THỐNG KÊ DỮ LIỆU KINH DOANH VỚI R

NGUYỄN QUANG (quang.nguyen@polytechnique.org)

Tài liệu này được biên soạn nhằm giúp cho các chuyên viên phân tích trong doanh nghiệp có được công cụ xử lý dữ liệu ngày càng tăng lên, có thể lên đến nhiều GB. Đây cũng là quy mô dữ liệu phổ biến của phần lớn các doanh nghiệp (vừa và nhỏ) ở Việt Nam hiện nay. Trước khi công ty có thể trang bị các hệ thống Big data đủ lớn thì công cụ R có thể là giải pháp hợp lý, chi phí thấp, khả năng triển khai nhanh cho những nhiệm vụ phân tích dữ liệu kinh doanh. R không chỉ bao gồm tất cả các chức năng của Excel mà còn cung cấp rất nhiều chức năng thống kê và mô hình hóa khác mà người phân tích kinh doanh cần làm chủ. Ngoài ra, đây cũng là kiến thức tốt mà các bạn sinh viên trước khi ra trường nên trang bị, nhất là các khối ngành kinh tế, tài chính-kế toán, marketing,…

Tôi xin cám ơn các đồng nghiệp và ban lãnh đạo công ty Cổ phần hỗ trợ dịch vụ thanh toán Việt Phú (Mobivi/IcareBenefits) đã cộng tác, chia sẻ và chỉ dẫn rất nhiều bài toán kinh doanh cho tôi. Sự trợ giúp đó đã giúp cho tôi có được góc nhìn ứng dùng phù hợp để kết hợp với những phương pháp và công cụ khoa học. Tôi cũng xin cám ơn các sinh viên ở trường Đại học, các học viên lớp "Data Science and Big Data Analytics" đã đóng góp ý kiến, đọc bản thảo và giúp chỉnh sửa nhiều đoạn trong tài liệu.

Tài liệu xin được gửi tặng đến tất cả các bạn đọc mong muốn khám phá dữ liệu!

TPHCM 2018

Giới thiệu

Ngôn ngữ phân tích dữ liệu trong doanh nghiệp hiện nay chính là Microsoft Office Excel[@]. Điểm mạnh lớn nhất của Excel là giao diện trực quan, người dùng tận mắt nhìn và thao tác dữ liệu, các kết quả phân tích, chuyển đổi dữ liệu. Excel còn có khả năng hiển thị xuất sắc với các đồ thị và khả năng tích hợp các công cụ văn phòng khác của Microsoft Office. Tuy nhiên Excel đang gặp phải những yếu điểm lớn xuất hiện trong thời đại dữ liệu này:

- Khả năng xử lý dữ liệu lớn – không thể xử lý các bộ dữ liệu lớn hơn khoảng 50 MB
- Dữ liệu, thông tin phân tán
- Các mô hình Khoa học dữ liệu hạn chế, không có khả năng triển khai các mô hình nâng cao (dự báo hay phân loại văn bản, nhận dạng hình ảnh, …)
- Không có khả năng lập trình để thực hiện các thao tác lặp lại, các hàm chuyên dụng,…

R là một trong những công cụ/ngôn ngữ được dùng để phân tích dữ liệu phổ biến hiện nay. R có nhiều ưu điểm giúp khắc phục được phần lớn những hạn chế nói trên:

- R xử lý dữ liệu trên RAM máy tính, tốc độ xử lý nhanh và khả năng xử lý dữ liệu lên đến nhiều GB (tùy thuộc RAM)
- R được viết từ ứng dụng thống kê nên có nhiều công cụ thống kê, máy học và phân tích dữ liệu nâng cao, hoặc được tích hợp hoặc có sẵn ở các gói thư viện phong phú (hơn 5000 thư viện)
- R là phần mềm mã nguồn mở, có hệ thống trợ giúp tốt, có cộng đồng phát triển đông đảo và đồ họa đẹp.

Nhưng R cũng có những hạn chế như:

- Chưa trực tiếp xử lý được dữ liệu lớn Big data (cỡ và trăm GB đến TB)
- Là ngôn ngữ lập trình nên cần thời gian học và làm quen, có thể là trở ngại với nhiều người chưa từng lập trình
- Giao diện không thuận tiện như các phần mềm khác
- Không có bộ phận trợ giúp thương mại

Xét trên góc độ doanh nghiệp vừa và nhỏ, R phù hợp cho việc thực hiện các tác vụ phân tích dữ liệu, mô hình dữ liệu, thậm chí thực hiện các dự án dữ liệu doanh nghiệp quan trọng. R có thể là một công cụ giúp xử lý dữ liệu lớn trở nên "bình dân” - đa số chuyên viên có thể làm được phần lớn các tác vụ phân tích dữ liệu - thay vì phụ thuộc vào một bộ phận chuyên gia ít ỏi trong doanh nghiệp.

R được Robert Gentleman và Ross Ihaka, khoa Thống kê của Đại học Auckland phát triển lên từ các ngôn ngữ S và S-Plus của AT&T năm 1995. Ban đầu chỉ phục vụ cho việc thống kê, sau đó đã được nhân rộng cho việc xử lý dữ liệu nói chung. R hiện đang được hỗ trợ bởi một đỗi ngũ core, tình nguyện, với thông tin chi tiết trên trang http://www.r-project.org. Đây cũng là nơi có chứa tất cả các thư viện hỗ trợ và tài liệu khác.

1 Cơ bản về R

Ngôn ngữ dạng câu lệnh như R tỏ ra khó học và không gần gũi với người phân tích kinh doanh. Tuy nhiên, chỉ cần bỏ một thời gian ban đầu là bạn sẽ có thể nắm được những cú pháp cơ bản và hiểu được những logic trong xử lý dữ liệu. Sau đó bạn sẽ nhận thấy là với R, cũng như một số ngôn ngữ khác, bạn hoàn toàn làm chủ được dữ liệu, cảm nhận được chúng và thao tác giống như với các ngôn ngữ "drag-and-drop" khác.

Cũng vì rào cản trên, tôi sẽ biên soạn giáo trình này thật ngắn gọn để hy vọng người học sẽ không sớm nản, tập trung vào ứng dụng còn các câu lệnh phức tạp hơn sẽ được gom lại vào phụ lục. Nếu bạn nào vẫn còn thấy ngại khi bắt đầu học các dòng lệnh thì tôi có thể khuyên bạn đọc lướt chương này, viết lệnh nhưng chưa cần hiểu hết rồi thực hành các ví dụ trong các chương sau. Sau đó bạn có thể quay trở lại chương 1 một cách đầy đủ hơn.

1.1 Cài đặt R

Cách nhanh nhất là tải bản binary từ Website chính thức của R - http://www.R-project.org. Nó sẽ dẫn bạn đến trang của CRAN ("ComprehensiveRArchive Network”). Bạn cài đặt theo hướng dẫn, sẽ đơn giản như cài một phần mềm đóng gói sẵn. Tuy nhiên chúng tôi khuyến nghị các bạn cài đặt thêm bản giao diện **Rstudio** (tương tự giao diện của Matlab) tại trang www.rstudio.com. Các ví dụ trong tài liệu này sẽ mô tả trên phiên bản R và RStudio cài trên hệ điều hành MacOS.

Sau khi cài đặt và khởi động Rstudio bạn sẽ thấy 4 cửa sổ chính:

- Cửa sổ biên soạn (Editor) (Bên trên – trái): dùng để soạn chương trình
- Cửa sổ câu lệnh (Console) (Bên dưới – trái): dùng để gõ trực tiếp câu lệnh và hiển thị kết quả
- Cửa sổ Môi trường (Environment) (Bên trên – phải): hiển thị các biến và đối tượng hiện có/Các câu lệnh cũ
- Cửa sổ Thư mục (File) (Bên dưới – phải): hiển thị các file trong thư mục/đồ thị/

Trên màn hình câu lệnh Console sẽ hiện lên dấu nhắc, ">”. Trong quá trình làm việc nếu thấy dấu nhắc có nghĩa là R đang chờ bạn nhập một câu lệnh mới.

Màn hình **Console**:

```
>
```

Từ màn hình Console bạn có thể gõ một câu lệnh và ấn ENTER để thực hiện. Nếu câu lệnh đúng có thể có giá trị trả về hoặc không và một dấu nhắc mới xuất hiện. Nếu câu lệnh sai thì chương trình sẽ báo lỗi.

Bạn cũng có thể dùng các dấu mũi tên để truy lại các câu lệnh cũ, rất cần thiết trong quá trình làm việc. Bạn cũng có thể Copy->Paste một đoạn mã vào Console hoặc Editor.

Nếu cần thoát R, bạn gõ câu lệnh q().

Cửa sổ **Console**:

```
>q()
```

Chương trình cơ bản của R đã khá mạnh, tuy vậy tiềm năng lớn của R lại ở tập hợp các gói công cụ/thư viện mà cộng đồng hỗ trợ tạo nên. Các thư viện này đặc biệt hữu ích và giúp việc lập trình trở nên ngắn gọn và hiệu quả hơn nhiều. Thư viện có thể được cài đặt theo 2 cách:

- Chọn Menu Tools -> Install Packages. Gõ tên hoặc chọn file packages nếu có sẵn
- Từ Console gõ dòng lệnh :

Cửa sổ Console:

```
> install.packages("ggplot2”)
```

Trên đây là ví dụ cài đặt package "ggplot2", một gói hỗ trợ mạnh cho việc vẽ đồ thị. Để kiểm tra các gói đã được cài đặt vào chương trình, bạn có thể gõ lệnh:

```
> library()
```

Một lệnh quan trọng khác là khi cần trợ giúp bất kỳ từ khóa nào, bạn có thể gõ:

```
> help(plot)
```

hoặc

```
> ?plot
```

Chỉ dẫn sẽ hiện trong Tab Help thuộc màn hình đường dẫn bên phải phía dưới

1.2 Đối tượng trong R

Tất cả các thành phần trong R đều là đối tượng (kể cả hàm số). Đối tượng có thể coi như một khoảng bộ nhớ được gán cho một tên. Tất cả các đối tượng đều được lưu trữ trong bộ nhớ (RAM) của máy tính khi chạy R.

Hàm cũng chính là một loại đối tượng đặc biệt, được thiết kế để thực hiện một hoặc vài thao tác nào đó. Có rất nhiều hàm sẵn có trong R và bạn cũng có thể viết các hàm riêng của mình.

Một dữ liệu có thể được chứa trong một đối tượng biến bằng một phép gán, thường ký hiệu bởi dấu "<-".[1]

```
> n <- 1000
```

Sau khi gán, dữ liệu là số 1000 sẽ được lưu trong đối tượng (một phần bộ nhớ) biến có tên n. Để biết giá trị của đối tượng n, bạn gõ tên của nó và ấn ENTER:

```
> n
[1] 1000
```

Sau khi ấn ENTER, màn hình hiện lên giá trị của nó, nhưng đầu dòng có số thứ tự 1 đóng trong ngoặc vuông, cho biết dòng đó ghi giá trị kể từ phần tử thứ 1 của đối tượng. Trong ví dụ này đối tượng n chỉ có 1 phần tử nên ta chỉ có 1 dòng và 1 giá trị, tuy nhiên khi làm việc với vector có nhiều phần tử thì các bạn sẽ thấy ký hiệu này rất hữu ích.

Bạn có thể thực hiện các phép gán khác cho các đối tượng hoặc gán đè lên đối tượng cũ, hoặc dùng cửa sổ Console chỉ như một máy tính (thực hiện phép tính mà không gán cho đối tượng nào cả).

Thử một vài ví dụ sau:

```
> y <- 12
> y
[1] 12

> y <- -6
>y
[1] -6
```

[1] Các phiên bản sau của R cũng cho phép gán bằng dấu "=" nhưng theo thói quen người ta vẫn hay dùng dấu "<-". Trong tài liệu này tôi dùng dấu "<-".

```
> z <- 5
> w <- z^2
> w
[1] 25

> (3*2+45)/3
[1] 17
```

Sau khi tính toán, bạn có thể kiểm tra những đối tượng hiện có bằng lệnh ls() hoặc object():

```
> ls()
[1] "x" "y" "z" "w"
```

Bạn cũng có thể xóa khỏi bộ nhớ những biến không cần thiết:

```
> rm(y)
> rm(z,w)
```

Lưu ý là R phân biệt giữa chữ hoa và chữ thường, do vậy bạn cần chú ý biến Color khác biến color. Lỗi này thường rất hay xảy ra cho người mới lập trình.

1.3 Vector và vector hoá

Bây giờ chúng ta sẽ làm việc với đối tượng cơ bản nhất trong R là Vector. Vector là đối tượng dùng để chứa một tập hợp các phần tử có cùng kiểu dữ liệu. Các biến mà chúng ta vừa tạo ở trên cũng là những vector chỉ có 1 phần tử duy nhất.

Mỗi vector được đặc trưng bởi 2 đặc tính là *kiểu* (mode) và *độ dài* (length).

Các kiểu cơ bản trong R là:

- charactor: bao gồm cả ký tự và chuỗi ký tự
- logical: chỉ nhận giá trị T hoặc F hay TRUE hoặc FALSE
- numeric: kiểu số
- complex: kiểu số phức

Độ dài của vector là số phần tử trong nó, có thể kiểm tra bằng lệnh length(). Vector thường được tạo bởi lệnh combine, ký hiệu c() và liệt kê tất cả phần tử có trong vector:

```
> x <- c(2,5,1,7)
> x
[1] 2 5 1 7
```

Sau đó chúng ta có thể kiểm tra độ dài và kiểu của nó:

```
> length(x)
[1] 4
> mode(x)
[1] "numeric"
```

Trong ví dụ trên, R đã tự định dạng kiểu của x là kiểu numeric. Trong trường hợp bạn nhập kiểu dữ liệu khác nhau, R sẽ tìm cách gán một kiểu có thể[2]

```
> v <- c(4,3,7,"answer")
> v
[1] "4"      "3"      "7"      "answer"
```

Ở đây, R đã gán cho vector v kiểu charactor. 3 phần tử đầu tiên được hiểu là ký tự số.

Có một ký tự đặc biệt là "NA", có nghĩa là không có giá trị:

```
> u <- c(4, 6, NA, 2)
> u
[1] 4  6 NA 2
> k <- c(T, F, NA, TRUE)
> k
[1] TRUE FALSE NA TRUE
```

Ở đây R vẫn hiểu u là vector kiểu numeric với phần tử thứ 3 bị trống, còn k là vector kiểu logical với phần tử thứ 3 bị trống.

Bạn có thể truy vấn đến phần tử bất kỳ trong vector bằng cách gọi thứ tự trong dấu ngoặc vuông:

```
> v[4]
[1] "answer"
```

Bạn cũng có thể gán cho chỉ riêng một phần tử của vector một giá trị khác

```
> v[1] <- "hello"
> v
[1] "hello"  "3"      "7"      "answer"
```

hoặc gán thêm một phần tử mới vào vector đã có. Ta thử gán thêm phần tử thứ 5 vào vector v:

```
> v[5] <- 4
> v
[1] "hello"  "3"      "7"      "answer" "4"
```

Nếu bạn gán thêm vào một phần tử có số thứ tự cao hơn độ dài của vector hiện tại cộng 1 thì R tự hiểu các phần tử ở giữa là trống:

```
> v[8] <- "you"
> v
[1] "hello"  "3"      "7"      "answer" "4"      NA       NA
[8] "you"
```

[2] Trong trường hợp không tìm ra kiểu chung có thể, R sẽ báo lỗi

Cuối cùng, bạn hãy nhớ rằng với vector ta dùng dấu ngoặc vuông [], còn với hàm số ta dùng dấu ngoặc tròn.

Một trong những điểm mạnh nhất của R là vector hóa, có nghĩa là nó sẽ cho phép chúng ta áp một số hàm thông dụng lên một vector, giá trị trả về sẽ là một vector mà mỗi phần tử là giá trị hàm trên phần tử tương ứng của vector ban đầu:

```
> x <- c(2,6,3,5)
> y <- sqrt(x)
> y
[1] 1.414214 2.449490 1.732051 2.236068
```

Ở đây ta sử dụng hàm khai căn sqrt() trên vector dạng số x.

Ví dụ khác, bạn có thể dùng phép toán trên 2 vector có cùng độ dài

```
> x <- c(2,6,3,5)
> y <- c(1,2,5,2)
> x+y
[1] 3 8 8 7
```

hoặc giữa 1 vector và một giá trị

```
> z <- x*2
> z
[1]  4 12  6 10
```

Khi làm việc nhiều với dữ liệu bạn sẽ thường xuyên phải sử dụng công cụ này.

1.4 Tạo chuỗi (Vector dạng chuỗi)

Trong nhiều trường hợp xử lý dữ liệu ta cần tạo một vector phụ mô tả đặc tính của một vector/đối tượng nào đó. Các vector phụ này thường dùng để đánh số thứ tự, tạo biến giả, mô tả tính chu kỳ, tính ngẫu nhiên, ...

Ví dụ như ta có một tập hợp 100 cửa hàng, ta có thểtạo một vector ghi số thứ tự của chúng từ 1 đến 100 như sau:

```
> x <- 1:100
```

nếu bạn thay đổi 2 giá trị đầu và cuối thì cần để chúng trong ngoặc ():

```
> n <- 100
> x <- 1:(n-1)
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
[37] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
[73] 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
```

Bạn cũng có thể tạo chuỗi theo chiều giảm:

```
> x <- 100:1
```

Trong trường hợp bạn cần thay đổi hiệu số giữa 2 phần tử liên tiếp thì cần dùng hàm seq() (sequence) ở dạng tổng quát:

```
> seq(-4, 1, 0.5)
```

```
[1] -4.0 -3.5 -3.0 -2.5 -2.0 -1.5 -1.0 -0.5 0.0 0.5 1.0
```

Với hàm seq(), ta có thể quy định giá trị đầu, cuối và số phần tử, hàm sẽ tự tìm hiệu số

```
> seq(from = 1, to = 5, length = 4)
[1] 1.000000 2.333333 3.666667 5.000000
```

Một dạng hàm khác cũng rất hữu ích là rep() (viết tắt của replicate).

Ví dụ ta muốn khởi động một chỉ số tài chính cho 100 công ty. Trước khi tính toán ta có thể gán chỉ số bằng 1 cho tất cả các công ty[3]:

```
> x <- rep(1,100)
> x
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [54] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Hoặc gán biến tên khu vực cho tập hợp các công ty từ 1-30

```
> x <- rep("Nam", 30)
```

Trong nhiều trường hợp, ta lặp lại cả một vector con. Ví dụ ta tạo một chuỗi thời gian ghi tên các quý liên tiếp:

```
> x <- rep(1:4,10)
> x
 [1] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
```

Với x là chỉ số Quý trong năm của 10 năm liên tiếp. Hoặc cho thứ trong tuần:

```
> x <- rep(c("Mon","Tue","Wed","Thu","Fri","Sat","Sun"),length=365)
```

Ở đây ta lặp lại một vector ký tự, và cho biết tổng số chiều dài của vector chứ không phải số lần lặp lại.

```
> x
  [1] "Mon" "Tue" "Wed" "Thu" "Fri" "Sat" "Sun" "Mon" "Tue" "Wed" "Thu" "Fri" "Sat" "Sun" "Mon" "Tue" "Wed"
 [18] "Thu" "Fri" "Sat" "Sun" "Mon" "Tue" "Wed" "Thu" "Fri" "Sat" "Sun" "Mon" "Tue" "Wed" "Thu" "Fri" "Sat"
 [35] "Sun" "Mon" "Tue" "Wed" "Thu" "Fri" "Sat" "Sun" "Mon" "Tue" "Wed" "Thu" "Fri" "Sat" "Sun" "Mon" "Tue"
 [52] "Wed" "Thu" "Fri" "Sat" "Sun" "Mon" "Tue" "Wed" "Thu" "Fri" "Sat" "Sun" "Mon" "Tue" "Wed" "Thu" "Fri"
 [69] "Sat" "Sun" "Mon" "Tue" "Wed" "Thu" "Fri" "Sat" "Sun" "Mon" "Tue" "Wed" "Thu" "Fri" "Sat" "Sun" "Mon"
 [86] "Tue" "Wed" "Thu" "Fri" "Sat" "Sun" "Mon" "Tue" "Wed" "Thu" "Fri" "Sat" "Sun" "Mon" "Tue" "Wed" "Thu"
[103] "Fri" "Sat" "Sun" "Mon" "Tue" "Wed" "Thu" "Fri" "Sat" "Sun" "Mon" "Tue" "Wed" "Thu" "Fri" "Sat" "Sun"
...
```

1.5 Tập con

Chúng ta đã thấy có thể truy xuất một phần tử của vector thông qua ngoặc vuông. Chúng ta cũng có thể đưa một vector vào trong ngoặc vuông để trích xuất vector con.

Chúng ta khởi tạo một vector x, ví dụ:

```
> x <- c(0, 3, -4, 1, 25, 14, -5)
> x
[1]  0  3 -4  1 25 14 -5
```

[3] Nhiều chỉ số tài chính có giá trị dao động quanh 1, ví dụ PB, beta

sau đó truy xuất ra vector y các phần tử 1 và 4 của vector x:

```
> y <- x[c(1,4)]
> y
[1] 0 1
```

Ở đây ta đã đưa vector c(1,4) vào trong ngoặc vuông để chỉ thứ tự của phần tử x ta muốn trích xuất. Ví dụ khác:

```
> y <- x[1:3]
> y
[1]  0  3 -4
```

Chúng ta cũng có thể dùng một vector dạng logical để trích xuất. Vector logical này thường xuyên được tạo bởi điều kiện hóa vector ban đầu:

```
> y <- x>0
> y
[1] FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE
> z <- x[y]
> z
[1]  3  1 25 14
```

Ở đây y là một vector logical ghi lại phần tử của x có >0 hay không, và z là giá trị x trích xuất nếu y là TRUE. Cũng có thể viết ngắn gọn hơn như sau:

```
> z <- x[x>0]
> z
[1]  3  1 25 14
```

Cách viết sau này cũng rất phổ biến

Ngoài ra, ta cũng có thể loại bỏ một hoặc một số phần tử của vector ban đầu bằng cách thêm dấu '-' trước các chỉ số trong ngoặc vuông, ví dụ:

```
> y <- x[-3]
> y
[1]  0  3  1 25 14 -5
```

ví dụ trên ta gán cho y giá trị của x nhưng loại bớt phần tử thứ 3. Nếu cần loại nhiều hơn 1 phần tử:

```
> y <- x[-c(1,3)]
> y
[1]  3  1 25 14 -5
```

hoặc

```
> y <- x[-(1:3)]
> y
```

```
[1]  1 25 14 -5
```

Cuối cùng, chỉ số trong ngoặc vuông có thể để trống, khi đó ta hiểu là thực hiện với tất cả các phần tử của vector:

```
> x[] <- 0
> x
[1] 0 0 0 0 0 0 0
```

Ngoài vector, các đối tượng phổ biến khác là Ma trận, Mảng và Chuỗi.

1.6 Ma trận và mảng

Vector chứa dữ liệu ở trong một chiều. Khi cần chứa các dữ liệu nhiều chiều hơn, ta có thể sử dụng ma trận (2 chiều) hoặc mảng.

Có thể tạo một ma trận từ một vector bằng cách định nghĩa lại số chiều của ma trận như sau:

```
> x <- c(20,14,25,30,24,16,25,28)
> x
[1] 20 14 25 30 24 16 25 28
> dim(x) <- c(2,4)
> x
     [,1] [,2] [,3] [,4]
[1,]   20   25   24   25
[2,]   14   30   16   28
```

Ở đây ta đã định nghĩa lại chiều của x là 2 dòng và 4 cột. R tự hiểu x trở thành 1 ma trận.

Chúng ta cũng có thể định nghĩa trực tiếp ma trận x bằng hàm matrix() như sau:

```
> x <- matrix(c(20,14,25,30,24,16,25,28),2,4)
> x
     [,1] [,2] [,3] [,4]
[1,]   20   25   24   25
[2,]   14   30   16   28
```

Hàm matrix đã xếp vector vào thành một mảng 2 dòng và 4 cột, lần lượt theo từng cột.

Nếu muốn xếp theo từng dòng ta cần đặt thêm tham số cho hàm là byrow = T (T có nghĩa là TRUE)

```
> x <- matrix(c(20,14,25,30,24,16,25,28),2,4,byrow=T)
> x
     [,1] [,2] [,3] [,4]
[1,]   20   14   25   30
[2,]   24   16   25   28
```

Ma trận đã được lấp đầy theo từng dòng (dòng 1 rồi dòng 2).

Giả sử ma trận trên là lợi nhuận theo quý trong 2 năm của 1 công ty, ta có thể đặt tên lại các dòng và cột cho dễ hiểu:

```
> rownames(x) = c("2013","2014")
```

```
> colnames(x) = c("Q1","Q2","Q3","Q4")
> x
     Q1 Q2 Q3 Q4
2013 20 14 25 30
2014 24 16 25 28
```

Các thao tác truy xuất phần tử của mảng, tạo mảng con tương tự như cho vector. Bạn đọc có thể thử và tìm hiểu với các lệnh sau:

```
> x[1,2]
[1] 14
> x[1,]
Q1 Q2 Q3 Q4
20 14 25 30
> x[1,-3]
Q1 Q2 Q4
20 14 30
> x[1,-c(3,4)]
Q1 Q2
20 14
> x[1,-c(3,4),drop=F]
     Q1 Q2
2013 20 14
```

Riêng câu lệnh cuối ta thêm tham số drop = F để chỉ vẫn giữ nguyên dạng ma trận (trong khi các câu lệnh trước đó R đã tự chuyển kết quả về dạng vector).

Chúng ta cũng có thể ghép các ma trận theo dòng và cột bằng các lệnh cbind() (ghép 2 ma trận có cùng số dòng) hoặc rbind() (ghép 2 ma trận có cùng số cột):

```
> y <- matrix(c(89,93),2,1)
> colnames(y) <- c("Total")
> y
     Total
[1,]    89
[2,]    93
> z <- cbind(x,y)
> z
     Q1 Q2 Q3 Q4 Total
2013 20 14 25 30    89
2014 24 16 25 28    93
```

Ở đây tôi đã tính trước giá trị của biến y là tổng mỗi dòng của x. Nếu bạn muốn tính lại thì có thể ghép lệnh như sau:

```
> y <- matrix(rowSums(x,1),2,1)
```

Tương tự ta có thể ghép dòng như ví dụ sau:

```
> y <- matrix(c(14,10,20,22),1,4)
> rownames(y) <- c("2015")
> y
      [,1] [,2] [,3] [,4]
2015   14   10   20   22
> z <- rbind(x,y)
> z
     Q1 Q2 Q3 Q4
2013 20 14 25 30
2014 24 16 25 28
2015 14 10 20 22
```

Mảng là mở rộng của ma trận cho số chiều nhiều hơn 2. Thay vì hàm matrix() ta khai báo mảng bằng hàm array() với số chiều tương ứng. Ví dụ:

```
> a <- array(1:24, dim = c(4, 3, 2))
>a
,,1
[,1] [,2] [,3]
[1,]159
[2,] 2 6 10
[3,] 3 7 11
[4,] 4 8 12
,,2
[,1] [,2] [,3]
[1,] 13 17 21
[2,] 14 18 22
[3,] 15 19 23
[4,] 16 20 24
```

Các phần tử trong mảng (phần tử, vector hoặc mảng con) có thể được truy xuất tương tự:

```
> a[1, 3, 2]
[1] 21
>a[1,,2]
[1] 13 17 21
> a[4, 3, ]
[1] 12 24
> a[c(2, 3), , -2]
[,1] [,2] [,3]
[1,] 2 6 10
[2,] 3 7 11
```

1.7 Bảng dữ liệu (data.frame)

Bây giờ chúng ta sẽ làm việc với đối tượng quan trọng và phổ biến nhất trong R đó là bảng dữ liệu, trong R thường gọi là data.frame hoặc dataset. Cũng giống như ma trận, bảng dữ liệu là một dữ liệu 2 chiều nhưng cái khác là: với ma trận tất cả các phần tử có cùng một kiểu dữ liệu trong khi với bảng dữ liệu, chỉ các phần tử trong cùng một cột mới có cùng kiểu dữ liệu.

Mỗi cột được gọi là một biến số của bảng dữ liệu và có thể có kiểu dữ liệu khác nhau. Mỗi dòng được coi như là một quan sát hoặc một trường hợp ("case"). Do vậy bảng dữ liệu chính là dạng dữ liệu thường thấy trên một bảng tính (Excel) mà đa số chúng ta vẫn thường gặp. Mỗi bảng dữ liệu tương đương một "Sheet" (dữ liệu) trong một file Excel – hình dung như vậy sẽ dễ hiểu hơn nhiều. Lưu ý là R cũng cho phép chứa dữ liệu trống trong bảng dữ liệu giống như trong vector.

Bạn đọc có thể tự tạo một bảng dữ liệu bằng cách nhập tay như hàm c() cho vector, nhưng thay bằng hàm data.frame() như sau:

```
> bang.dulieu <- data.frame(ten = c("A","B","C"), von = c(100,150,120), loinhuan = c(10,5,15))
> bang.dulieu
  ten von loinhuan
1   A 100       10
2   B 150        5
3   C 120       15
```

Ta đã tạo một bảng dữ liệu tên bang.dulieu bằng cách liệt kê tên và giá trị của mỗi cột. Tên của mỗi biến/cột được định dạng dưới dạng ký tự là "ten","von" và "loinhuan". Giá trị mỗi biến là một vector được định nghĩa ngay trong câu lệnh bởi hàm c().

Bây giờ ta có thể truy xuất đến phần tử hoặc biến như sau:

```
> bang.dulieu[2,3]          # công ty thứ 2, biến lợi nhuận
[1] 5
> bang.dulieu[,3]           # biến lợi nhuận của tất cả công ty
[1] 10  5 15
> bang.dulieu[1,]           # các biến của công ty thứ nhất
  ten von loinhuan
1   A 100       10
```

Bây giờ để minh hoạ rõ nét hơn, chúng ta sử dụng cách nhập dữ liệu phổ biến hơn là nhập từ file qua giao diện (cách nhập file qua câu lệnh sẽ được thực hành ở chương sau). Trước hết bạn cần tải file "company_list.csv" từ trang web cophieu68.com hoặc từ đây, đặt trong thư mục của dự án.

Bạn "Import Dataset" trong cửa sổ Environment, chọn import từ "tex file", chọn file company_list.csv nói trên, đặt "Header" là Y và Import.

Sau khi hoàn thành bạn có thể thấy bảng dữ liệu hiện trên cửa sổ Environment với 1017 doanh nghiệp và 28 variables (biến số hay chỉ số của công ty). Trong chương này để làm quen với bảng dữ liệu ta tạm thời chỉ dùng 5 biến đầu tiên. Chúng ta truy xuất 5 biến đầu và gán vào bảng cũ như sau:

```
> company_list <- company_list[,1:5]
```

Để kiểm tra lại số biến ta có thể dùng lệnh dim()

```
> dim(company_list)
```

```
[1] 1017    5
```

Chúng ta hãy thử quan sát 10 công ty đầu tiên

```
> company_list[1:10,]
```

```
   MaCK SanCK GiaHienTai GiaSoSach  P.B
1   AAA HASTC       16.5      28.6 0.58
2   AAM  HOSE       15.0      24.6 0.61
3   ABT  HOSE       51.0      36.1 1.41
4   ACB HASTC       15.2      13.1 1.16
5   ACC  HOSE       26.8      11.2 2.39
6   ACL  HOSE       12.4      15.1 0.82
7   ADC HASTC       24.5      15.0 1.63
8   AGF  HOSE       22.4      33.1 0.68
9   AGM  HOSE       12.3      19.1 0.64
10  AGR  HOSE        7.4      10.7 0.69
```

5 biến hiện có là "MaCK", "SanCK", "GiaHienTai", "GiaSoSach", "P.B". Giả sử ta chỉ cần tìm hiểu biến Giá hiện tại (biến GiaHienTai) bạn có thể lưu thành một vector mới:

```
giaHT <- company_list$GiaHienTai
```

Bạn cũng có thể tạo một bảng khác với ít biến hơn, ví dụ ta chỉ quan tâm "MaCK", "GiaHienTai", "GiaSoSach":

```
> company_list_2 <- company_list[,c("MaCK","GiaHienTai","GiaSoSach")]
```

hoặc bằng cách viết tắt các biến theo thứ tự của chúng

```
> company_list_2 <- company_list[,c(1,3,4)]
```

Chúng ta đã thấy với vector, ma trận, mảng hay bảng, nếu bỏ trống chỉ số của một chiều nào đó (hàng hay cột hay chiều lớn hơn với mảng) thì ta mặc định là chọn tất cả các phần tử của chiều đó. Tuy nhiên chúng ta cũng có thể lựa chọn các quan sát thoả mãn điều kiện nào đó bằng cách thay thế chỉ số bằng điều kiện:

```
> company_list_3 <- company_list[company_list$GiaHienTai >= 10,c(1,3,4)]
```

Trong câu lệnh trên ta tạo bảng với 3 biến "MaCK", "GiaHienTai", "GiaSoSach" nhưng chỉ chọn những công ty có giá hiện tại lớn hơn 10.

Cuối cùng, bên cạnh lệnh dim() nói trên cho ta số dòng và cột của bảng, ta cũng có thể tìm dòng và cột bằng lệnh riêng lẻ nrow() và ncol(). Các lệnh này rất hay dùng khi ta cần tạo vòng lặp:

```
> nrow(company_list_3)              # Hiển thị số dòng
[1] 342
> n=nrow(company_list_3)            # Lưu số dòng vào biến n
> n
[1] 342
> m = ncol(company_list_3)          # Lưu số cột vào biến n
> m
[1] 3
```

Cuối cùng, trong R cũng có rất nhiều bộ dữ liệu đi kèm, chúng ta có thể khám phá chúng sư sau:

Liệt kê tất cả các bộ dữ liệu đang có trong R

```
> data()
```

Tải bộ dữ liệu "mtcars" vào bộ nhớ

```
> data(mtcars)
```

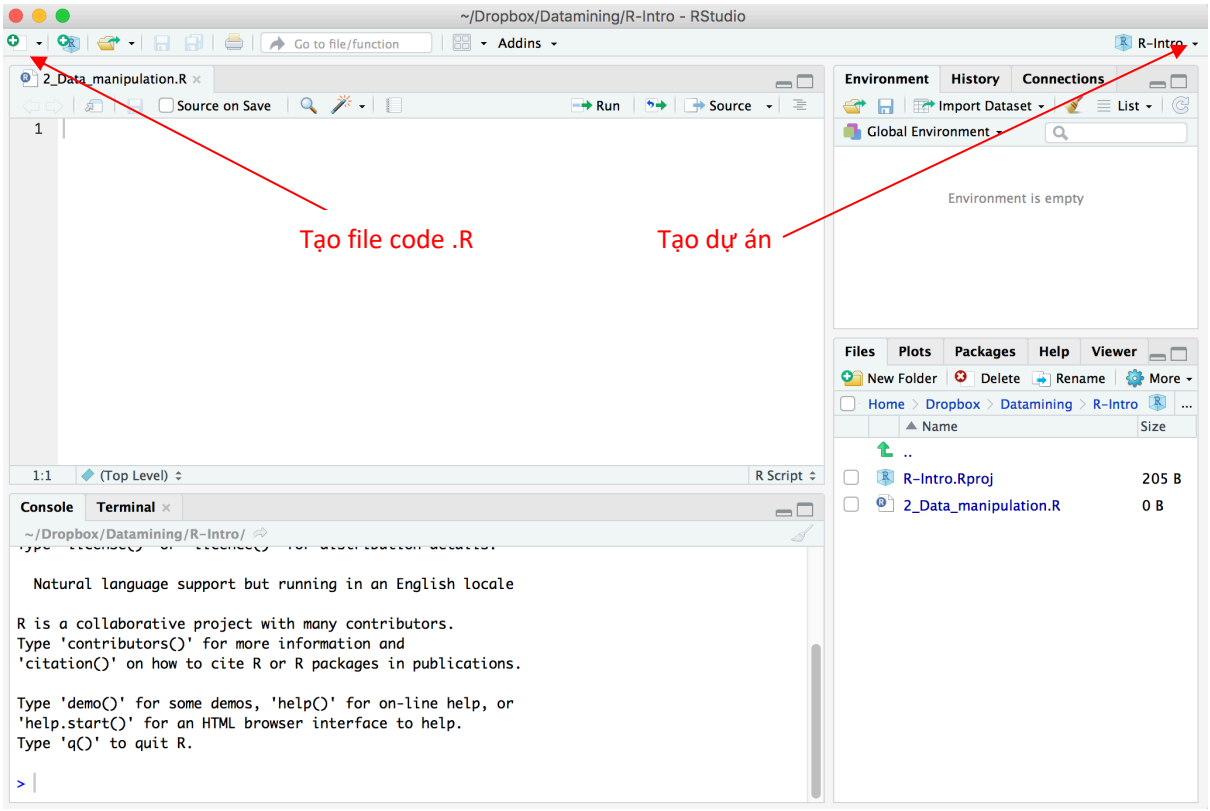
Bộ dữ liệu "mtcars" đã được tải vào bộ nhớ thành một bảng dữ liệu/dataframe cùng tên.

Sau khi học xong chương này coi như bạn đã nắm phần cơ bản nhất của công cụ R. 2 chương sau sẽ mô tả các ví dụ thực tế để các bạn hiểu rõ hơn các thành phần và đối tượng chính của R. Chương cuối sẽ nói thêm về các chức năng nâng cao của R.

2 Thao tác dữ liệu với R

Trong chương này và chương sau tôi sử dụng các bộ dữ liệu từ trang hỗ trợ của cuốn sách "Data Analysis Using SQL and Excel" của tác giả Gordon S. Linoff, xuất bản bởi John Wiley & Sons năm 2008. Thư mục dữ liệu bao gồm 7 bảng, có thể tải ở đây.

Các ví dụ sau được thực hành với Rstudio đã được cài đặt như hướng dẫn ở phần trước. Thông thường cho mỗi chương trình chúng ta tạo một Dự án ("Project"), ở đây tôi đặt tên "R-Intro" và tạo file thực hành "2_Data_manipulation.R" cho mục 2 như sau:



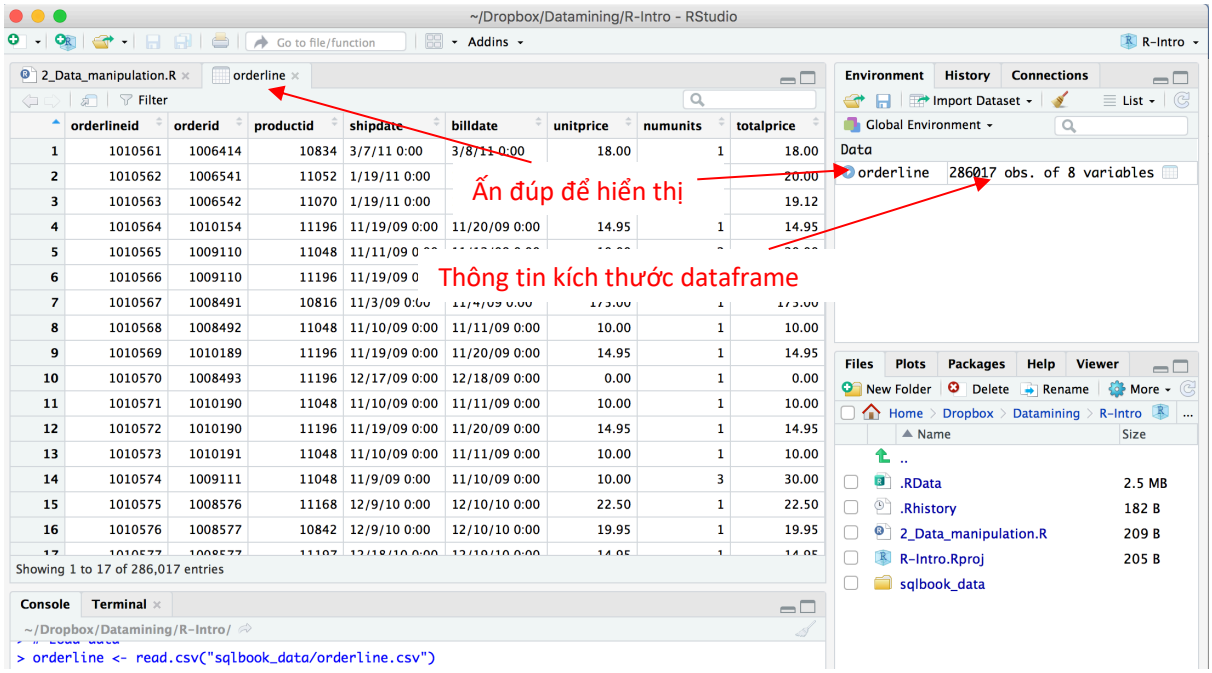
2.1 Nhập/xuất bảng dữ liệu

Chúng ta sẽ nhập bảng orderline là bảng ghi các giao dịch ứng với mỗi lượt sản phẩm được mua. Để minh hoạ tôi đã chuyển đổi file .txt của tác giả Linoff thành file .csv, là định dạng gần với file Excel phổ biến hơn trong Doanh nghiệp. Các file .csv có thể tải ở đây.

Giả sử các bảng dữ liệu đã được lưu ở máy trong thư mục "/sqlbook_data" nằm trong thư mục của Dự án đã tạo. Chúng ta đọc file orderline.csv bằng lệnh read.csv() và gán vào dataframe cùng tên orderline.

```
orderline <- read.csv("sqlbook_data/orderline.csv")
```

Kích thước bảng orderline được hiển thị trong khung Môi trường bên phải phía trên, ta thấy có 286,017 dòng và 8 biến. Có thể click vào đối tượng để hiển thị các ô đầu tiên



Hình 2: Bảng dữ liệu orderline đã được tải vào R

Ngoài ra chúng ta có thể sử dụng các lệnh tổng hợp bảng như sau (có thể viết trong file "2_Data_manipulation.R" hoặc viết dưới cửa sổ Console:

Kích thước bảng - số dòng và cột: (chú thích ở phần comment phía sau)

```
> dim(orderline)                # Tính số dòng và cột của bảng

[1] 286017      8

> nrow(orderline)     # Tính số dòng của bảng

[1] 286017

> ncol(orderline)      # Tính số cột của bảng

[1] 8
```

Sau khi biết kích thước bảng chúng ta có thể xem sơ bộ một vài dòng dữ liệu đầu hoặc cuối nhờ các lệnh sau:

```
> head(orderline)               # Hiển thị các dòng đầu tiên của bảng
  orderlineid orderid productid      shipdate      billdate unitprice numunits totalprice
1     1010561 1006414     10834   3/7/11 0:00   3/8/11 0:00     18.00        1      18.00
2     1010562 1006541     11052  1/19/11 0:00  1/20/11 0:00     10.00        2      20.00
3     1010563 1006542     11070  1/19/11 0:00  1/20/11 0:00     19.12        1      19.12
4     1010564 1010154     11196 11/19/09 0:00 11/20/09 0:00     14.95        1      14.95
5     1010565 1009110     11048 11/11/09 0:00 11/12/09 0:00     10.00        2      20.00
```

6 1010566 1009110 11196 11/19/09 0:00 11/20/09 0:00 14.95 2 29.90

Nếu cần xác định rõ số dòng đầu tiên cần xem, ta thêm tham số:
head(orderline,20)

Xem những dòng cuối cùng tương tự với lệnh tail()

tail(orderline)

Với các lệnh head() và tail() chúng ta đã có thể xem các bảng chứa những trường gì. Tuy nhiên nếu số trường quá nhiều thì chúng ta có thể liệt kê tên trường như sau:

names(orderline)

- hoặc

colnames(orderline)

Tiếp theo chúng ta có thể khảo sát cấu trúc dữ liệu, kiểu dữ liệu, của mỗi trường bằng lệnh str()

str(orderline)

Một lệnh tổng hợp khác là summary() cho phép thống kê nhanh tất cả các trường dữ liệu của bảng:

summary(orderline)

Các lệnh str() và summary() rất tiện dụng cho việc xây dựng Từ điển dữ liệu ("Data Dictinary")

Chúng ta có thể nhập tiếp các bảng khác trong bộ dữ liệu vào các dataframe tương ứng

orders <- read.csv("sqlbook_data/orders.csv")

customer <- read.csv("sqlbook_data/orderline.csv")

product <- read.csv("sqlbook_data/orderline.csv")

và có thể khả sát các bảng trên bằng những lệnh tổng hợp tương tự như với bảng orderline.

Ở chiều ngược lại, chúng ta có thể xuất dữ liệu ra file .csv bằng lệnh write.csv()

Ví du:

write.csv(orderline, file = "orderline.csv")

2.2 Thao tác trên bảng dữ liệu

1. Chuẩn bị dữ liệu

Sau khi đã nhập các bảng vào R, để phân tích dữ liệu chúng ta thường phải thực hiện một vài biến đổi cơ bản. Sau đây là những ví dụ phổ biến

* Từ lệnh str(orderline) và summary(orderline) có thể thấy 2 trường shipdate và billdate được đọc dưới dạng văn bản. Chúng ta nên đổi sang dạng ngày của R

orderline$billdate <- as.Date(orderline$billdate, format = "%m/%d/%y %H:%M")

orderline$shipdate <- as.Date(orderline$shipdate, format = "%m/%d/%y %H:%M")

với tham số format được viết theo định dạng của dữ liệu đọc vào (cần thay đổi cho phù hợp với định dạng khác). Sau khi biến đổi chúng ta có thể kiểm tra lại bằng lệnh:

summary(orderline)

Chúng ta thấy 2 trường ngày đã được chuyển đúng và thống kê cho thấy ngày giao dịch billdate bắt đầu từ 22/10/2009 đến ngày 21/09/2016.

* Các bảng dữ liệu thu thập từ bộ phận công nghệ thường được tách riêng thành cho các đối tượng chính như phần trước đã minh họa (giao dịch, lệnh, khách hàng, sản phẩm,...). Cách thiết kế bảng như vậy rất thuận tiện cho việc vận hành nhanh của hệ thống. Tuy nhiên, việc phân tích kinh doanh lại thường diễn ra với góc nhìn đa chiều, người phân tích muốn khảo sát tất cả các yếu tố có thể ảnh hưởng đến kết quả kinh do-

anh. Do vậy công việc đầu tiên thường làm là ghép nối các bảng lại để có được các trường cần thiết trong phân tích. Trong Excel việc này được thực hiện bằng hàm thông dụng VLOOKUP(). Trong R chúng ta dùng hàm merge(), ví dụ cho các bảng orderline và orders đã tải trước đó như sau (cũng có thể thực hiện cho các bảng có dung lượng đến GB khác):

```
ordercomplete <- merge(orderline, orders, by.x = "orderid", by.y = "orderid", all.x = TRUE)
```

Câu lệnh xác định 2 bảng, trường nối giữa 2 bảng là cùng một biến số kinh doanh (mã đơn hàng orderid), luật ghép là giữ lại tất cả các dòng của bảng bên trái. Sau khi ghép ta có thể kiểm tra bằng lệnh

```
head(ordercomplete)
```

Chúng ta nhận thấy 2 trường numunits và totalprice có ở cả 2 bảng nên khi ghép lại chúng bị lặp lại. Ta có thể loại bỏ 2 trường này ở bảng orders khi ghép như sau:

```
# neglect 2 columns numunits and total price
ordercomplete <- merge(orderline, orders[,!(colnames(orders) %in% c("numunits","totalprice"))],
                       by.x = "orderid", by.y = "orderid", all.x = TRUE)
```

ở đây ta đã sử dụng lệnh lọc các cột của bảng orders

và kiểm tra lại

```
head(ordercomplete)
```

chúng ta thấy trường orderdate của bảng orders mới được ghép lại có định dạng văn bản, ta chuyển sang dạng này như sau:

```
ordercomplete$orderdate <- as.Date(ordercomplete$orderdate, format = "%m/%d/%y")
```

sau khi bảng ordercomplete đã hoàn thành chúng ta đã có thể thực hiện rất nhiều phân tích kinh doanh.

* Lọc và sắp xếp

2. Tìm kiếm và truy vấn (tương ứng hàm MATCH, INDEX, OFFSET)

Sau khi có được dữ liệu ban đầu và những thống kê cơ bản, người phân tích có thể chỉ quan tâm đến một phần dữ liệu con. Các lệnh này thường được sử dụng bởi các hàm MATCH(), INDEX(), OFFSET() trong Excel. Dưới đây là một số ví dụ tìm kiếm và truy vấn ra các tập con của một bảng dữ liệu trong R.

- Lựa chọn một số cột xác định:

```
data <- ordercomplete[,c("orderid", "orderlineid", "productid", "customerid")]
```

ở đây chúng ta truy vấn đến tập con của bảng được chỉ ra bởi các chỉ số trong dấu [,]. Các chỉ số dòng được viết trước dấu ",", chỉ số cột viết sau dấu ",". Lệnh trên để trống chỉ số dòng có nghĩa là lấy tất cả các dòng, trong khi chỉ chọn ra 4 cột được lệt kê bằng một hàm vector c().

- Lựa chọn một số dòng xác định

```
data <- ordercomplete[10001:11000,]
```

- Lựa chọn theo điều kiện của một hoặc một số cột

```
data <- ordercomplete[ordercomplete$orderid == "1055595",]  # Lấy các giao dịch của KH mã 1055595
```

ở đây chúng ta đã thực hiện một truy vấn con bằng cách lấy ra cột orderid của bảng ordercomplete, được nối bằng dấu "$". Đây là một cách truy vấn khác nếu chỉ truy vấn 1 cột duy nhất. Nếu dùng cách truy vấn tổng quát hơn ở trên thì cần viết như sau (và kết quả là tương đương):

```
data <- ordercomplete[ordercomplete[,c("orderid")] == "1055595",]
```

- Ví dụ khác

```
data <- ordercomplete[ordercomplete$state== "NY",]   # Lấy các giao dịch từ bang NY
data <- ordercomplete[ordercomplete$ paymenttype== "AE",]
# Lấy các giao dịch dùng kiểu thanh toán AE
data <- ordercomplete[ordercomplete$totalprice > 1000,]          # Lấy các giao dịch có giá trị > 1000
data <- ordercomplete[ordercomplete$orderdate >= as.Date("2016-01-01"),]
# Lấy các giao dịch từ ngày 01/01/2016
data <- ordercomplete[is.na(ordercomplete$unitprice),]
# Lấy các giao dịch không có giá đơn vị (để kiểm tra)
data <- ordercomplete[ordercomplete$state== "NY" & ordercomplete$ paymenttype== "AE",]
# Lấy bởi nhiều điều kiện
data <- ordercomplete[ordercomplete$state== "NY" & ordercomplete$ paymenttype== "AE",c("productid","customerid","totalprice")]
```

Trong một số trường hợp chúng ta chỉ cần truy vấn dữ liệu của một biến duy nhất, chúng ta có thể lấy biến trước rồi truy vấn sau, hoặc truy vấn trước rồi chọn biến sau, ví dụ:

```
# Lấy biến totalprice rồi truy vấn
data <- ordercomplete$totalprice[ordercomplete$state== "NY" & ordercomplete$ paymenttype== "AE"]
# Truy vấn ra bộ dữ liệu con và chọn biến totalprice
data <- ordercomplet[ordercomplete$state== "NY" & ordercomplete$ paymenttype== "AE",c("totalprce")]
```

Hoặc truy vấn ra bộ dữ liệu con trước rồi chọn biến totalprice sau đó

```
data <- ordercomplete[ordercomplete$state== "NY" & ordercomplete$paymenttype== "AE",][,c("totalprice")]
```

3 câu lệnh trên cho kết quả tương đương.

3. Thống kê nhanh

Sau khi đã truy vấn ra các dữ liệu cần quan tâm, người dùng có thể thực hiện những hàm thống kê nhanh như sau:

```
sum(ordercomplete$totalprice[ordercomplete$state== "NY" & ordercomplete$ paymenttype== "AE"])
# Tính tổng giá trị bán hàng tại bang NY và trả bằng phương thức AE (tương đương lệnh SUMIF() trong Excel)
mean(ordercomplete$totalprice[ordercomplete$state== "NY" & ordercomplete$ paymenttype== "AE"])
# Tính giá trị trung bình giao dịch tại bang NY và trả bằng phương thức AE (tương đương lệnh AVERAGEIF() trong Excel)
length(ordercomplete$totalprice[ordercomplete$state== "NY" & ordercomplete$ paymenttype== "AE"])
# Tính số lượng giao dịch tại bang NY và trả bằng phương thức AE (tương đương lệnh COUNT() trong Excel)
```

Với lệnh length(), có thể thấy R chỉ đếm số dòng thỏa mãn các điều kiện lọc, có nghĩa là nếu thay biến totalprice bằng một biến khác như productid, kết quả cũng cho giống nhau là số giao dịch:

```
length(ordercomplete$productid[ordercomplete$state== "NY" & ordercomplete$paymenttype== "AE"])
```

Tuy nhiên, lệnh length có thể được sử dụng ghép với lệnh unique(), lọc ra các giá trị duy nhất, để đếm số kết quả duy nhất. Ví dụ lệnh sau tính số các sản phẩm được mua ở bang NY và trả bằng phương thức AE:

```
length(unique(ordercomplete$productid[ordercomplete$state== "NY" & ordercomplete$paymenttype== "AE"]))
# Có 1788 sản phẩm được mua ở bang NY theo phương thức AE
```

Lệnh trên tương đương với hàm COUNTDISTINCT() trong Excel

2.3 Thống kê và tổng hợp dữ liệu

Các thống kê nhanh ở phần trên cho ta các số liệu duy nhất và đưa ra những đánh giá tổng quan ban đầu. Để phân tích chi tiết hơn các yếu tố kinh doanh chúng ta có thể thực hiện các thống kê tổng hợp. Trước hết chúng ta sẽ phân loại các yếu tố kinh doanh, các biến số trong ngôn ngữ kỹ thuật, thành 2 loại:

- Các biến liên tục nếu biến số nhận số giá trị bất kỳ trong một phạm vi nào đó (và có thể vô cùng lớn): Ví dụ giá đơn vị, số tuổi, tiền lương, doanh số, số lượng sản phẩm bán,…
- Các biến phân loại: chỉ nhận một số lượng ít các giá trị: Ví dụ giới tính, địa bàn (tỉnh/thành phố), ngành sản phẩm, phương thức thanh toán,…

Chúng ta sẽ lần lượt phân tích các biến, cặp biến theo từng loại như sau:

1. TK mô tả 1 biến phân loại

Với biến phân loại, thông thường chúng ta cần đếm số lượng mẫu theo từng giá trị của biến. Ví dụ với biến paymentype, chúng ta đếm số giao dịch theo loại payment như sau:

```
> table(ordercomplete$paymenttype)

          ??      AE      DB      MC      OC      VI
 250     390   72989   17869   70934   10637  112948
```

Chúng ta thấy phần lớn giao dịch dùng phương thức VI (có thể là loại thẻ Visa), có 250 giao dịch để trống (không ký tự) và 390 giao dịch có phương thức "??". Để phân tích rõ hơn chúng ta có thể tính tỷ lệ phần trăm bằng lệnh prop.table()

```
> prop.table(table(ordercomplete$paymenttype))

                    ??           AE           DB           MC           OC           VI
0.0008740739 0.0013635553 0.2551911250 0.0624753074 0.2480062374 0.0371900971 0.3948996039
```

Và biến đổi dạng số thành dạng làm tròn:

```
> format(prop.table(table(ordercomplete$paymenttype)), digit = 3)

                ??          AE          DB          MC          OC          VI
"0.000874" "0.001364" "0.255191" "0.062475" "0.248006" "0.037190" "0.394900"
```

Chúng ta có thể đưa cả 2 thống kê này vào thành chung một dataframe, sau đó xắp xếp theo giá trị, đưa thêm cột cộng tích lũy của tỷ lệ phần trăm bằng các lệnh sau:

```
> paymentype_summary <- data.frame(paymentype = table(ordercomplete$paymenttype)) # Tạo dataframe từ bảng thống kê
> colnames(paymentype_summary) <- c("paymentype", "frequency")  # Đổi lại tên cột
> paymentype_summary$percentage <- prop.table(table(ordercomplete$paymenttype))
> paymentype_summary <- paymentype_summary[order(paymentype_summary$frequency, decreasing = TRUE),]
> paymentype_summary$cumpercentage <- cumsum(paymentype_summary$percentage)
> paymentype_summary
  paymentype frequency   percentage cumpercentage
7         VI    112948 0.3948996039     0.3948996
3         AE     72989 0.2551911250     0.6500907
5         MC     70934 0.2480062374     0.8980970
4         DB     17869 0.0624753074     0.9605723
6         OC     10637 0.0371900971     0.9977624
2         ??       390 0.0013635553     0.9991259
1                  250 0.0008740739     1.0000000
```

Và có thể xuất bảng thống kê trên thành file csv:

```
> write.csv(paymentype_summary, file = "paymentype_summary.csv")
```

(Các thống kê trên với biến phân loại có thể được viết gọn thành một hàm duy nhất, chúng ta sẽ quay lại trong phần 4)

2. TK mô tả 2 biến phân loại

Với 2 biến phân loại, chúng ta có thể chọn 2 biến paymentype và state. Tuy nhiên do số lượng state nhiều, tôi nhóm những state có số giao dịch ít (dưới 1000) vào nhóm "OTHERS" bằng cách chèn vào một biến số mới state_new như sau:

```
> tmp <- table(ordercomplete$state)
> ordercomplete$state_new <- ifelse(ordercomplete$state %in% names(tmp[tmp > 1000]),
                    as.character(ordercomplete$state), "OTHERS")
```

Chúng ta có thể thống kê biến mới với lệnh table()

```
> table(ordercomplete$state_new)
```

và thống kê 2 biến phân loại như sau:

```
> table(ordercomplete$state_new,ordercomplete$paymenttype)
```

```
        ??   AE   DB   MC   OC   VI
    250  4  360   91  469    0  826
AZ    0  6  719  136  788  106 1696
CA    0 46 5067 1026 6724  650 12919
CO    0  1  572  112  588  137 1745
CT    0 18 3467  890 3875  472 5022
DC    0  0 1019  136  622   34 1260
DE    0  1  154   73  223   48  511
FL    0 29 5003  568 3296  678 5621
```

Chúng ta có thể thấy 250 giao dịch có kiểu thanh toán trống cũng là những giao dịch không có tên bang. Chúng ta có thể thống kê theo tỷ lệ bởi lệnh prop.table(). Tuy nhiên với 2 biến ta cần chỉ rõ tỷ lệ được xác định trên biến nào:

```
> prop.table(table(ordercomplete$state_new,ordercomplete$paymenttype),1)
```

```
                ??          AE          DB          MC          OC          VI
    0.1250000000 0.0020000000 0.1800000000 0.0455000000 0.2345000000 0.0000000000 0.4130000000
AZ  0.0000000000 0.0017386265 0.2083454071 0.0394088670 0.2283396117 0.0307157346 0.4914517531
CA  0.0000000000 0.0017403148 0.1916994552 0.0388165860 0.2543886199 0.0245914044 0.4887636199
CO  0.0000000000 0.0003169572 0.1812995246 0.0354992076 0.1863708399 0.0434231379 0.5530903328
CT  0.0000000000 0.0013096624 0.2522555297 0.0647555297 0.2819412107 0.0343422584 0.3653958091
DC  0.0000000000 0.0000000000 0.3318137415 0.0442852491 0.2025398893 0.0110713123 0.4102898079
DE  0.0000000000 0.0009900990 0.1524752475 0.0722772277 0.2207920792 0.0475247525 0.5059405941
FL  0.0000000000 0.0019085225 0.3292530438 0.0373807173 0.2169134584 0.0446199408 0.3699243172
```

Lệnh trên tính tỷ lệ theo dòng tương ứng biến state_new: các giá trị cho biết tỷ lệ giao dịch theo từng loại thanh toán của từng bang

Nếu đổi tham số thành 2, chúng ta có tỷ lệ giao dịch theo từng bang của từng loại thanh toán

```
> prop.table(table(ordercomplete$state_new,ordercomplete$paymenttype),2)
```

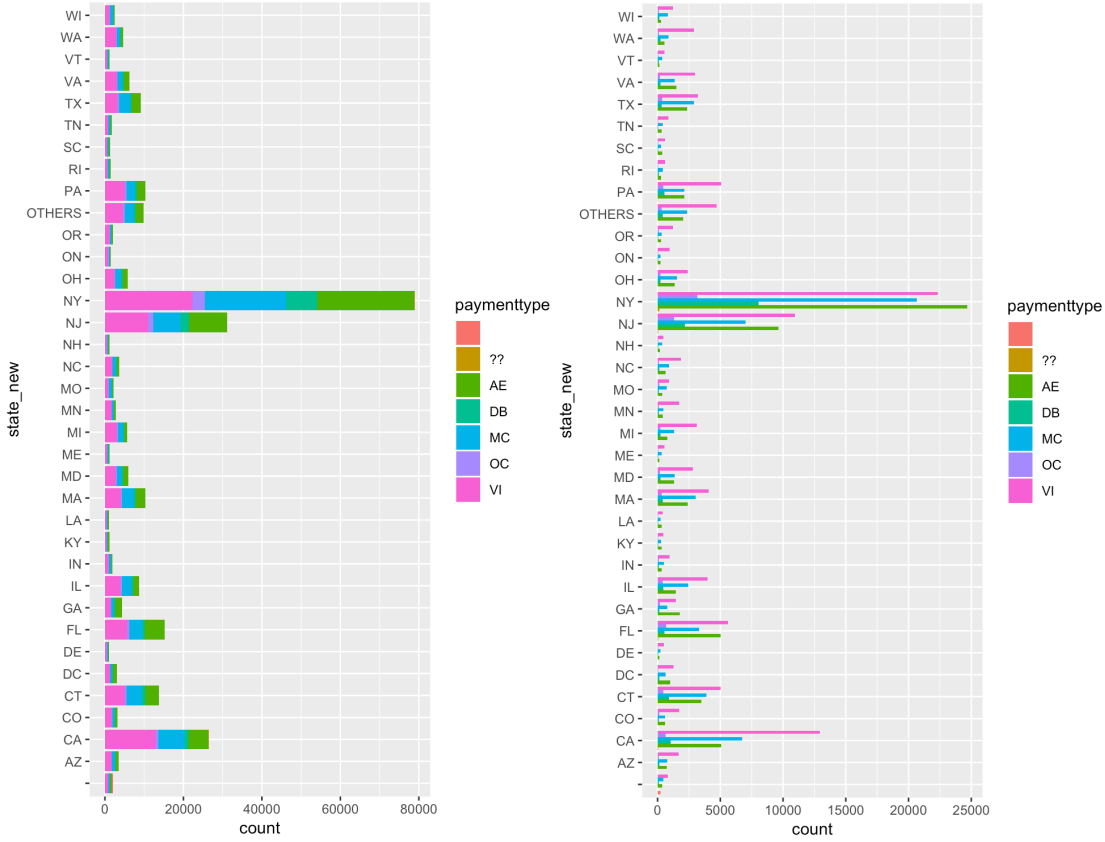
```
                ??          AE          DB          MC          OC          VI
    1.000000000 0.010256410 0.004932250 0.005092619 0.006611780 0.000000000 0.007313100
AZ  0.000000000 0.015384615 0.009850799 0.007610946 0.011108918 0.009965216 0.015015759
CA  0.000000000 0.117948718 0.069421420 0.057417875 0.094792342 0.061107455 0.114380069
CO  0.000000000 0.002564103 0.007836797 0.006267838 0.008289396 0.012879571 0.015449587
CT  0.000000000 0.046153846 0.047500308 0.049806928 0.054628246 0.044373414 0.044462939
DC  0.000000000 0.000000000 0.013961008 0.007610946 0.008768715 0.003196390 0.011155576
DE  0.000000000 0.002564103 0.002109907 0.004085287 0.003143767 0.004512551 0.004524206
FL  0.000000000 0.074358974 0.068544575 0.031786894 0.046465729 0.063739776 0.049766264
```

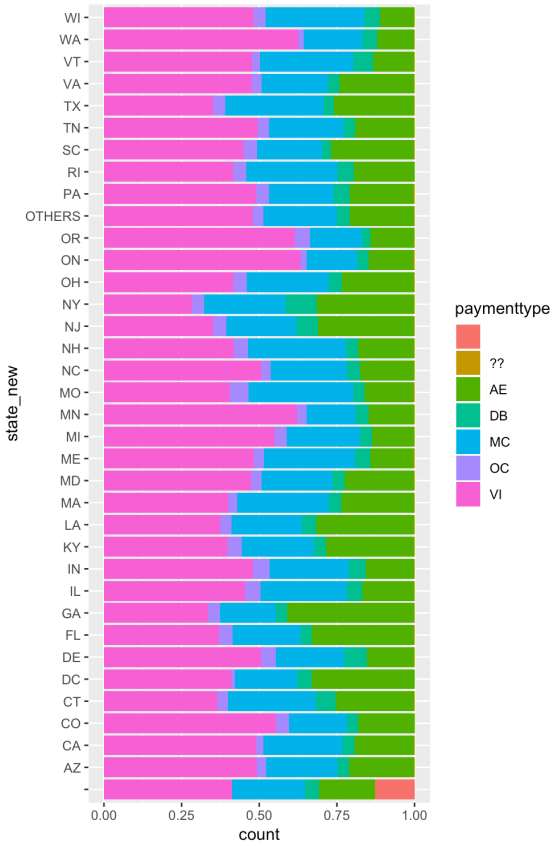
Với 2 biến phân loại, việc biểu diễn bằng đồ thị sẽ giúp phân tích dễ hơn. Chúng ta sử dụng thư viện đồ họa phổ biến "ggplot2" bằng lệnh gọi library(), với điều kiện đã cài đặt gói này như hướng dẫn trong mục 1.1:

```
library(ggplot2)
ggplot(ordercomplete,aes(state_new,fill=paymenttype)) + geom_bar() + coord_flip()
ggplot(ordercomplete,aes(state_new,fill=paymenttype)) + geom_bar(position="dodge") + coord_flip()
ggplot(ordercomplete,aes(state_new,fill=paymenttype)) + geom_bar(position="fill") + coord_flip()
```

Cú pháp của các câu lệnh của gói "ggplot2" bao gồm các lớp cộng vào nhau: lớp ggplot chứa các tham số là nguồn dữ liệu (dataframe), biến được chọn cho thống kê X và Y); lớp geom_bar() xác định kiểu đồ thị cột và lớp coord_flip() cho phép quay đồ thị dạng dọc. Các cú pháp chi tiết của gói "ggplot2" có thể tìm ở tài liệu tóm tắt sau.

Các đồ thị của 3 câu lệnh trên được biểu diễn trong hình 3.





Hình 3: Đồ thị thống kê cho 2 biến phân loại: Dạng (a) được dùng để đánh giá tổng số giao dịch của mỗi bang và biết thêm thành phần kiểu thanh toán trong giao dịch của mỗi bang. Có thể thấy các bang NY, NJ, CA, FL là các bang có nhiều giao dịch nhất. Dạng (b) cho chúng ta thấy độ lớn của tổng số giao dịch của mỗi bang theo mỗi kiểu thanh toán, có thể so sánh từng loại giao dịch của các bang khác nhau. Dạng (c) cho biết tỷ lệ của mỗi loại giao dịch của mỗi bang. Dạng này cho phép so sánh tỷ lệ dễ hơn nhưng thiếu con số tuyệt đối.

3. TK mô tả 1 biến liên tục

Với biến liên tục, các thông kê từ lệnh summary() mà chúng ta đã chạy cho cả bộ dữ liệu cũng có thể chạy được với biến duy nhất cho biết các chỉ số quan trọng của giá trị biến (lấy ví dụ biến totalprice):

```
summary(ordercomplete$totalprice)
```

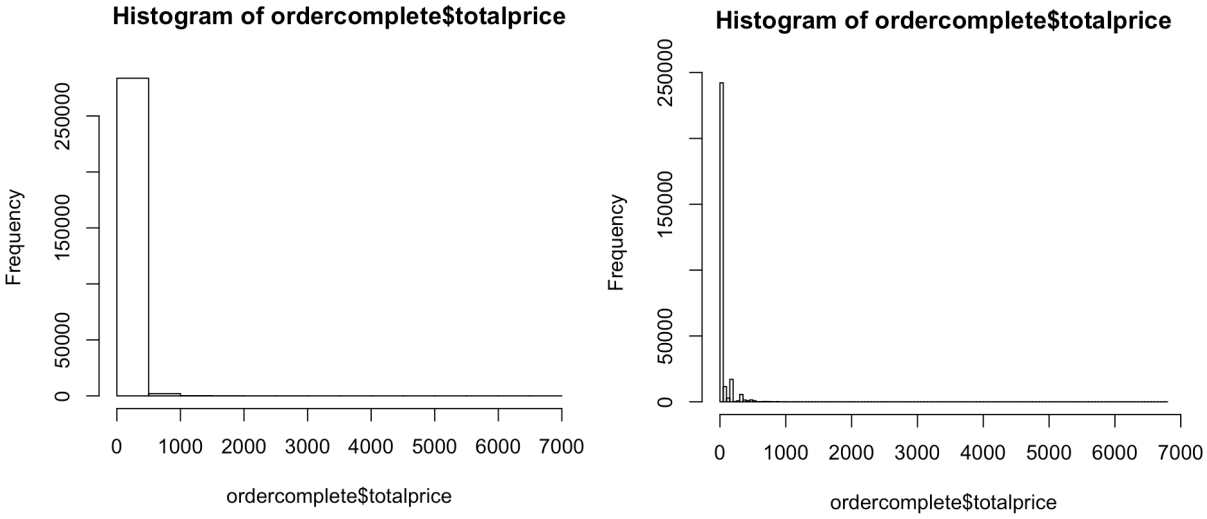
```
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.00   10.05   18.00   47.93   30.00 6780.00
```

Thống kê cho thấy dải giá trị totalprice chạy từ 0 đến 6780.

Giá trị trung bình là 47.93, tuy nhiên trung vị chỉ là 18.00. Trung vị cho biết giá trị mà dưới ngưỡng đó có 50% số giao dịch (và ngược lại cũng có 50% số giao dịch có giá trị lớn hơn trung vị). Giá trị trung vị khác trung bình (Mean) cho thấy sự phân bố bị lệch về một bên. Ta có thể kiểm chứng điều này bằng đồ thị histogram.

```
hist(ordercomplete$totalprice)
hist(ordercomplete$totalprice,100)
```

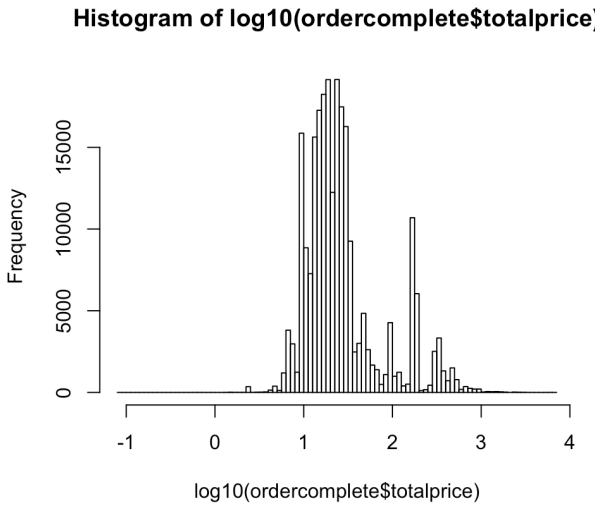
Với lệnh hist() ban đầu, khoảng chia không phù hợp với phân bố của biến nên ta thêm tham số xác định số khoảng chia là 100 trong câu lệnh sau.



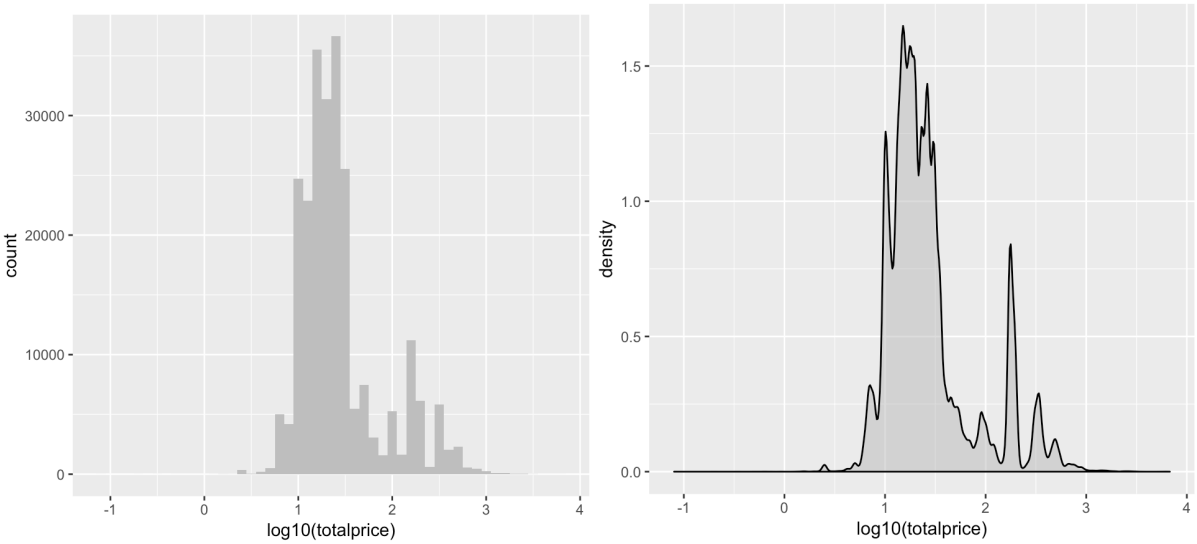
Hình 4:

Phân bố trên cho thấy có một số giao dịch có giá trị rất lớn, lớn hơn nhiều số đông các giao dịch khác. Đây là phân bố thường gặp trong kinh doanh: phân bố lợi nhuận, lương, doanh thu,… Để nhìn rõ hơn chúng ta có thể chuyển đổi thành phân bố của log().

```
hist(log10(ordercomplete$totalprice),100)
```



Hình 5: Phân bố hàm log10 của biến totalprice. Chúng ta thấy phần lớn giao dịch có giá trị giữa 7 và 100. Có 2 dải giá trị lớn hơn có tần số cao là khoảng gần 170 và 300.



Hình 6: Phân bố hàm log10 của biến totalprice được vẽ với thư viện "ggplot2". Hình (a) dạng histogram, (b) dạng mật độ (density)

Hàm phân bố cũng có thể vẽ với thư viện "ggplot2" như trong hình 6.

```
ggplot(ordercomplete) + geom_histogram(aes(x=log10(totalprice)),binwidth=0.1, fill="grey")
ggplot(ordercomplete) + geom_density(aes(x=log10(totalprice)), fill="grey", alpha = 0.5)
```

4. TK mô tả 1 biến phân loại – 1 hoặc nhiều biến liên tục (Pivotable)

Đây là một trong những phân tích phổ biến trong kinh doanh. Biến liên tục là biến số cần quan tâm, còn biến phân loại là tính chất của hoạt động kinh doanh. Chúng ta muốn đánh giá hiệu quả kinh doanh dựa trên tính phất phân loại đó:

Ví dụ 1: Doanh số theo bang

- Cách dùng hàm cơ bản aggregate()

```
tmp <- aggregate(totalprice ~ state_new, data = ordercomplete, sum)
tmp <- tmp[order(tmp$totalprice,decreasing = TRUE),]
tmp
```

hàm aggregate nhóm doanh số theo state_new và tính tổng totalprice của mỗi nhóm. Tuy nhiên chúng ta muốn hiển thị theo thứ tự sắp xếp giảm dần nên cần lưu ra biến trung gian tmp và thực hiện sắp xếp trên biến trung gian tmp.

- Cách dùng thư viện "dplyr" (cài đặt nếu máy chưa cài)

```
library("dplyr")
ordercomplete %>% group_by(state_new) %>% summarise(totalrevenue = sum(totalprice)) %>%
  arrange(desc(totalrevenue))
```

Với thư viện dplyr, chúng ta thực hiện các thao tác trên theo dạnh "pipeline", với các câu lệnh nối tiếp bởi dấu %?%. Đây là một câu lệnh rất hữu ích trong phân tích dữ liệu. Chúng ta có thể thống kê nhiều trường, nhiều hàm thống kê khác nhau trong cùng một câu lệnh. Ví dụ:

```
ordercomplete %>% group_by(state_new) %>% summarise(totalrevenue = sum(totalprice),
                                                    numbertrans = length(orderlineid),
                                                    avgtransvalue = mean(totalprice),
```

```
                                maxtransvalue = max(totalprice)) %>%
 arrange(desc(totalrevenue))
```

```
# A tibble: 36 x 5
   state_new totalrevenue numbertrans avgtransvalue maxtransvalue
   <chr>           <dbl>        <int>         <dbl>         <dbl>
 1 NY            3752449.        78939          47.5          5500
 2 CA            1524212.        26432          57.7          6250
 3 NJ            1438056.        31151          46.2          3000
 4 CT             642502.        13744          46.7          2475
 5 FL             641982.        15195          42.2          2430
 6 IL             496298.         8759          56.7          2304
 7 TX             473342.         9184          51.5          2500
 8 MA             472521.        10266          46.0          1750
 9 OTHERS         457837.         9821          46.6          6780
10 PA             435774.        10318          42.2          2665
# ... with 26 more rows
```

Có thể thấy bang NY có tổng doanh số cao nhất nhưng các bang CA, IL có giá trị giao dịch trung bình cao nhất. Và giao dịch có giá trị cao nhất, 6780, nằm ở các bang khác (OTHERS). Câu lệnh trên cho phép thống kê nhiều trường liên tục (totalprice và orderlineid) theo 1 trường phân loại (state_new).

Ví dụ 2: Giá trị giao dịch theo loại thanh toán

Chúng ta sẽ dùng thư viện "dplyr" cho các phân tích tương tự về sau. Câu lệnh sau cho phép thông kê các thông số cơ bản của giá trị giao dịch theo loại thanh toán

```
ordercomplete %>% group_by(paymenttype) %>% summarise(totalrevenue = sum(totalprice),
                                numbertrans = length(orderlineid),
                                avgtransvalue = mean(totalprice),
                                maxtransvalue = max(totalprice)) %>%
 arrange(desc(totalrevenue))
```

```
# A tibble: 7 x 5
  paymenttype totalrevenue numbertrans avgtransvalue maxtransvalue
  <fct>             <dbl>        <int>         <dbl>         <dbl>
1 VI              5008900.       112948          44.3          6250
2 AE              4651053.        72989          63.7          3165
3 MC              3298890.        70934          46.5          3165
4 DB               470842.        17869          26.3          6780
5 OC               264547.        10637          24.9          2665
6 ""                13481.          250          53.9           645
7 ??                 1184.          390           3.04          306
```

Có thể thấy loại giao dịch VI, AE và MC chiếm đa số, trong đó AE có giá trị giao dịch trung bình vượt trội. Chúng ta có thể đưa thêm các thống kê theo khách hàng vào cùng câu lệnh:

```
ordercomplete %>% group_by(paymenttype) %>%
 summarise(totalrevenue = sum(totalprice),
 numbertrans = length(orderlineid),
 avgtransvalue = mean(totalprice),
 maxtransvalue = max(totalprice),
 numbercustomer = length(unique(customerid)),
 avgcustvalue = sum(totalprice)/length(unique(customerid))) %>%
arrange(desc(totalrevenue))
```

```
# A tibble: 7 x 7
  paymenttype totalrevenue numbertrans avgtransvalue maxtransvalue numbercustomer avgcustvalue
  <fct>             <dbl>       <int>         <dbl>         <dbl>           <int>        <dbl>
1 VI              5008900.      112948          44.3          6250           75233         66.6
2 AE              4651053.       72989          63.7          3165           46752         99.5
3 MC              3298890.       70934          46.5          3165           46535         70.9
4 DB               470842.       17869          26.3          6780           12430         37.9
5 OC               264547.       10637          24.9          2665            8155         32.4
6 ""                13481.         250          53.9           645             157         85.9
7 ??                 1184.         390           3.04          306             304          3.90
```

TK có điều kiện

Dựa trên các thông kê trên chúng ta có thể quan tâm đến một nhóm đối tượng nào đó và tiếp tục thực hiện các thống kê sâu hơn trong nhóm, chúng ta có thể dùng hàm filter trong "dplyr":

Ví dụ thống kê trên cho riêng bang NY:

```
ordercomplete %>% group_by(paymenttype) %>% filter(state == "NY") %>%
 summarise(totalrevenue = sum(totalprice),
        numbertrans = length(orderlineid),
        avgtransvalue = mean(totalprice),
        maxtransvalue = max(totalprice),
        numbercustomer = length(unique(customerid)),
        avgcustvalue = sum(totalprice)/length(unique(customerid))) %>%
 arrange(desc(totalrevenue))
```

```
  paymenttype totalrevenue numbertrans avgtransvalue maxtransvalue numbercustomer avgcustvalue
  <fct>             <dbl>       <int>         <dbl>         <dbl>           <int>        <dbl>
1 AE              1597055.       24666          64.7          3127           16234         98.4
2 VI               986460.       22322          44.2          2848.          15659         63.0
3 MC               928027.       20636          45.0          2398.          13777         67.4
4 DB               174626.        8037          21.7          5500            4887         35.7
5 OC                65991.        3155          20.9           625            2523         26.2
6 ??                  290.         123           2.36           81.0            95          3.05
```

Chúng ta thấy riêng với bang NY, giao dịch của AE lại chiếm chủ đạo về tổng doanh số và số lượng giao dịch (với giá trị giao dịch trung bình lớn giống như thống kê trên).

5. Kiểm định trung bình 2 tập dữ liệu

Kiểm định giả thuyết là một phương pháp quan trọng trong thống kê. Người phân tích kinh doanh nên sử dụng công cụ này để khẳng định/phủ nhận các giả thuyết một cách có ý nghĩa thống kê. Chúng ta lấy ví dụ đơn giản khi so sánh giá trị trung bình của 2 tập dữ liệu.

Ví dụ 1: Doanh số trung bình giao dịch theo bang

Giả sử chúng ta cần đánh giá và xếp loại khách hàng của mỗi bang theo giá trị giao dịch trung bình. Lấy ví dụ các bang AL, FL

```
x <- subset(ordercomplete, state == "AL")  # 929 transactions
y <- subset(ordercomplete, state == "FL")  # 15195 transactions
mean(x$totalprice)
mean(y$totalprice)
```

AL có 929 giao dịch, giá trị trung bình 58.67

FL có 15195 giao dịch, giá trị trung bình 42.24

Chúng ta thấy giá trị trung bình của AL cao hơn nhiều FL. Tuy nhiên để khẳng định điều này chúng ta cần thực hiện kiểm định. Phương pháp phổ biến cho kiểm định này là t-test:

```
t.test(x$totalprice, z$totalprice, var.equal=TRUE)
```

```
        Two Sample t-test

data:  x$totalprice and y$totalprice
t = 4.3127, df = 16122, p-value = 1.622e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
  8.961867 23.895457
sample estimates:
mean of x mean of y
 58.67819  42.24953
```

p-value rất nhỏ, có thể bác bỏ H0 với độ tin cậy trên 95% (bác bỏ giả thuyết giá trị trung bình của 2 bang bằng nhau)

Với giả thuyết ban đầu H0 là giá trị trung bình của 2 bang không khác nhau. Kết quả kiểm định có thể được đánh giá bởi giá trị thống kê p-value, là xác suất có sự ngẫu nhiên về số liệu như quan sát nếu giả thuyết ban đầu H0 là đúng. Trong kiểm định với bang AL và FL, p-value rất nhỏ có nghĩa là nếu H0 đúng sẽ rất ít khả năng quan sát được dữ liệu như thực tế, từ đó suy ra cần bác bỏ H0. Hay nói cách khác là có thể khẳng định giá trị trung bình của 2 bang AL và FL khác nhau. (giá trị p-value nhỏ hơn 0.05 thường được coi là có thể bác bỏ H0) và quyết định kinh doanh có thể cân nhắc là có chính sách khách hàng khác cho AL so với FL.

Tương tự, chúng ta thực hiện kiểm định giữa bang FL và DE

```
z <- subset(ordercomplete, state == "DE")  # 1010 transactions
mean(z$totalprice)
```

Bang DE có 1010 giao dịch, giá trị trung bình 45.67 cao hơn của FL. Tuy nhiên kết quả kiểm định cho thấy giá trị trung bình của 2 bang không khác nhau do p-value = 0.26 (không đủ nhỏ để bác bỏ H0). Kết luận là chưa thể áp dụng các chính sách khách hàng khác cho bang DE so với FL.

```
t.test(y$totalprice, z$totalprice, var.equal=TRUE)
```

```
        Two Sample t-test

data:  y$totalprice and z$totalprice
t = -1.1098, df = 16203, p-value = 0.2671
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -9.466682  2.622077
sample estimates:
mean of x mean of y
 42.24953  45.67183
```

p-value không đủ nhỏ để bác bỏ H0 (không bác bỏ giả thuyết giá trị trung bình của 2 bang bằng nhau)

Trong nhiều phân tích kinh doanh, người dùng có thể chỉ so sánh giá trị trung bình giữa 2 tập dữ liệu và ra quyết định. Điều này có thể vội vàng vì quyết định đó chưa tính đến độ lớn của sự khác nhau cũng như số lượng mẫu của 2 tập dữ liệu. Việc nắm vững phương pháp kiểm định thống kê rất quan trọng trong việc hỗ trợ ra quyết định kinh doanh dựa trên số liệu.

TÀI LIỆU THAM KHẢO

3 Tham khảo

James, Gareth; Daniela Witten; Trevor Hastie; and Robert Tibshirani. *An Introduction to Statistical Learning*. Springer, 2013.

Nina Zumel, John Mount. *Practical Data Science with R*. Manning Publications (2014)

Luis Torgo. *Data mining with R: learning with case studies*. Chapman & Hall (2011)

Gordon S. Linoff. "*Data Analysis Using SQL and Excel*". John Wiley & Sons.(2008)