chương 6 Khám phá dữ liệu

```

v = "Doanh
nghiệp" s_corp = sire_sample(k, v,
features, m) print ('\nLát Office tại nhà:', s_home[1]) print('Lát người tiêu
dùng:', s_cons[1]) print ('Lát cắt
doanh nghiệp :', s_corp[1])
print ('kích thước mẫu:', m)
df_home = pd.DataFrame(s_home[0])
df_cons = pd.DataFrame(s_cons[0])
df_corp = pd.DataFrame(s_corp[0] )
frames = [df_home, df_cons,
df_corp]
result = pd.concat(frames) plt.figure()
parallel_coordin(result, 'segm', color= ['orange', 'red'])
df = pd.DataFrame(numeric_data)
X = df.ix[:,].values
X_std = StandardScaler().fit_transform(X)
mean_vec = np.mean(X_std, axis=0)
cov_mat = np.cov(X_std.T)
print ('\nma trận hiệp phương sai:\n',
cov_mat) eig_vals, eig_vecs =
np.linalg.eig(cov_mat) print
('Eigenvalue:\n', eig_vecs) print ('\nEigenvalues:
\n', np.sort(eig_vals)
[::-1]) tot = sum(eig_vals) var_exp = [(i / tot)*100
for i in
sorted(eig_vals, reverse=True)] print
('phương sai giải thích:
\n', var_exp) corr_mat = np.corrcoef(XT )
print ('\ncorrelation matrix:\n', corr_mat)
eig_vals, eig_vecs =
np.linalg.eig(corr_mat) print ('\nEigenvalue:\n', eig_vecs) print ('\nEigen

```

```

tot = sum(eig_vals)
var_exp = [(i / tot)*100 for i in sorted(eig_vals,
reverse=True)]
print ('\nvi tri duoc giải thích:\n', var_exp)
cum_var_exp = np.cumsum(var_exp)
fig , ax = plt.subplots()
label = ['PC1', 'PC2', 'PC3', 'PC4']
width = 0,35
index = np.arange(len(var_exp))
ax.bar(index, var_exp,
       color=['fuchsia', 'lime', 'thistle', 'thistle'])
for i, v in enumerate(var_exp):
    v = round(v, 2)
    val = str(v) + '%'
    ax. text(i, v+0.5, val, ha='center', color='b',
              fontsize=9, fontweight='bold')
plt.xticks(index, label)
plt.title('Giải thích phương sai') plt .trình diễn()

```

Đầu ra:

```

classes in "segm" feature:
['Home Office', 'Consumer', 'Corporate']

Home Office slice: 1783
Consumer slice: 5191
Coporate slice: 3020
sample size: 100

```

## Chương 6 Khám phá dữ liệu

```
covariance matrix:
[[ 1.00010007 -0.21950937  0.00862383 -0.02819299]
 [-0.21950937  1.00010007  0.06625978  0.47911246]
 [ 0.00862383  0.06625978  1.00010007  0.20081494]
 [-0.02819299  0.47911246  0.20081494  1.00010007]]

Eigenvectors:
[[ -0.27037624  0.24517839  0.71856986  0.59198108]
 [ 0.65545599  0.68416982 -0.19540197  0.25319396]
 [ 0.28863648  0.16325021  0.63082196 -0.70149983]
 [ 0.64339966 -0.66719456  0.21803458  0.30553103]]

Eigenvalues:
[ 1.59012581  1.05880782  0.88144442  0.47002223]

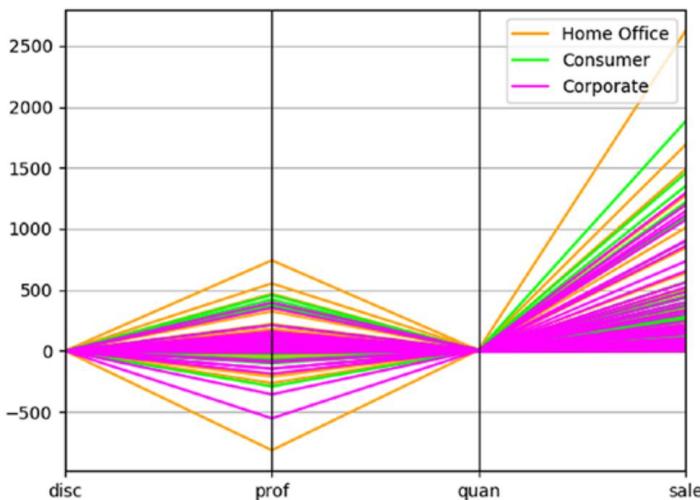
variance explained:
[39.749167611521841, 26.467546859432439, 22.033905616069589, 11.749379912976137]

correlation matrix:
[[ 1.          -0.21948741  0.00862297 -0.02819017]
 [-0.21948741  1.          0.06625315  0.47906452]
 [ 0.00862297  0.06625315  1.          0.20079484]
 [-0.02819017  0.47906452  0.20079484  1.        ]]

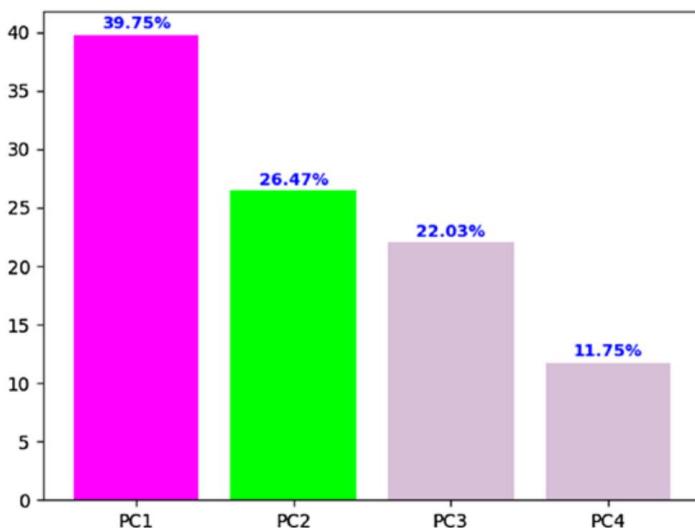
Eigenvectors:
[[ -0.27037624  0.24517839  0.71856986  0.59198108]
 [ 0.65545599  0.68416982 -0.19540197  0.25319396]
 [ 0.28863648  0.16325021  0.63082196 -0.70149983]
 [ 0.64339966 -0.66719456  0.21803458  0.30553103]]

Eigenvalues:
[ 1.5899667  1.05870187  0.88135622  0.4699752 ]

variance explained:
[39.749167611521855, 26.467546859432396, 22.033905616069603, 11.749379912976144]
```



Hình 6-2. tọa độ song song



Hình 6-3. phương sai giải thích

Ví dụ về mã bắt đầu bằng cách nhập các thư viện matplotlib, pandas, numpy, json, random và sklearn. Hàm `read_json()` đọc tệp JSON. Hàm `unique_features()` chắt lọc các danh mục (lớp) duy nhất từ một thứ nguyên (tính năng). Trong trường hợp này, nó chắt lọc ba lớp-Văn phòng tại nhà, Doanh nghiệp và Người tiêu dùng-từ tính năng phân đoạn. Vì tập dữ liệu có gần 10.000 bản ghi nên tôi muốn chắc chắn có những lớp nào trong đó. Chức năng `sire_features()` chắt lọc một tập dữ liệu mới chỉ với các tính năng quan tâm. Hàm `sire_numeric()` chuyển chuỗi số thành float. Hàm `sire_sample()` trả về một mẫu ngẫu nhiên gồm n bản ghi được lọc cho một lớp. Chức năng `sample()` tạo một mẫu ngẫu nhiên. Hàm `convert_float()` chuyển đổi dữ liệu chuỗi số thành float.

Khối chính bắt đầu bằng cách đọc `wangled.json` và tạo các tính năng tập dữ liệu chỉ với các tính năng quan tâm. Mã tiếp tục bằng cách tạo tập dữ liệu số chỉ bao gồm các tính năng có dữ liệu số. Số tập dữ liệu được sử dụng để tạo PCA. Tiếp theo, ba mẫu có kích thước 100 được tạo ra; một cho mỗi lớp. Các mẫu được sử dụng để tạo ra các

## Chương 6 Khám phá dữ liệu

trực quan hóa tọa độ song song (Hình 6-2). Mã cho PCA theo sau bằng cách chuẩn hóa và chuyển đổi tập dữ liệu số. Một ma trận hiệp phương sai được tạo để có thể tạo ra các vectơ riêng và giá trị riêng. Tôi bao gồm PCA bằng cách sử dụng ma trận tương quan vì một số ngành thích nó hơn. Cuối cùng, một hình ảnh trực quan của các thành phần chính được tạo ra.

Các tọa độ song song cho thấy prof (lợi nhuận) và sale (bán hàng) là những tính năng quan trọng nhất. Hình ảnh PCA (Hình 6-3) cho thấy thành phần chính thứ 1 chiếm 39,75%, thứ 2 26,47%, thứ 3 22,03% và thứ 4 11,75%. Phân tích PCA không hữu ích lắm trong trường hợp này, vì cả bốn thành phần chính đều cần thiết, đặc biệt là ba thành phần đầu tiên. Vì vậy, chúng tôi không thể loại bỏ bất kỳ thứ nguyên nào khỏi phân tích trong tương lai.

Ví dụ mã thứ 2 sử dụng bộ dữ liệu iris cho PCA:

```
nhập matplotlib.pyplot dưới dạng plt, pandas dưới dạng pd, numpy dưới dạng np
từ sklearn.preprocessing import StandardScaler
từ pandas.plotting nhập song(song_coords)

def conv_float(d):
    trả lại d.astype (float)

nếu __name__ == "__main__":
    df = pd.read_csv('data/iris.csv')
    X = df.ix[:,0:4].values
    y = df.ix[:,4].values
    X_std = StandardScaler().fit_transform(X)
    mean_vec = np.mean(X_std, axis=0)
    cov_mat = np.cov(X_std.T)
    eig_vals, eig_vecs = np.linalg.eig(cov_mat)
    print ('Vectơ riêng:\n', eig_vecs)
    in ('\nGiá trị bản địa:\n', eig_vals)
    plt.figure()
    song(song_tọa độ (df, 'Tên', màu =
                    ['cam', 'chanh', 'fuchsia']))
```

```

tot = sum(eig_vals)
var_exp = [(i / tot)*100 for i in sorted(eig_vals,
reverse=True)]
cum_var_exp = np.cumsum(var_exp)
fig, ax = plt.subplots()
label = ['PC1 ', 'PC2', 'PC3', 'PC4']
width = 0,35
index = np.arange(len(var_exp))
ax.bar(index, var_exp,
       color=['fuchsia', 'lime', 'thistle', 'thistle'])
for i, v in enumerate(var_exp):
    v = round(v, 2)
    val = str(v) + '%'
    ax.text(i, v+0.5, val, ha='center', color='b',
            fontsize=9, fontweight='bold')
plt.xticks(index, label)
plt.title('Giải thích phương
sai') plt .trình diẽn()

```

Đầu ra:

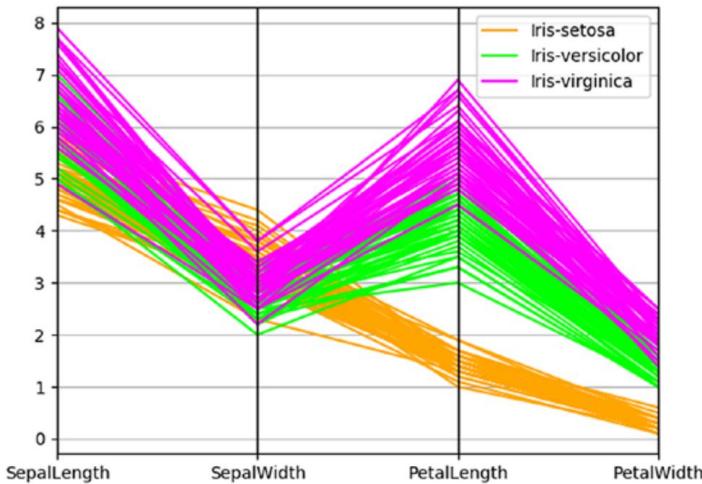
```

Eigenvectors:
[[ 0.52237162 -0.37231836 -0.72101681  0.26199559]
 [-0.26335492 -0.92555649  0.24203288 -0.12413481]
 [ 0.58125401 -0.02109478  0.14089226 -0.80115427]
 [ 0.56561105 -0.06541577  0.6338014   0.52354627]]

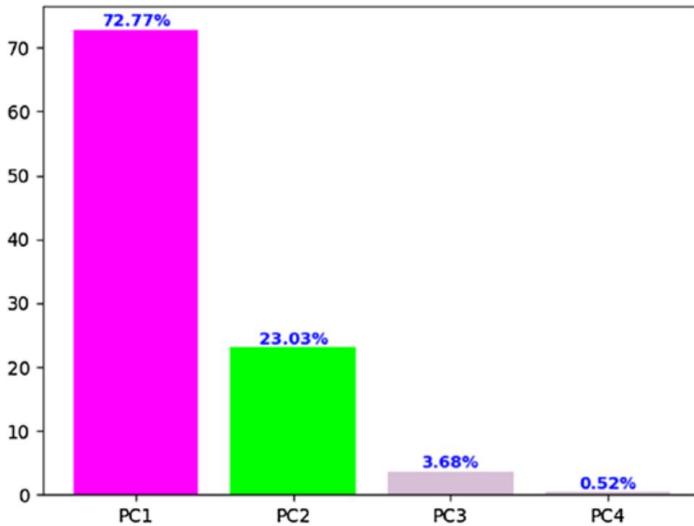
Eigenvalues:
[ 2.93035378  0.92740362  0.14834223  0.02074601]

```

## Chương 6 Khám phá dữ liệu



Hình 6-4. tọa độ song song



Hình 6-5. phương sai giải thích

## Chương 6 Khám phá dữ liệu

Ví dụ về mã ngắn hơn nhiều so với ví dụ trước, vì chúng tôi không phải sắp xếp, dọn dẹp (nhiều) và tạo các mẫu ngẫu nhiên (để trực quan hóa Tạo độ song song). Mã bắt đầu bằng cách nhập các thư viện matplotlib, pandas, numpy và sklearn. Hàm conv\_float() chuyển đổi các chuỗi số thành float. Khối chính bắt đầu bằng cách đọc bộ dữ liệu mồng mắt. Nó tiếp tục bằng cách chuẩn hóa và chuyển đổi dữ liệu cho PCA. Tạo độ song song và giải thích phương sai sau đó được hiển thị.

Tạo độ song song cho thấy PetalLength và PetalWidth là các tính năng quan trọng nhất (Hình 6-4). Hình ảnh PCA (Giải thích phương sai) cho thấy thành phần chính thứ nhất chiếm 72,77%, thứ 2 23,03%, thứ 3 3,68% và thứ 4 0,52% (Hình 6-5). Phân tích PCA rất hữu ích trong trường hợp này vì hai thành phần chính đầu tiên chiếm hơn 95% phương sai. Vì vậy, chúng ta có thể loại bỏ PC3 và PC4 khỏi việc xem xét thêm.

Để rõ ràng, bước đầu tiên cho PCA là khám phá các vectơ riêng và giá trị riêng. Các vectơ riêng có giá trị riêng thấp nhất mang ít thông tin nhất về phân phối dữ liệu, vì vậy chúng có thể bị loại bỏ. Trong ví dụ này, hai giá trị riêng đầu tiên cao hơn nhiều, đặc biệt là PC1. Do đó, việc loại bỏ PC3 và PC4 là theo thứ tự. Bước thứ 2 là đo lường phương sai được giải thích, phương sai này có thể được tính toán từ các giá trị riêng. Phương sai được giải thích cho chúng ta biết có bao nhiêu thông tin (phương sai) có thể được quy cho mỗi thành phần chính. Nhìn vào phương sai được giải thích xác nhận rằng PC3 và PC4 không quan trọng.

## Mô phỏng tốc độ

Tốc độ trong khoa học dữ liệu rất quan trọng, đặc biệt là khi bộ dữ liệu trở nên lớn hơn. Trình tạo rất hữu ích trong việc tối ưu hóa bộ nhớ, bởi vì hàm trình tạo trả về một mục tại một thời điểm (khi cần) thay vì tất cả các mục cùng một lúc.

## Chương 6 Khám phá dữ liệu

Ví dụ mã tương phản tốc độ giữa danh sách và trình tạo:

```

nhập json, thân thiện với con người dưới
dạng hf từ dòng hò nhập thời gian

def read_json(f):
    với open(f) là f:
        return json.load(f)

def mk_gen(k, d):
    cho hàng trong d:
        dic = {}
        cho khóa trong
            k: dic[key] = float(row[key])
        mang lại dic

def conv_float(keys, d):
    return [dict([k, float(v)] for k, v in row.items() if k in
                keys) for row in d]

nếu __name__ == "__main__":
    f = 'data/wrangle.json'
    data = read_json(f)
    keys = ['sale', 'quan', 'disc', 'prof'] print
    ('tạo, chuyển đổi và hiển thị danh sách:') start =
    clock () data =
    conv_float(keys, data) for i, row
    in enumerate(data): if i < 5: print
        (row) end
        = clock()
    elapsed_ls =
    end - start print
    (hf.format_timespan(elapsed_ls, detail=True )) print ('\ntạo,
    chuyển đổi và hiển thị trình tạo:')

```

## Chương 6 Khám phá dữ liệu

```
bắt đầu = đồng hồ()
trình tạo = mk_gen(phím, dữ liệu)
đối với tôi, hàng trong liệt kê (máy phát điện):
nếu tôi < 5:
    in (hang)

kết thúc = đồng hồ()
elapsed_gen = kết thúc - bắt đầu
in (hf.format_timespan(elapsed_gen, chi tiết=True))
tốc độ = vòng(elapsed_ls / elapsed_gen, 2)
print ('\nngenerator là', tốc độ, 'nhanh gấp nhiều lần')
```

Đầu ra:

```
create, convert, and display list:
{'sale': 261.96, 'quan': 2.0, 'disc': 0.0, 'prof': 41.91}
{'sale': 731.94, 'quan': 3.0, 'disc': 0.0, 'prof': 219.58}
{'sale': 14.62, 'quan': 2.0, 'disc': 0.0, 'prof': 6.87}
{'sale': 957.58, 'quan': 5.0, 'disc': 0.45, 'prof': -383.03}
{'sale': 22.37, 'quan': 2.0, 'disc': 0.2, 'prof': 2.52}
46.03 milliseconds

create, convert, and display generator:
{'sale': 261.96, 'quan': 2.0, 'disc': 0.0, 'prof': 41.91}
{'sale': 731.94, 'quan': 3.0, 'disc': 0.0, 'prof': 219.58}
{'sale': 14.62, 'quan': 2.0, 'disc': 0.0, 'prof': 6.87}
{'sale': 957.58, 'quan': 5.0, 'disc': 0.45, 'prof': -383.03}
{'sale': 22.37, 'quan': 2.0, 'disc': 0.2, 'prof': 2.52}
20.38 milliseconds

generator is 2.26 times faster
```

Ví dụ mã bắt đầu bằng cách nhập các thư viện json, thân thiện với con người và thời gian. Bạn có thể phải cài đặt thân thiện với con người như tôi đã làm: pip cài đặt thân thiện với con người. Hàm read\_json() đọc JSON. Hàm mk\_gen() tạo trình tạo dựa trên bốn tính năng từ wrangled.json và chuyển đổi giá trị thành float. Hàm conv\_float() chuyển đổi các giá trị từ điển từ danh sách thành float. Khỏi chính bắt đầu bằng cách đọc lộn xộn.

json vào một danh sách. Mã tiếp tục bằng cách định thời gian cho quá trình tạo danh sách mới từ khóa và chuyển đổi giá trị thành float. Tiếp theo, một trình tạo được tạo bắt chước quá trình tạo và chuyển đổi danh sách. Trình tạo nhanh hơn 2,26 lần (trên máy tính của tôi).

## Chương 6 Khám phá dữ liệu

### Dữ liệu lớn

Dữ liệu lớn là cơn thịnh nộ của thế kỷ 21. Vì vậy, hãy làm việc với một tập dữ liệu tương đối lớn. GroupLens là một trang web cung cấp quyền truy cập vào bộ dữ liệu điện toán xã hội lớn cho lý thuyết và thực hành. GroupLens đã thu thập và cung cấp bộ dữ liệu xếp hạng từ trang web MovieLens:

<https://grouplens.org/datasets/movielens/>. Chúng ta sẽ khám phá bộ dữ liệu 1M, chứa khoảng một triệu xếp hạng từ sáu nghìn người dùng trên bốn nghìn phim. Tôi đã do dự để sắp xếp, dọn dẹp và xử lý tập dữ liệu hơn một triệu do khả năng xử lý hạn chế của chiếc PC tương đối mới của tôi.

Ví dụ mã đầu tiên đọc, dọn dẹp, định cỡ và kết xuất dữ liệu MovieLens sang JSON:

nhập json, csv

```
xác định read_dat(h, f):
    trả về csv.DictReader((line.replace('::', ':')
                            cho dòng mở (f)),
                            dấu phân cách = ':', tên trường = h,
                            trích dẫn=csv.QUOTE_NONE)

def gen_dict(d):
    cho hàng trong d:
        năng suất chính tả (hàng)

def dump_json(f, l, d):
    f = mở(f, 'w')
    f.write('[')
    đối với tôi, hàng trong liệt kê (d):
        j = json.dumps( hàng)
        f.write(j)
        nếu tôi < l - 1:
            f.write(',')
```

khác:

```
f.write(']')  
f.close()  
  
def read_json(f):  
    với open(f) là f:  
        return json.load(f)  
  
def display(n, f):  
    cho i, hàng trong liệt kê(f):  
        if i < n:  
            print (row)  
    print()  
  
if __name__ == "__main__":  
    print ('... định rõ dữ liệu...\n')  
    u_dat = 'data/ml-1m/users.dat'  
    m_dat = 'data/ml-1m/phim.dat'  
    r_dat = 'data/ml-1m/ratings.dat'  
    unames = ['user_id', 'giới tính', 'tuổi', 'nghề nghiệp', 'zip']  
    mnames = ['movie_id', 'title', 'thể loại']  
    rnames = ['user_id', 'movie_id', 'rating', 'timestamp']  
    users = read_dat(unames, u_dat)  
    ul = len(list(gen_dict(users)))  
    phim = read_dat(mnames, m_dat) ml  
    = len (list(gen_dict(phim)))  
    ratings = read_dat(rnames, r_dat)  
    rl = len(list(gen_dict(ratings)))  
    print ('kích thước của bộ dữ  
liệu:') print  
    ('users', ul) print  
    ('phim ', ml) print  
    ('xếp hạng', rl) print ('\n... kết xuất dữ liệu ...')  
    print
```

## Chương 6 Khám phá dữ liệu

```

người dùng = read_dat(unames,
u_dat) người dùng =
gen_dict(users) phim =
read_dat(mnames, m_dat) phim
= gen_dict(phim) xếp hạng =
read_dat(rnames, r_dat) xếp
hạng = gen_dict(xếp
hạng) uf = 'data/users.json
' dump_json(uf, ul, users)
mf = 'data/movies.json'
dump_json(mf, ml, movies)
rf = 'data/ratings.json'
dump_json(rf, rl, ratings) print
('`n.. .đang xác
minh dữ liệu..`n')
u = read_json(uf) m = read_json(mf) r = read_json(rf)
n = 1
hiển thị(n, u)
hiển thị(n, m)
hiển thị(n, r)

```

Đầu ra:

```

... sizing data ...

size of datasets:
users 6040
movies 3883
ratings 1000209

... dumping data ...

... verifying data ...

{'user_id': '1', 'gender': 'F', 'age': '1', 'occupation': '10', 'zip': '48067'}
{'movie_id': '1', 'title': 'Toy Story (1995)', 'genres': "Animation|Children's|Comedy"}
{'user_id': '1', 'movie_id': '1193', 'rating': '5', 'timestamp': '978300760'}

```

## Chương 6 Khám phá dữ liệu

Ví dụ mã bắt đầu bằng cách nhập các thư viện json và csv. Chức năng `read_dat()` đọc và xóa dữ liệu (thay thế dấu hai chấm bằng dấu hai chấm đơn làm dấu phân cách). Hàm `gen_dict()` chuyển đổi danh sách `OrderedDict` thành danh sách từ điển thông thường để xử lý dễ dàng hơn. Hàm `dump_json()` là một hàm tùy chỉnh mà tôi đã viết để kết xuất dữ liệu sang JSON. Hàm `read_json()` đọc JSON. Hàm `display()` hiển thị một số dữ liệu để xác minh. Khối chính bắt đầu bằng cách đọc ba bộ dữ liệu và tìm kích thước của chúng. Nó tiếp tục bằng cách đọc lại các bộ dữ liệu và chuyển sang JSON. Các bộ dữ liệu cần phải được đọc lại vì trình tạo chỉ có thể được duyệt qua một lần. Vì bộ dữ liệu xếp hạng có hơn một triệu bản ghi nên sẽ mất vài giây để xử lý.

Ví dụ mã thứ 2 làm sạch tập dữ liệu phim, yêu cầu làm sạch bổ sung rỗng rãi:

```
nhập json, numpy dưới dạng np
xác định read_json (f):
    với open(f) là f:
        trả về json.load(f)

def dump_json(f, d):
    với open(f, 'w') là lối:
        json.dump(d, fout)

hiển thị def(n, d):
    [in (hàng) cho i, hàng trong liệt kê (d) nếu tôi < n]

def get_indx(k, d):
    trả về [hàng[k] cho hàng trong d nếu 'null' trong hàng]

def get_data(k, l, d):
    trả về [hàng cho i, hàng trong liệt kê(d) nếu hàng[k] trong l]

def get_unique(key, d):
    s = bô()
    cho hàng trong d:
```

## Chương 6 Khám phá dữ liệu

```

    cho k, v trong hàng.items():
        nếu k trong
            khóa:
                s.add(v) return np.sort(list(s))

nếu __name__ == "__main__":
    mf = 'data/movies.json' m =
    read_json(mf)
    n = 20

    display(n, m)
    print()

    indx = get_indx('movie_id', m) cho
    hàng trong m:
        nếu hàng ['movie_id'] trong indx:
            hàng['tiêu đề'] = hàng['tiêu đề'] + ':' + hàng['thể loại']
            hàng['thể loại'] = hàng['null'][0]
            del hàng['null']

            tiêu đề = hàng ['title'].split(" ")
            year = title.pop()
            year = ''.join(c cho c trong năm nếu c không ở '()')
            row['title'] = ' '.join (tiêu đề)
            row['year'] = year

    data = get_data('movie_id', indx, m)
    n = 2

    display(n, data) s
    = get_unique('year', m) print
    ('\n', s, '\n') rec =
    get_data('year', ['Assignment'], m) print (rec[ 0])

```

```
rec = get_data('year', ["L'Associe1982"], m)
print (rec[0], '\n')
b1, b2, cnt = False, False, 0
cho hàng trong m:

if row['movie_id'] in ['1001']:
    row['year'] = '1982'
    print (row)
    b1 = True

elif row['movie_id'] in ['2382']:
    row['title'] = 'Học viện cảnh sát 5: Nhiệm vụ: Miami
    Beach'

    row['genres'] = 'Hài kịch'
    row['year'] = '1988'
    print (row)
    b2 = True

elif b1 và b2: phá vỡ
    cnt += 1

print ('\n', cnt, len(m))
mf = 'data/cmovies.json'
dump_json(mf, m)
m = read_json(mf)
display(n, m)
```

## Chương 6 Khám phá dữ liệu

Đầu ra:

```
('movie_id': '1', 'title': 'Toy Story (1995)', 'genres': "Animation|Children's|Comedy")
('movie_id': '2', 'title': 'Jumanji (1995)', 'genres': "Adventure|Children's|Fantasy")
('movie_id': '3', 'title': 'Grumpier Old Men (1995)', 'genres': 'Comedy|Romance')
('movie_id': '4', 'title': 'Waiting to Exhale (1995)', 'genres': 'Comedy|Drama')
('movie_id': '5', 'title': 'Father of the Bride Part II (1995)', 'genres': 'Comedy')
('movie_id': '6', 'title': 'Heat (1995)', 'genres': 'Action|Crime|Thriller')
('movie_id': '7', 'title': 'Sabrina (1995)', 'genres': 'Comedy|Romance')
('movie_id': '8', 'title': 'Tom and Huck (1995)', 'genres': "Adventure|Children's")
('movie_id': '9', 'title': 'Sudden Death (1995)', 'genres': 'Action')
('movie_id': '10', 'title': 'GoldenEye (1995)', 'genres': 'Action|Adventure|Thriller')
('movie_id': '11', 'title': 'American President, The (1995)', 'genres': 'Comedy|Drama|Romance')
('movie_id': '12', 'title': 'Dracula', 'genres': 'Dead and Loving It (1995)', 'null': ['Comedy|Horror'])
('movie_id': '13', 'title': 'Balto (1995)', 'genres': "Animation|Children's")
('movie_id': '14', 'title': 'Nixon (1995)', 'genres': 'Drama')
('movie_id': '15', 'title': 'Cutthroat Island (1995)', 'genres': 'Action|Adventure|Romance')
('movie_id': '16', 'title': 'Casino (1995)', 'genres': 'Drama|Thriller')
('movie_id': '17', 'title': 'Sense and Sensibility (1995)', 'genres': 'Drama|Romance')
('movie_id': '18', 'title': 'Four Rooms (1995)', 'genres': 'Thriller')
('movie_id': '19', 'title': 'Ace Ventura', 'genres': 'When Nature Calls (1995)', 'null': ['Comedy'])
('movie_id': '20', 'title': 'Money Train (1995)', 'genres': 'Action')

('movie_id': '12', 'title': 'Dracula: Dead and Loving It', 'genres': 'Comedy|Horror', 'year': '1995')
('movie_id': '19', 'title': 'Ace Ventura: When Nature Calls', 'genres': 'Comedy', 'year': '1995')

[1919' '1920' '1921' '1922' '1923' '1925' '1926' '1927' '1928' '1929'
'1930' '1931' '1932' '1933' '1934' '1935' '1936' '1937' '1938' '1939'
'1940' '1941' '1942' '1943' '1944' '1945' '1946' '1947' '1948' '1949'
'1950' '1951' '1952' '1953' '1954' '1955' '1956' '1957' '1958' '1959'
'1960' '1961' '1962' '1963' '1964' '1965' '1966' '1967' '1968' '1969'
'1970' '1971' '1972' '1973' '1974' '1975' '1976' '1977' '1978' '1979'
'1980' '1981' '1982' '1983' '1984' '1985' '1986' '1987' '1988' '1989'
'1990' '1991' '1992' '1993' '1994' '1995' '1996' '1997' '1998' '1999'
'2000' 'Assignment' 'L'Associe1982"]

('movie_id': '2382', 'title': 'Police Academy 5:', 'genres': 'Miami Beach (1988)', 'year': 'Assignment')
('movie_id': '1001', 'title': 'Associate, The', 'genres': 'Comedy', 'year': 'L'Associe1982')

('movie_id': '1001', 'title': 'Associate, The', 'genres': 'Comedy', 'year': '1982')
('movie_id': '2382', 'title': 'Police Academy 5: Assignment: Miami Beach', 'genres': 'Comedy', 'year': '1988')

2314 3883
('movie_id': '1', 'title': 'Toy Story', 'genres': "Animation|Children's|Comedy", 'year': '1995')
('movie_id': '2', 'title': 'Jumanji', 'genres': "Adventure|Children's|Fantasy", 'year': '1995')
```

Ví dụ mã bắt đầu bằng cách nhập các thư viện json và numpy.

Hàm `read_json()` đọc JSON. Hàm `dump_json()` lưu JSON. Hàm `display()` hiển thị n bản ghi. Hàm `get_idx()` trả về chỉ số của các phần tử từ điển bằng khóa null. Hàm `get_data()` trả về tập dữ liệu được lọc theo chỉ số và khóa `movie_id`. Hàm `get_unique()` trả về danh sách các giá trị duy nhất từ danh sách các thành phần từ điển. Khối chính bắt đầu bằng cách đọc `movies.json` và hiển thị để kiểm tra. Bản ghi 12 và 19 có khóa rỗng. Mã tiếp tục bằng cách tìm tất cả các chỉ số `movie_id` bằng khóa null. Một số dòng tiếp theo làm sạch tất cả các phim. Những người có khóa null yêu cầu logic bổ sung để xóa hoàn toàn, nhưng tất cả các bản ghi đều có tiêu đề đã sửa đổi và khóa năm mới. Để xác minh, các bản ghi 12 và 19 được hiển thị. Để chắc chắn rằng tất cả đều ổn, đoạn mã tìm tất cả các khóa duy nhất dựa trên năm.

## Chương 6 Khám phá dữ liệu

Lưu ý rằng có hai bản ghi không có năm hợp lệ. Vì vậy, mã làm sạch hai bản ghi. Elif thứ 2 đã được thêm vào mã để ngừng xử lý sau khi hai bản ghi vẫn đã được xóa. Mặc dù không có trong mã, nhưng tôi đã kiểm tra các phím movie\_id, title và thể loại nhưng không tìm thấy vấn đề gì.

Mã để kết nối với MongoDB như sau:

kết nối lớp:

```
từ pymongo nhập MongoClient client =
MongoClient('localhost', port=27017) def __init__(self,
dbname): self.db =
    conn.client[dbname] def getDB(self):
    return self.db
```

Tôi đã tạo thư mục 'các lớp' và lưu mã trong 'conn.py' Ví dụ về mã thứ 3 tạo thông tin hữu ích từ ba bộ dữ liệu:

```
nhập json, numpy dưới dạng np, sys, os, thân thiện với con người
dưới dạng hf từ thời điểm
nhập đồng hồ sys.path.append(os.getcwd()+'/
classes') import conn

def read_json(f):
    với open(f) là f:
        return json.load(f)

def get_column(A, v): trả
    về [A_i[v] cho A_i trong A]

def remove_nr(v1, v2):
    set_v1 = set(v1)
    set_v2 = set(v2)
    diff = list(set_v1 - set_v2)
    return diff
```

## Chương 6 Khám phá dữ liệu

```

def get_info(*args): a =
    [arg for arg in args] ratings =
    [int(row[a[0][1]]) for row in a[2] if row[a[0][0]] == a[1]] uids =
    [row[a[0][3]]]

    for row in a[2] if row[a[0][0]] == a[1] title = [row[ a[0][2]]] cho hàng
    trong a[3] if row[a[0][0]] == a[1] age = [int(row[a[0][4]])] cho col trong
    uid cho hàng trong a[4] if col == row[a[0][3]]] gioi tính = [row[a[0][5]]] cho
    col trong uid cho hàng

    trong người dùng if col == row[a[0][3]]] return (xếp hạng, tiêu đề[0], uid,
    tuổi, giới tính)

def tạo (k, v, r, m, u):
    for i, mid in enumerate(v): dic =
        {} rec =
        get_info(k, mid, r, m, u) dic = {'_id':i,
        'mid':mid, 'title':rec[1 ], 'xếp hạng trung
        bình':np.mean(rec[0]),
        'n_ratings':len(rec[0]), 'avg_age':np.
        mean(rec[3]),
        'M':rec[4].count('M'), 'F':rec[4].count('F')}
        dic['avg_rating'] = round(float (str(dic['avg_rating'])) [:6]),2

        dic['avg_age'] = round(float(str(dic['avg_age'])[:6])) năng suất dic

def gen_ls(g):
    cho i, hàng trong kiểu liệt kê(g):
        hàng lợi nhuận

```

```

if __name__ == "__main__":
    print ('... đang tạo bộ dữ liệu...\n') m
    = 'data/cmovies.json'
    movies = np.array(read_json(m)) r
    = 'data/ratings.json'
    ratings = np.array(read_json(r)) r
    = 'data/users.json'
    users = np.array(read_json(r))
    print ('... tạo dữ liệu vectơ chỉ số phim ... \n') mv =
    get_column(phim, 'movie_id') rv =
    get_column(ratings, 'movie_id') print
    ('... tạo vector chỉ số phim chưa được xếp hạng ... \n') nrw =
    remove_nr(mv, rv) diff =
    [ int(row) for row in nrw] print
    (np.sort(diff), '\n') new_mv
    = [x for x in mv if x not in nrw] mid = '1'

    keys = ('movie_id', 'rating', 'title', 'user_id', 'age',
    'gender')
    stats = get_info(keys, mid, ratings, movies, users)
    avg_rating = np.mean(stats[0])
    avg_age = np.mean(stats[3])
    n_ratings = len(stats[0])
    title = stats[1]
    M, F = stats[4].count('M'), stats[4].count('F')
    print ('xếp hạng trung bình cho:',
    end=' ') print (title + " is" , round(avg_rating, 2),
    end=' () print (n_ratings,
    'ratings)\n') gen = generate(keys, new_mv, ratings,
    movies, users) gls
    = gen_ls(gen) obj = conn.conn ('Bài kiểm tra')

```

## Chương 6 Khám phá dữ liệu

```
db = obj.getDB()
phim_info = db.movie_info
phim_info.drop()
print ('...đang lưu movie_info vào MongoDB...\n')
bắt đầu = đồng hồ()
cho hàng trong gls:
    movie_info.insert(hàng)
kết thúc = đồng hồ()
elapsed_ls = kết thúc - bắt đầu
in (hf.format_timespan(elapsed_ls, chi tiết=True))
```

Đầu ra:

```
... creating datasets ...
... creating movie indicies vector data ...
... creating unrated movie indicies vector ...
[ 51 109 115 143 284 285 395 399 400 403 604 620 625 629 636
 654 675 676 683 693 699 713 721 723 727 738 739 752 768 770
 772 773 777 794 795 797 812 816 819 822 825 845 855 856 857
 871 873 890 894 979 983 1001 1045 1052 1065 1075 1106 1108 1109 1110
1122 1137 1140 1141 1143 1146 1155 1156 1157 1158 1159 1166 1308 1309 1314
1318 1319 1368 1400 1424 1443 1448 1462 1467 1524 1557 1559 1568 1577 1578
1628 1697 1698 1705 1706 1708 1710 1716 1723 1738 1740 1742 1757 1765 1768
1774 1776 1781 1789 1819 1847 2030 2199 2216 2220 2222 2224 2225 2228 2229
2230 2270 2274 2319 2489 2547 2564 2588 2595 2601 2603 2604 2680 2684
2698 2832 2838 2910 2954 2957 2958 2980 3009 3023 3059 3080 3170 3191 3193
3195 3226 3227 3231 3234 3278 3279 3332 3348 3356 3369 3383 3411 3455 3541
3558 3560 3561 3582 3583 3589 3630 3650 3750 3829 3856 3907]

avg rating for: "Toy Story" is 4.15 (2077 ratings)

... saving movie_info to MongoDB ...

31 minutes, 29 seconds and 96.07 milliseconds
```

Ví dụ mã bắt đầu bằng cách nhập json, numpy, sys, os, humanfriendly, time và conn (một lớp tùy chỉnh mà tôi đã tạo để kết nối với MongoDB). Hàm `read_json()` đọc JSON. Hàm `get_column()` trả về một vectơ cột. Hàm `remove_nr()` loại bỏ các giá trị `movie_id` không được xếp hạng. Hàm `get_info()` trả về xếp hạng, người dùng, tuổi và giới tính dưới dạng vectơ cột cũng như tiêu đề của phim. Hàm này rất phức tạp vì mỗi vectơ được tạo bằng cách duyệt qua một trong các tập dữ liệu

## Chương 6 Khám phá dữ liệu

và làm phép so sánh. Để làm cho nó ngắn gọn hơn, hiều danh sách đã được sử dụng rộng rãi. Hàm generate() tạo ra một phần tử từ điển chứa xếp hạng trung bình, độ tuổi trung bình, số lượng người xếp hạng nam và nữ, số lượng xếp hạng, movie\_id và tiêu đề của mỗi bộ phim. chức năng gen\_ls() tạo từng phần tử từ điển được tạo bởi hàm tạo().

Khối chính bắt đầu bằng cách đọc ba bộ dữ liệu JSON. Nó tiếp tục bằng cách lấy hai vectơ cột-mỗi movie\_id từ tập dữ liệu phim và movie\_id từ tập dữ liệu xếp hạng. Mỗi vectơ cột được chuyển đổi thành một tập hợp để loại bỏ các bản sao. Các vectơ cột được sử dụng thay cho các bản ghi đầy đủ để xử lý nhanh hơn. Tiếp theo, một vectơ cột mới được trả về chỉ chứa các phim được xếp hạng. Mã tiếp tục bằng cách nhận các vectơ tiêu đề và cột cho xếp hạng cũng như người dùng, độ tuổi và giới tính cho mỗi bộ phim với movie\_id của 1. Xếp hạng trung bình cho phim này được hiển thị cùng với tiêu đề và số lượng xếp hạng. Phần cuối cùng của mã tạo một trình tạo chứa danh sách các thành phần từ điển. Mỗi phần tử từ điển chứa movie\_id, tiêu đề, xếp hạng trung bình, độ tuổi trung bình, số lượng xếp hạng, số lượng người xếp hạng nam và số lượng người xếp hạng nữ. Tiếp theo, một trình tạo khác được tạo để tạo danh sách. Việc tạo trình tạo diễn ra ngay lập tức, nhưng việc làm sáng tỏ (mở ra) nội dung cần có thời gian. Hãy nhớ rằng trình tạo thứ nhất chạy hàng tỷ quy trình và trình tạo thứ 2 chạy trình tạo thứ nhất. Vì vậy, lưu nội dung vào MongoDB mất gần nửa giờ.

Để xác minh kết quả, hãy xem dữ liệu trong MongoDB. Lệnh show collections là lệnh đầu tiên tôi chạy để kiểm tra xem bộ sưu tập movie\_info đã được tạo chưa:

```
> show collections
movie_info
```

Tiếp theo, tôi chạy db.movie\_info.count() để kiểm tra số lượng tài liệu:

```
> db.movie_info.count()
3706
```

## Chương 6 Khám phá dữ liệu

Bây giờ tôi đã biết số lượng tài liệu, tôi có thể hiển thị tài liệu đầu tiên và năm ký lục cuối cùng:

```
> db.movie_info.find({ }, {title:1}).limit(5)
```

```
{'_id': 0, 'title': 'Toy Story'}
{'_id': 1, 'title': 'Jumanji'}
{'_id': 2, 'title': 'Grumpier Old Men'}
{'_id': 3, 'title': 'Waiting to Exhale'}
{'_id': 4, 'title': 'Father of the Bride Part II'}
```

```
> db.movie_info.find({ }, {title:1}).skip(3701)
```

```
{'_id': 3701, 'title': 'Meet the Parents'}
{'_id': 3702, 'title': 'Requiem for a Dream'}
{'_id': 3703, 'title': 'Tigerland'}
{'_id': 3704, 'title': 'Two Family House'}
{'_id': 3705, 'title': 'Contender, The'}
```

Từ quá trình khám phá dữ liệu, có vẻ như bộ sưu tập movie\_info đã được tạo chính xác.

Ví dụ mã thứ 4 lưu ba tập dữ liệu—users.json, cmovies.json và ratings.json—đến MongoDB:

```
nhập sys, os, json, thân thiện với con người dưới dạng
hf từ thời điểm nhập dòng hò
sys.path.append(os.getcwd() + '/classes') import conn
```

```
def read_json(f): với
    open(f) là f: return
        json.load(f)

def create_db(c, d): c =
    db[c]
    c.drop()
```

```

đối với i, hàng trong liệt
kê(d): row['_id']
= i c.insert(row)

if __name__ == "__main__":
    = read_json('data/users.json') u =
    read_json('data/cmovies.json') r =
    read_json('data/ratings.json') obj =
    conn.conn( 'test') db =
    obj.getDB() print
    ('... đang tạo bộ sưu tập MongoDB ...\\n') start =
    clock()
    create_db('users', u)
    create_db('phim', m)
    create_db( 'xếp hạng', r)
    end = clock()
    elapsed_ls = end - start
    print (hf.format_timespan(elapsed_ls, detail=True))

```

Đầu ra:

```

... creating MongoDB collections ...
2 minutes, 28 seconds and 619.93 milliseconds

```

Ví dụ về mã bắt đầu bằng cách nhập sys, os, json, humanfriendly, time và custom class conn. Hàm read\_json đọc JSON. Hàm create\_db() tạo bộ sưu tập MongoDB. Khối chính bắt đầu bằng cách đọc ba bộ dữ liệu-users.json, cmovies.json và ratings.json và lưu chúng vào bộ sưu tập MongoDB. Vì tập dữ liệu ratings.json có hơn một triệu bản ghi nên sẽ mất một khoảng thời gian để lưu tập dữ liệu đó vào cơ sở dữ liệu.

## Chương 6 Khám phá dữ liệu

Ví dụ mã thứ 5 giới thiệu quy trình tổng hợp, đây là khung MongoDB để tổng hợp dữ liệu được mô hình hóa dựa trên khái niệm quy trình xử lý dữ liệu. Các tài liệu đi vào một quy trình nhiều tầng để biến chúng thành các kết quả tổng hợp. Ngoài việc nhóm và sắp xếp tài liệu theo trường hoặc các trường cụ thể và tổng hợp nội dung của mảng, các giai đoạn quy trình có thể sử dụng toán tử cho các tác vụ như tính toán giá trị trung bình hoặc nối chuỗi. Đường ống cung cấp khả năng tổng hợp dữ liệu hiệu quả bằng cách sử dụng các hoạt động MongoDB gốc và là phương pháp ưa thích để tổng hợp dữ liệu trong MongoDB.

```

nhập sys, os
sys.path.append(os.getcwd() + '/classes')
kết nối nhập khẩu

def match_item(k, v, d):
    đường ống = [ {'$match' : { k : v }} ]
    q = db.command('aggregate',d,pipeline=pipeline)
    trả lại q

nếu __name__ == "__main__":
    obj = conn.conn('kiểm tra')
    db = obj.getDB()
    phim = 'Câu chuyện đồ chơi'
    q = match_item('tiêu đề', phim, 'thông tin_phim')
    r = q['kết quả'][0]
    in (phim, 'tài liệu:')
    in (r)
    print ('xếp hạng trung bình', r['avg_rating'], '\n')
    user_id = '3'
    in ('*** người dùng', user_id, '***')
    q = match_item('user_id', user_id, 'users')
    r = q['kết quả'][0]

```

## Chương 6 Khám phá dữ liệu

```

print ('tuổi', r['tuổi'], 'giới tính', r['giới
tính'], 'nghề
nghiệp',\ r['nghề nghiệp'], 'zip', r['zip'],
'\n ') print ('*** "người dùng 3" xếp hạng phim
là 5 ***) q = match_item('user_id', user_id,
'ratings') mid =
q['result'] cho hàng ở giữa:

if row['rating'] == '5':
    q = match_item('movie_id', row['movie_id'], 'movies')
    title = q['result'][0]['title'] thẻ
    loại = q['result'][0]['genres']
    print (row['movie_id'], title, thẻ loại)
mid = '1136'

q = match_item('mid', mid, 'movie_info')
title = q[ 'result'][0]['title']
avg_rating = q['result'][0]['avg_rating']
print ()
print ('"' + title + "'", 'xếp hạng trung bình:', avg_rating )

```

Đầu ra:

```

Toy Story document:
{_id: 0, 'mid': '1', 'title': 'Toy Story', 'avg_rating': 4.15, 'n_ratings': 2077, 'avg_age': 28, 'M': 1486, 'F': 591}
average rating 4.15

*** user 3 ***
age 25 gender M occupation 15 zip 55117

*** "user 3" movie ratings of 5 ***
1079 Fish Called Wanda, A Comedy
1615 Edge, The Adventure|Thriller
1259 Stand by Me Adventure|Comedy|Drama
2167 Blade Action|Adventure|Horror
260 Star Wars: Episode IV - A New Hope Action|Adventure|Fantasy|Sci-Fi
1264 Unforgiven Western
733 Rock, The Action|Adventure|Thriller
2355 Bug's Life, A Animation|Children's|Comedy
1197 Princess Bride, The Action|Adventure|Comedy|Romance
1198 Raiders of the Lost Ark Action|Adventure
1378 Young Guns Action|Comedy|Western
3552 Caddyshack Comedy
1304 Butch Cassidy and the Sundance Kid Action|Comedy|Western
3671 Blazing Saddles Comedy|Western
1136 Monty Python and the Holy Grail Comedy

"Monty Python and the Holy Grail" average rating: 4.34

```

## Chương 6 Khám phá dữ liệu

Ví dụ mã bắt đầu bằng cách nhập sys, os và lớp tùy chỉnh conn.

Hàm match\_item() sử dụng quy trình tổng hợp để khớp các bản ghi với tiêu chí. Khối chính bắt đầu bằng cách sử dụng đường dẫn tổng hợp để trả về tài liệu Câu chuyện đồ chơi từ bộ sưu tập movie\_info. Mã tiếp tục bằng cách sử dụng đường dẫn để trả lại tài liệu người dùng 3 từ người dùng bộ sưu tập. Tiếp theo, quy trình tổng hợp được sử dụng để trả về tất cả xếp hạng phim là 5 cho người dùng 3. Cuối cùng, quy trình này được sử dụng để trả về xếp hạng trung bình cho Monty Python và Chén Thánh từ bộ sưu tập movie\_info.

Đường ống tổng hợp hiệu quả và cung cấp một loạt các chức năng.

Ví dụ mã thứ 6 minh họa một đường dẫn tổng hợp nhiều tầng:

```
nhập sys, os
sys.path.append(os.getcwd() + '/classes')
kết nối nhập khẩu
```

các giai đoạn xác định (k, v, r, d):

```
đường ống = [ { '$match' : { '$and' : [ { k : v }, { 'xếp hạng':{$eq}:r } ] } },
{ '$dự án' : {
    '_id' : 1,
    'user_id' : 1,
    'phim_id' : 1,
    'xếp hạng': 1 } },
{ '$limit' : 100}]
```

```
q = db.command('aggregate',d,pipeline=pipeline)
```

```
trả lại q
```

```
def match_item(k, v, d):
    đường ống = [ {'$match' : { k : v }} ]
    q = db.command('aggregate',d,pipeline=pipeline)
    return q

if __name__ == "__main__":
    obj = conn.conn('test')
    db = obj.getDB()
    u = '3'
    r = '5'

    q = giai đoạn('user_id', u, r, 'xếp
    hạng' ) result =
    q['result'] print ('xếp hạng của',      + str(u) + ':')
    r, 'cho user cho i, row in

    enumerate(result): print (row) n = i+1
    print ()
    print (n, 'tên phim liên quan:') for i,
    row in enumerate(result): q =
        match_item('movie_id', row['movie_id'], 'phim') r =
        q['result'][0] in
        (r['tiêu đề'])
```

## Chương 6 Khám phá dữ liệu

Đầu ra:

```

ratings of 5 for user 3:
{'_id': 192, 'user_id': '3', 'movie_id': '1079', 'rating': '5'}
{'_id': 194, 'user_id': '3', 'movie_id': '1615', 'rating': '5'}
{'_id': 196, 'user_id': '3', 'movie_id': '1259', 'rating': '5'}
{'_id': 198, 'user_id': '3', 'movie_id': '2167', 'rating': '5'}
{'_id': 201, 'user_id': '3', 'movie_id': '260', 'rating': '5'}
{'_id': 209, 'user_id': '3', 'movie_id': '1266', 'rating': '5'}
{'_id': 210, 'user_id': '3', 'movie_id': '733', 'rating': '5'}
{'_id': 213, 'user_id': '3', 'movie_id': '2355', 'rating': '5'}
{'_id': 214, 'user_id': '3', 'movie_id': '1197', 'rating': '5'}
{'_id': 215, 'user_id': '3', 'movie_id': '1198', 'rating': '5'}
{'_id': 216, 'user_id': '3', 'movie_id': '1378', 'rating': '5'}
{'_id': 219, 'user_id': '3', 'movie_id': '3552', 'rating': '5'}
{'_id': 220, 'user_id': '3', 'movie_id': '1304', 'rating': '5'}
{'_id': 226, 'user_id': '3', 'movie_id': '3671', 'rating': '5'}
{'_id': 231, 'user_id': '3', 'movie_id': '1136', 'rating': '5'}

15 associated movie titles:
Fish Called Wanda, A
Edge, The
Stand by Me
Blade
Star Wars: Episode IV - A New Hope
Unforgiven
Rock, The
Bug's Life, A
Princess Bride, The
Raiders of the Lost Ark
Young Guns
Caddyshack
Butch Cassidy and the Sundance Kid
Blazing Saddles
Monty Python and the Holy Grail

```

Ví dụ mã bắt đầu bằng cách nhập sys, os và lớp tùy chỉnh conn.

Các giai đoạn chức năng() sử dụng đường dẫn tổng hợp ba giai đoạn. Giai đoạn 1 tìm tất cả xếp hạng 5 từ người dùng 3. Giai đoạn 2 dự đoán các trường sẽ được hiển thị. Giai đoạn thứ 3 giới hạn số lượng tài liệu được trả lại. Điều quan trọng là bao gồm một giai đoạn giới hạn, bởi vì cơ sở dữ liệu kết quả lớn và các đường ống dẫn có các giới hạn về kích thước. Hàm match\_item() sử dụng quy trình tổng hợp để khớp các bản ghi với tiêu chí. Khối chính bắt đầu bằng cách sử dụng đường dẫn stage() để trả về tất cả xếp hạng 5 từ người dùng 3. Mã tiếp tục bằng cách lặp lại dữ liệu này và sử dụng đường dẫn match\_item() để nhận các tiêu đề mà người dùng 3 xếp hạng là 5. Đường dẫn là một phương pháp hiệu quả để truy vấn tài liệu từ MongoDB, nhưng cần thực hành để làm quen với cú pháp của nó.

## Twitter

Twitter là một nguồn dữ liệu tuyệt vời vì bạn có thể lấy dữ liệu về hầu hết mọi thứ. Để truy cập dữ liệu từ Twitter, bạn cần kết nối với Twitter Streaming API. Kết nối yêu cầu bốn mẫu thông tin từ Twitter-Khóa API, bí mật API, mã thông báo truy cập và bí mật mã thông báo truy cập (được mã hóa). Sau khi đăng ký và nhận thông tin đăng nhập, bạn cần cài đặt API Twitter. Tôi đã chọn Tìm kiếm Twitter API, nhưng có nhiều cái khác.

Ví dụ mã đầu tiên tạo JSON để giữ thông tin đăng nhập Twitter của tôi (chèn thông tin đăng nhập của bạn vào từng biến):

nhập json

```
nếu __name__ == '__main__':
    Consumer_key =
    Consumer_secret =
    access_token =
    access_encrypted = data
    = {}

    data['ck'] = Consumer_key
    data['cs'] = Consumer_secret
    dữ liệu['at'] = access_token
    data['ae'] = access_encrypted
    json_data = json.dumps(đữ liệu)
    tiêu đề = '[\n'
    kết thúc = ']'

    obj = open('data/credentials.json', 'w')
    obj.write(tiêu đề)
    obj.write(json_data + '\n')
    obj.write(kết thúc)
    obj.close()
```

## Chương 6 Khám phá dữ liệu

Tôi đã chọn lưu thông tin xác thực trong JSON để ẩn chúng khỏi chế độ xem. Mật mã ví dụ nhập thư viện json. Khỏi chính lưu thông tin xác thực vào JSON.

Ví dụ mã thứ 2 truyền dữ liệu Twitter bằng TwitterSearch API. Để cài đặt: pip cài đặt TwitterSearchAPI.

```
*  
từ TwitterSearch nhập json,  
sys
```

lớp twitTim kiếm:

```
def __init__(bản thân, tín dụng, ls, giới hạn):  
    self.cred = tín dụng  
    self.ls = ls  
    self.limit = giới hạn  
  
tìm kiếm xác định (bản thân):  
    số = 0  
  
    dt = []  
    dic = {}  
  
    thử:  
        tso = TwitterSearchOrder()  
        tso.set_keywords(self.ls)  
        tso.set_language('en')  
        tso.set_include_entities(False) ts =  
        TwitterSearch( Consumer_key  
            = self.cred[0]['ck'], Consumer_secret =  
            self.cred[0]['cs'], access_token = self.cred[0]  
            ['at'], access_token_secret = self.cred[0]  
            ['ae'] )  
  
        cho tweet trong ts.search_tweets_iterable(tso):  
            nếu num <= self.limit:  
                dic['_id'] = num  
                dic['tweeter'] = tweet['user']['screen_name']  
                dic['tweet_text'] = tweet['text']
```

```

        dt.append(dic)
        dic = {}
khác:
    phâ vỡ
    số += 1

ngoại trừ TwitterSearchException as e:
    print(e)
return dt

def get_creds():
    với open('data/credentials.json') dưới dạng json_data:
        d = json.load(json_data)
        json_data.close()
    trả về d

def write_json(f, d):
    với open(f, 'w') là lõi:
        json.dump(d, fout)

def translate():
    trả về dict.fromkeys(range(0x10000, sys.maxunicode + 1), 0xffffd)

def read_json(f):
    với open(f) là f:
        return json.load(f)

if __name__ == '__main__':
    cred
    = get_creds()
    ls =
    ['máy', 'học']
    giới hạn = 10

    obj = twitSearch(cred, ls, limit)
    data
    = obj.search()
    f =
    'data/TwitterSearch.json'

```

## Chương 6 Khám phá dữ liệu

```

write_json(f, data)
non_bmp_map = translate()
print ('twitter data:')
cho hàng trong

dữ liệu: row['tweet_text'] =
str(row['tweet_text']).
translate(non_bmp_map) tweet_text =
row['tweet_text'][0:50] print ('{:<3}{:18s}
{}'.format(row['_id'],
row['tweeter'],
tweet_text )) print
('`nxiá minh JSON:') read_data =

read_json(f) for i, p in enumerate(read_data):
    if i < 3: p['tweet_text']
    = str(p['tweet_text']).
    translate(non_bmp_map) tweet_text =
    p['tweet_text'][0:50] print ('{:<3}{:18s}{}'.format(p['_id'], p['tu

```

Đầu ra:

```

twitter data:
0 TradingEngineer4 RT @sonmdevelopment: Great news! Running ML algori
1 lrsevey #RT @Nexosis: How Businesses Are Using AI and Mach
2 ICM_Change Digital transformation: How machine learning could
3 RawadKhazem RT @SASSoftware: The ultimate artificial intellige
4 F5Security "#CISOs look to #MachineLearning to augment securi
5 jayhinman RT @Ascendify: Bringing talent and intelligence to
6 meisshaily Understanding Machine Learning [INFOGRAPHIC] https
7 eriningrassia RT @DeepLearn007: Machine Learning & Marketing
8 trishia_ani RT @wef: A computer was asked to predict which sta
9 AmandaMSaunders RT @dougtroll11: Hear from Volkswagen at #GTC18 on
10 DD_Bun_ Machine Learning: What's in It for Business? https

verify JSON:
0 TradingEngineer4 RT @sonmdevelopment: Great news! Running ML algori
1 lrsevey #RT @Nexosis: How Businesses Are Using AI and Mach
2 ICM_Change Digital transformation: How machine learning could

```

## Chương 6 Khám phá dữ liệu

Ví dụ về mã bắt đầu bằng cách nhập các thư viện TwitterSearch, json và sys. Lớp twitSearch truyền dữ liệu Twitter dựa trên thông tin đăng nhập Twitter, danh sách từ khóa và giới hạn. Hàm get\_cred() trả về thông tin đăng nhập Twitter từ JSON. Hàm write\_json() ghi dữ liệu vào JSON. Hàm translate() chuyển đổi dữ liệu được truyền phát bên ngoài Mật phẳng đa ngôn ngữ cơ bản (BMP) sang định dạng có thể sử dụng được. Ví dụ, biểu tượng cảm xúc nằm ngoài BMP. Hàm read\_json() đọc JSON. Khối chính bắt đầu bằng cách lấy thông tin đăng nhập Twitter, tạo danh sách từ khóa tìm kiếm và giới hạn. Trong trường hợp này, danh sách các từ khóa tìm kiếm chứa machine and learning, vì tôi muốn truyền dữ liệu về machine learning. Giới hạn mười hạn chế các bản ghi được phát trực tuyến thành mười tweet. Mã tiếp tục bằng cách ghi dữ liệu Twitter vào JSON, dịch các tweet để kiểm soát dữ liệu không phải BMP và in tweet. Cuối cùng, mã đọc JSON để xác minh rằng các tweet đã được lưu đúng cách và in một số.

## Rút trích nội dung trang web

Quét web là một cách tiếp cận có lập trình để trích xuất thông tin từ các trang web. Nó tập trung vào việc chuyển đổi dữ liệu định dạng HTML phi cấu trúc thành dữ liệu có cấu trúc. Quét web đòi hỏi nhiều lập trình vì bản chất không có cấu trúc của HTML. Đó là, HTML có rất ít nếu có bất kỳ quy tắc cấu trúc nào, điều đó có nghĩa là các mẫu cấu trúc HTML có xu hướng khác nhau giữa các trang web. Vì vậy, hãy sẵn sàng viết mã tùy chỉnh cho mỗi cuộc phiêu lưu tìm kiếm trên Web.

Ví dụ này lấy thông tin sách từ một công ty xuất bản sách kỹ thuật nổi tiếng. Bước đầu tiên là xác định vị trí trang web. Bước thứ 2 là mở một cửa sổ có mã nguồn. Bước thứ 3 là duyệt qua mã nguồn để xác định dữ liệu cần cạo. Bước thứ 4 là cạo.

Với Google Chrome, nhập vào Công cụ khác rồi nhập vào Công cụ dành cho nhà phát triển để mở cửa sổ mã nguồn. Tiếp theo, di con trỏ chuột qua nguồn cho đến khi bạn tìm thấy dữ liệu. Di chuyển xuống cây mã nguồn để tìm các thẻ bạn muốn cạo. Cuối cùng, cạo dữ liệu.

## Chương 6 Khám phá dữ liệu

Để cài đặt 'BeautifulSoup', hãy cài đặt BeautifulSoup.

từ bs4 nhập Yêu cầu nhập BeautifulSoup,

json

```
def build_title(t):
```

```
    t = t.văn bản
```

```
    t = t.split() ls
```

```
    = [] cho
```

```
    hàng trong t:
```

```
        nếu hàng != '-':
```

```
            ls.append(row)
```

```
        elif row == '-':
```

```
            break
```

```
    trở lại ' '.join(ls)
```

xác định ngày phát hành (r):

```
    r = r.văn bản
```

```
    r = r.split()
```

```
    tiền tố = r[0] + s + r[1] if
```

```
    len(r) == 5: date
```

```
        = r[2] + s + r[3] + s + r[4] other :
```

```
        ngày = r[2] + s + r[3] trả
```

```
        về tiền tố, ngày
```

```
def write_json(f, d): với
```

```
    open(f, 'w') là lõi: json.dump(d,
```

```
        fout)
```

```
def read_json(f): với
```

```
    open(f) là f: return
```

```
        json.load(f)
```

```

nếu __name__ == '__main__':
    s =
        dic_ls = []
        base_url = "https://sssearch.oreilly.com/?q=data+science" soup =
BeautifulSoup(requests.get(base_url).text, 'lxml') books =
soup.find_all('article') cho i,
hàng trong liệt kê(sách): dic = {}
        tag =
            row.name tag_val
            = row['class'] title =
                row.find('p', {'class' : 'title'}) title =
                    build_title(title) url =
                        row.find('a', {'class' : 'tìm hiểu thêm'}) learn_more
                        = url.get('href') author =
                            row.find('p', {'class' : 'ghi chú'}).text release =
                                row.find('p', {'class' : 'note date2'}) tiền tố, ngày =
                                    release_date(release) if len(tag_val) ==
                                        2: Publisher = row.
                                            find('p', {'class' : 'nhà xuất bản ghi chú'}).text
                                                item =
                                                    row.find('img', {'class' : 'book'}) cat =
                                                        item.get('class')[0]
khác:
    nhà xuất bản, mèo = Không có,
    Không có desc = row.find('p', {'class' : 'description'}).
        text.split()
    desc = [hàng cho i, hàng trong liệt kê(desc) nếu tôi < 7]
    desc = '...'.join(desc) +
        dic['title'] = title
        dic['learn_more'] = learn_more if
            author[0:3] != 'Pub':

```

## Chương 6 Khám phá dữ liệu

```

dic['author'] = tác giả nếu
nhà xuất bản không có Không có:
dic['publisher'] = nhà xuất bản
dic['category'] = con mèo
khác:
dic['event'] = desc
dic['date'] = ngày
dic_ls.append(dic) f =
'data/scraped.json'
write_json(f, dic_ls) data
= read_json(f) for i, row
in enumerate( dữ liệu):
nếu tôi < 6:
    in (hàng['tiêu đề']) nếu
    'tác giả' trong hàng.keys(): in
        (hàng['tác giả']) nếu 'nhà
        xuất bản' trong hàng.keys(): in
            (hàng['nhà xuất bản']) nếu 'danh
            mục' trong row.keys(): print ('Danh
            mục:', hàng['danh mục']) print ('Ngày phát
            hành:', hàng['ngày']) nếu 'sự kiện' trong
            hàng.keys(): print ('Sự kiện:',
            hàng ['sự kiện']) print ('Ngày xuất
            bản:', hàng ['ngày']) print ('Tìm hiểu thêm:',
            hàng ['tìm hiểu thêm']) print ()

```

Đầu ra:

```

Going Pro in Data Science
By Jerry Overton
Publisher: O'Reilly Media
Category: book
Release Date: March 15, 2016
Learn more: http://www.oreilly.com/data/free/going-pro-in-data-science.csp

2015 Data Science Salary Survey
By John King, Roger Magoulas
Publisher: O'Reilly Media
Category: book
Release Date: September 11, 2015
Learn more: http://www.oreilly.com/data/free/2015-data-science-salary-survey.csp

2016 Data Science Salary Survey
By John King, Roger Magoulas
Publisher: O'Reilly Media
Category: book
Release Date: September 16, 2016
Learn more: http://www.oreilly.com/data/free/2016-data-science-salary-survey.csp

Ten Signs of Data Science Maturity
By Peter Guerra, Kirk Borne
Publisher: O'Reilly Media
Category: book
Release Date: March 14, 2016
Learn more: http://www.oreilly.com/data/free/ten-signs-of-data-science-maturity.csp

Statistics for Data Science
By James Miller
Publisher: Packt Publishing
Category: book
Release Date: November 2017
Learn more: http://shop.oreilly.com/product/9781788290678.do

Scalable Data Science on a Laptop
By Alice Zheng
Event: Hosted By: Ben Lorica Watch the webcast ...
Publish Date: February 12, 2016
Learn more: http://www.oreilly.com/pub/e/3124

```

Ví dụ về mã bắt đầu bằng cách nhập các thư viện BeautifulSoup, request và json. Hàm build\_title() xây dựng dữ liệu tiêu đề đã loại bỏ thành một chuỗi. Hàm release\_date() xây dựng dữ liệu ngày đã loại bỏ thành một chuỗi. Hàm write\_json() và read\_json() viết và đọc JSON tương ứng. Khối chính bắt đầu bằng cách chuyển đổi trang URL thành đối tượng BeautifulSoup.

Mã tiếp tục bằng cách đặt tất cả các thẻ bài viết vào sách biển. Từ việc khám phá, tôi thấy rằng các thẻ bài viết chứa thông tin mà tôi muốn thu thập. Tiếp theo, mỗi thẻ bài viết được duyệt qua. Việc cạo sẽ dễ dàng hơn nhiều nếu thông tin trong mỗi thẻ bài viết được cấu trúc nhất quán. Vì không phải vậy nên logic để trích xuất từng mẩu thông tin rất rộng. Mỗi phần thông tin được đặt trong một phần tử từ điển, phần tử này sau đó được thêm vào danh sách. Cuối cùng, danh sách được lưu vào JSON.

JSON được đọc và một vài bản ghi được hiển thị để xác minh rằng tất cả đều ổn.

# Mục lục

MỘT

đường ống tổng hợp  
nhiều tầng, [198-200](#) sử  
dụng các hoạt động gốc, [196-198](#)

Đường cong Andrews, [141-143](#)

b

Súp Đẹp, [206](#)

Dữ liệu lớn

MongoDB, bộ dữ  
liệu phim [189](#), [185](#)  
Dữ liệu MovieLens sang JSON, [182](#) bộ  
dữ liệu lưu, [194-195](#) ba bộ dữ  
liệu, [189](#), [194](#)

C

Tệp Giá trị được Phân tách bằng Đầu

phầy (CSV), [146-148](#)

tương quan

dữ liệu  
hệ số đo lường, [135-137](#) phân phối  
bình thường và,  
[134-135](#)

gầu trúc (xem tương quan gầu trúc)

Dữ liệu khói, [149-150](#)

Chức năng phân phối tích lũy

(CDF), [52-60](#), [62](#), [64-65](#), [134](#)

© David Paper 2018

D. Paper, Nguyên tắc cơ bản về khoa học dữ liệu cho Python và  
MongoDB, <https://doi.org/10.1007/978-1-4842-3597-3>

Đ.

Dữ liệu

Đường cong Andrews, tương quan  
[141-143](#)

hệ số đo lường, [135-137](#) phân phối  
bình thường

và, [134-135](#) khói

lập phương, [149-](#)

[150](#) hạt chia, [148-](#)

[149](#) phân phối chuẩn, [129-131](#) một  
chiều, [129-132](#) gầu trúc (xem  
tương quan Pandas) tọa

độ song song,  
[143-144](#)

RadViz, phân

phối

chuẩn theo tỷ lệ [144](#), [154-156](#) lần

đọc bộ dữ liệu CSV,

[154](#), [157](#), [161](#)

trung bình và đơn vị bằng không

phương sai, [157-161](#)

cắt lát, [148-149](#)

hai chiều, phân bố đồng đều [132-](#)

[135](#),

[129-132](#)

tranh cãi, [129](#), [161](#), [163-165](#)

Khung dữ liệu (df), [88](#)

Ngẫu nhiên hóa dữ liệu, [22-27](#)

## Mục lục

## Khoa học dữ liệu

nguyên tắc cơ bản,  
[2 đọc và ghi dữ liệu, trực quan hóa](#)  
[12-15 , 34-36](#)

Dữ liệu thái hạt lựu, [148-149](#)

Từ điển, [10-12](#)

## E, F

khoảng cách Euclidean

giảm thiểu

Không gian 3 chiều , [109-112](#)

MCS, phương pháp

numpy [112-114 , 115, 117-118](#)

## G

máy phát điện

lợi thế, [22](#)  
 hiệu, [21-22](#)  
 Các phần tử OrderedDict, [19](#)  
 hàm `read_dict()`, [21](#) hàm  
`sim_times()`, [21](#) mô phỏng, [19,](#)  
[21](#)

Thuật toán giảm độ dốc

(GD), [2](#)  
 khoảng cách Euclidean (xem  
 giảm thiểu khoảng cách  
 Euclidean) chức  
 năng đơn giản (xem Giảm thiểu  
 chức năng đơn giản)  
 ngẫu nhiên (xem Giảm dần độ  
 dốc ngẫu nhiên)

Trang web GroupLens, [182](#)

## h

Bản đồ nhiệt,  
 tương quan [140](#) gấu trúc và [141](#)  
 dữ liệu wrangled.json, [167-169](#)

Phân phối tích lũy nghịch đảo  
 chức năng (ICDF), [52](#)  
 Bộ dữ liệu Iris, [176-177, 179](#)

Dữ liệu đã làm sạch JSON thành, [146-148](#)  
 MongoDB và, [27-34](#)

## L

Đánh giá lười biếng, [18](#)  
 Đại số tuyến tính,  
[67](#) phép biến đổi ma trận cơ bản, [84](#)  
 thuộc tính đơn vị, [84-85](#) ma  
 trận numpy, [86-88](#) lát  
 cắt, [85](#)  
 phép toán ma trận, [75,](#)  
[83](#) hàm `display()`, [81](#) hàm  
`dot()`, độ lớn [80](#) ,  
 phép nhân [82-83 , 78](#)  
[-80](#) hàm `mult_scalar()`, [77](#)  
 ma trận numpy, [75-76](#) trung bình  
 ứng dụng ma trận gấu  
 trúc, [94-96](#)

- cáu trúc dữ liệu, [92-93](#)
- df,
- [88](#) phương thức
- `head()` và `tail()`, [90](#)
- ma trận numpy, phép toán
- `vector` [91-92](#), đoạn thẳng
- có hướng cộng, độ lớn [68](#),
- [74](#) từ gốc tọa
- độ, phép trừ
- [69](#), trực quan hóa
- [73-74](#), không gian
- `vector` [71-73](#), [67](#)
- danh sách
- hiều, [15-18](#) tạo mới, [6-9](#)
- thành phần tử
- diễn, [11](#) bộ và, [9-10](#)
- m**
- Đường ống
- tổng hợp MongoDB
- nhiều tầng, [198](#), [200](#)
- sử dụng các hoạt động gốc, [196](#), [198](#)
- kết nối với, [189](#)
- JSON và, [27-34](#) bộ
- dữ liệu lưu, [194-195](#)
- Mô phỏng Monte Carlo (MCS), [37](#), [112-114](#)
- biểu đồ tăng cường, [41](#) dự đoán chính xác mô phỏng
- nhu cầu sản phẩm, [49-51](#) hàm nhu cầu (), [48](#)
- hàm `mcs()`, [51](#) hàm
- `max_bar()`, [48](#) phân phối
- chuẩn, [44](#) thuật toán sản xuất, [45-47](#) hàm sản xuất(), [48](#)
- tính ngẫu nhiên sử dụng PDF
- và
- CDF, [52-53](#)
- ngẫu nhiên liên tục
- biên, [52](#)
- định lý cơ bản của
- phép tính,
- giảm độ dốc, [53](#)
- Tích hợp ICDF, [52](#), [57-58](#), [52](#)
- biến thể ít hơn (lỗi), [59](#)
- thư viện matplotlib, [53-54](#)
- hàm `np.rstrip()`, [58](#)
- PPF, [52](#)
- phép tính lợi nhuận, [59-62](#)
- trực quan, [55-56](#), [63-65](#)
- Phân tích What-If, [59](#)
- mô phỏng chứng khoán, [37](#), [39-41](#)
- Phân tích What-If, [42-44](#)
- Bộ dữ liệu phim, [185](#)
- Trang web MovieLens, [182](#)
- N**
- NoSQL, [27](#)
- Phương pháp Numpy, [115](#), [117-118](#)
- 0**
- Dữ liệu một chiều, [129-132](#)

## Mục lục

**P, Q**

Tệp CSV làm sạch

tương quan gấu trúc, [146-148](#)  
tạo trực quan hóa ma trận, [138-139](#)

bản đồ nhiệt, [140-141](#)Tọa độ song song, [143-144](#)Hàm điểm phần trăm (PPF), [52](#)

Phân tích thành phần chính (PCA)

eigenvector, tập

dữ liệu iris [170](#), tậpdữ liệu [176-179](#) wrangled.json, [170, 174-175](#)Hàm mật độ xác suất (PDF), [52-60, 62-65](#)PyMongo, [28](#)

Hàm và

chuỗi Python, [3-5](#) nguyên tắc  
cơ bản, [3](#) ma trận,  
[9](#) mã lập  
trình, [2](#) số giả ngẫu  
nhiên, [22](#)

Dữ liệu cắt lát, [148-149](#)Mô phỏng tốc độ, [179, 181](#)Độ lệch chuẩn (SD), [132](#)

tiêu chuẩn bình thường

phân phối, [154, 161](#)

Lô giảm dần độ dốc ngẫu nhiên,

tối đa [118-119](#),

theta tối thiểu

[126, 123](#) n lần, [122](#)Hàm RSS, [119](#) chạy nlần, [119](#)**t**Tuple, [9-10](#)

Twitter

pip cài đặt TwitterSearchAPI, [202](#) lưu  
thông tin xác thực trong JSON,  
[201-202](#)

Dữ liệu hai chiều, [132-135](#)

bạn

Phân bố đều, [129-132](#)**V**Quán tưởng, [34-36](#)**W, X, Y, Z**Quét web, BeautifulSoup, [205-209](#)Dữ liệu lộn xộn, [129, 161, 163-165](#)**R**RadViz, [144](#)Tổng bình phương còn lại (RSS), [119](#)**S**

Tối thiểu hóa hàm sigmoid tối đa  
cực bộ, [107-109](#) tối thiểu cực  
bộ, [104-107](#)

Giảm thiểu chức năng đơn giản,  
[97-103](#)