Hoàng Đức Anh

Phân tích dữ liệu thực tế với R

Gửi con trai,

nếu không có con, hai năm trước bố đã có thể hoàn thành cuốn sách này!

Mục lục

Danh mục bảng

Danh mục biểu đồ

Lời mở đầu

Trong quá trình triển khai công việc thực tế tại nhiều tổ chức khác nhau, tôi nhận thấy một hiện thực là mặt bằng kiến thức thực tế về phân tích dữ liệu trên thị trường Việt Nam còn rất yếu. Phần lớn, các bạn làm trong ngành phân tích dữ liệu rơi vào một trong hai nhóm sau:

- Nhóm một, **chưa có nền tảng về phân tích thống kê**. Ở nhóm này, các bạn thường không được đào tạo bài bản hoặc không có đủ điều kiện (phần lớn là về thời gian) để học các kiến thức về phân tích thống kê. Do nhu cầu của công việc, các bạn có thành thạo các kỹ năng về xây dựng báo cáo và phân tích khám phá dữ liệu đơn giản với SQL và Excel. Các bạn này thường thuộc các nhóm phân tích báo cáo (Business Intelligence) tại các tổ chức lớn hoặc làm trong startup. Điểm mạnh của nhóm này là làm việc sát với các bộ phận kinh doanh, hiểu rõ nghiệp vụ và nhu cầu nghiệp vụ. Tuy nhiên, điểm yếu của các bạn lại là không thể ứng dụng hoặc tự học và không biết cách triển khai các ứng dụng của khoa học dữ liệu ở mức độ cao vào công việc thực tế.
- Nhóm hai, **có nền tảng vững vàng về kiến thức thống kê, dự báo nhưng lại quá chú trọng vào các yếu tố kỹ thuật**. Ở nhóm này, các bạn đều có nền tảng kiến thức về toán, thống kê rất tốt. Một số bạn được học và đào tạo cơ bản về khoa học dữ liệu, học máy và các ứng dụng của khoa học dữ liệu. Các bạn này có thiên hướng thích xây dựng mô hình dự báo, thích làm các bài toán lớn trong kinh doanh. Điểm mạnh của nhóm này là rất thông minh, chịu khó học hỏi và có thể áp dụng những kỹ thuật phân tích mới vào thực tế một cách nhanh chóng. Nhưng ngược lại, nhóm này lại có nhược điểm chết người là có thói quen chỉ tập trung vào việc phân tích dữ liệu mà thiếu đi cái nhìn tổng quát trong việc giải quyết bài toán thực tế. Không chỉ thế, nhóm này không có thế mạnh trong việc trình bày và giao tiếp, dẫn đến các kết quả thực tế không được các đơn vị kinh doanh nghiệp vụ đón nhận và sử dụng.

Đối với một tổ chức muốn phát triển dựa vào dữ liệu và muốn biến các quyết định của tổ chức dựa vào phân tích dữ liệu, cả hai nhóm trên đều là các trạng thái nên tránh và phải cân bằng được cả hai. Cuốn sách này sẽ phân tích và giúp các bạn làm trong lĩnh vực phân tích dữ liệu hiểu rõ hơn các ưu nhược điểm của chính mình.

Tại sao nên đọc cuốn sách này

Việc viết cuốn sách này xuất phát thuần túy từ nhu cầu cá nhân của tác giả. Trong quá trình làm việc thực tế, bản thân tác giả luôn có mong muốn tổng hợp và đúc rút các kiến thức phân tích thực tế. Tuy nhiên, khi bắt đầu bắt tay vào xây dựng cuốn sách, tác giả cũng có mong muốn có thể đúc kết và giúp cho các tổ chức, bộ phận muốn tập trung vào và khai thác sức mạnh của phân tích dữ liệu trong thực tế có thể có thêm các nguồn tài liệu và kinh nghiệm, vốn rất ít được chia sẻ thực tế, để có thể thành công hơn trong công việc triển khai.

Thêm vào đó, trong quá trình làm việc và giảng dạy cho các học viên, tác giả luôn nhận được câu hỏi: *Thưa thày, em không được đào tạo bài bản về phân tích thống kê cũng như khoa học dữ liệu, liệu em có thể trở thành chuyên gia phân tích dữ liệu được không?*

Tác giả luôn trăn trở với câu hỏi trên cũng như với kinh nghiệm đào tạo thực tế và xây dựng năng lực phân tích dữ liệu ở VPBank, quyển sách này được viết ra theo cách tiếp cận *ứng dụng* của khoa học dữ liệu trong hoạt động kinh doanh. Do đó, ngôn ngữ cũng như cách tiếp cận trong cuốn sách này được cố gắng viết một cách đơn giản, dễ hiểu để trình bày các kiến thức, thuật ngữ khó hiểu của khoa học dữ liệu thành các ngôn ngữ bình dân phù hợp với nhiều đối tượng.

Cuốn sách này sẽ không đi sâu vào lý thuyết của các thuật toán, mô hình thống kê mà sẽ cố gắng trả lời các khía cạnh sau.

- Thuật toán, mô hình đó là gì?
- Mô hình đó được sử dụng như thế nào?
- Khi giải thích cho các đơn vị kinh doanh, ta cần phải giải thích điều gì?
- Ứng dụng của mô hình trong thực tế

Cấu trúc của sách

Cuốn sách được chia làm 3 phần theo thứ tự từ dễ đến khó, bao gồm.

- Phần một, các kiến thức cơ bản
- Phần hai, các ứng dụng nâng cao
- Phần ba, các vấn đề khác.

Chapters **??** introduces a new topic, and …

Các phần mềm được sử dụng

I used the **knitr** package (Xie, 2015) and the **bookdown** package (Xie, 2018) to compile my book. My R session information is shown below:

```
xfun::session_info()
```

```
## R version 3.4.0 (2017-04-21)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 17134)
##
## Locale:
##   LC_COLLATE=English_United States.1252
##   LC_CTYPE=English_United States.1252
##   LC_MONETARY=English_United States.1252
##   LC_NUMERIC=C
##   LC_TIME=English_United States.1252
##
## Package version:
##   assertthat_0.2.0   backports_1.1.2
##   base64enc_0.1.3    BH_1.66.0.1
##   bindr_0.1.1        bindrcpp_0.2
##   bookdown_0.7.18    cli_1.0.0
##   colorspace_1.3-0   compiler_3.4.0
##   crayon_1.3.4       dichromat_2.0.0
##   digest_0.6.16      dplyr_0.7.4
##   evaluate_0.11      ggplot2_3.0.0
##   ggthemes_3.0.1     glue_1.3.0
##   graphics_3.4.0     grDevices_3.4.0
##   grid_3.4.0         gtable_0.2.0
##   highr_0.7          htmltools_0.3.6
##   jsonlite_1.5       knitr_1.20
##   labeling_0.3       lattice_0.20.35
##   lazyeval_0.2.0     magrittr_1.5
##   markdown_0.8       MASS_7.3.47
##   Matrix_1.2.9       methods_3.4.0
##   mgcv_1.8.17        mime_0.5
##   munsell_0.4.3      nlme_3.1.131
##   pillar_1.2.1       pkgconfig_2.0.1
##   plogr_0.2.0        plyr_1.8.4
##   R6_2.2.2           RColorBrewer_1.1.2
##   Rcpp_0.12.18       reshape2_1.4.2
##   rlang_0.2.1        rmarkdown_1.10
```

```
##   rprojroot_1.3-2    rstudioapi_0.7
##   scales_0.5.0       stats_3.4.0
##   stringi_1.1.7      stringr_1.3.1
##   tibble_1.4.2       tinytex_0.8
##   tools_3.4.0        utf8_1.1.3
##   utils_3.4.0        viridisLite_0.3.0
##   withr_2.1.2        xfun_0.3
##   yaml_2.2.0
```

Package names are in bold text (e.g., **rmarkdown**), and inline code and filenames are formatted in a typewriter font (e.g., `knitr::knit('foo.Rmd')`). Function names are followed by parentheses (e.g., `bookdown::render_book()`).

Lời cảm ơn

Cảm ơn những người sau đã giúp chúng tôi hoàn thành quyển sách

Frida Gomam
on the Mars

Về tác giả

Tác giả Hoàng Đức Anh hiên đang là trưởng phòng phân tích nâng cao của ngân hàng `VPBank`. Đức Anh được đào tạo về phân tích dữ liệu tại trường đại học kinh tế Vác-sa-va (Warsaw School of Economics), Vác-sa-va, cộng hòa Ba Lan ở cả hai cấp, đại học và thạc sỹ từ năm 2009-2014.

Phần I

Các kiến thức cơ bản

1

Khoa học dữ liệu và nghề phân tích dữ liệu

1.1 Khoa học dữ liệu

Định lý 1.1 (Pytago)**.** *Trong một tam giác vuông, bình phương cạnh huyền bằng tổng bình phương hai cạnh góc vuông*

$$a^2 + b^2 = c^2$$

1.2 Tại sao phân tích dữ liệu là nghề khó?

Phân tích dữ liệu đòi hỏi cùng lúc thực hiện ba nhóm công việc sau:

- Hiểu biết về vấn đề kinh doanh. Nghe có vẻ đơn giản nhưng qua quá trình làm việc thực tế, kinh nghiệm của tác giả cho thấy đây có lẽ là phần dễ bị bỏ qua nhất bởi lẽ mấy nguyên nhân sau.
  - Sự khác biệt của vận hành kinh doanh so với hoạt động phân tích dữ liệu.
  - Tư duy và thái độ của nhóm phân tích dữ liệu. Các bạn phân tích kinh doanh (hoặc đôi khi được gọi là phân tích kinh doanh) thường tự coi mình là đơn vị hỗ trợ và *cung cấp* dữ liệu theo yêu cầu. Với lối tư duy thụ động này, các bạn sẽ không có nhu cầu tìm hiểu cặn kẽ các hoạt động kinh doanh, dẫn đến không hiểu hoạt động kinh doanh đủ sâu để có thể tư vấn và thuyết phục các bên kinh doanh trong việc triển khai các dự án phân tích dữ liệu mới.
  - Nghiệp vụ kinh doanh rất phức tạp và thiếu hệ thống tài liệu ghi chép dưới góc độ khái quát cho hoạt động phân tích dữ liệu. Đây là vấn đề phần lớn các nhóm phân tích dữ liệu gặp phải. Để giải quyết vấn đề này, trong phần sau tác giả sẽ đưa ra phương pháp tìm hiểu hoạt động kinh doanh theo 6 nhóm vấn đề.
- Khai thác và phân tích dữ liệu

- Trình bày, thuyết phục và tư vấn cho các bên kinh doanh về kết quả phân tích dữ liệu.

Ba cấp độ của viết code: Khi sử dụng các công cụ phân tích dữ liệu (viết code), ta sẽ trải qua 3 cấp độ như sau:

1. Viết thứ đơn giản. Ở cấp độ này, các bạn thường mới nhập môn phân tích dữ liệu, đầy lo lắng và thiếu tự tin ở bản thân và bắt đầu với những bài phân tích đơn giản để áp dụng các kiến thức mới học.
2. Viết càng nguy hiểm càng tốt. Ở giai đoạn này, các bạn đã có một lượng kiến thức nền tương đối vững và bắt đầu đi vào các phương pháp nâng cao. Do đó, các bạn thường có xu hướng khiến mọi thứ trở nên *nguy hiểm*, tô vẽ và đưa ra nhiều yếu tố không thực sự cần thiết. Với các bạn có xu hướng trực quan hóa, sẽ là đưa ra các biểu đồ đầy màu sắc và cực kỳ nguy hiểm. Với các bạn có xu hướng xây dựng mô hình dự báo, sẽ là dùng mô hình dự báo, học máy hoặc deep-learning mọi lúc, mọi nơi. Kết quả sẽ khiến người đọc ấn tượng nhưng có thể chưa thực sự có tính ứng dụng cao.
3. Chỉ viết những thứ có khả năng tái sử dụng, giải quyết vấn đề bằng phương pháp đơn giản nhất có thể có. Ở giai đoạn này, các bạn đã dung hòa được rất nhiều kiến thức của nghành phân tích dữ liệu với nhau và tiếp cận các vấn đề một cách mạch lạc, logic và rất chặt chẽ. Các bạn nắm rất vững khi nào nên dùng các phương pháp phân tích khám phá dữ liệu đơn giản thay cho các thuật toán phức tạp trong việc giải quyết các vấn đề kinh doanh.

Định lý 1.2 (Pytago)**.** *Trong một tam giác vuông, bình phương cạnh huyền bằng tổng bình phương hai cạnh góc vuông*

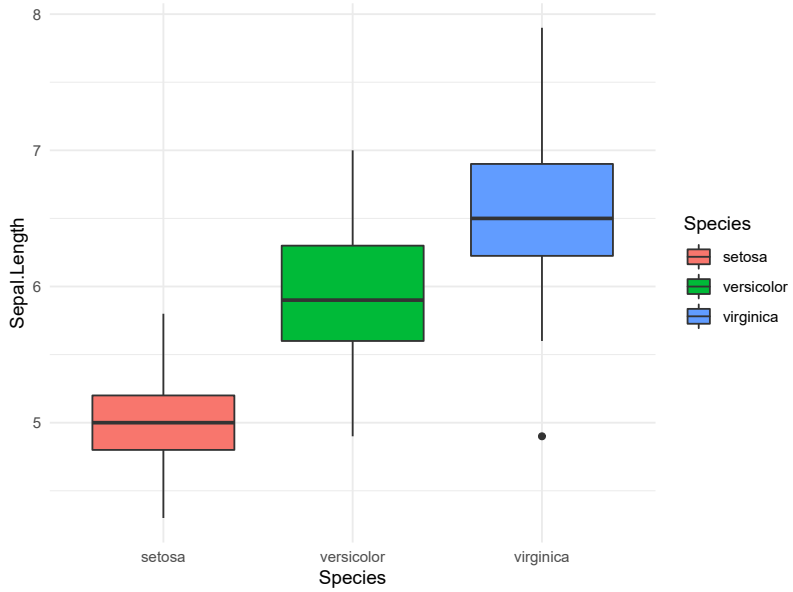
$$a^2 + b^2 = c^2$$

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

```
library(tidyverse)
iris %>%
  ggplot(aes(Species, Sepal.Length)) +
  geom_boxplot(aes(fill = Species)) +
  theme_minimal()
```

Reference a figure by its code chunk label with the `fig:` prefix, e.g., see Figure 1.1. Similarly, you can reference tables generated from `knitr::kable()`, e.g., see Table ??.

```
knitr::kable(
  head(iris, 20), caption = 'Here is a nice table!',
```



Hình 1.1: Here is a nice figure!

```
  booktabs = TRUE
)
```

You can write citations, too. For example, we are using the **bookdown** package (Xie, 2018) in this sample book, which was built on top of R Markdown and **knitr** (Xie, 2015).

Bảng 1.1: Here is a nice table!

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa

Phần II

Ứng dụng thực tế

2

Các phương pháp phân tích dữ liệu thực tế ứng dụng với R

We talk about the *FOO* method in this chapter.

2.1 Phần 1

2.2 Phần 2

2.3 Phần 3

3

Ngữ pháp của biến đổi dữ liệu với DPLYR

Khi bắt tay vào công việc phân tích số liệu, việc đầu tiên ta cần phải làm là thu thập dữ liệu từ nhiều nguồn khác nhau. Sau khi hoàn thành xong bước này, ta sẽ phải dành phần lớn thời gian để làm sạch, biến đổi và tổng hợp dữ liệu nhằm tìm kiếm các insights hoặc chuẩn bị dữ liệu cho các bước xây dựng mô hình, dự báo.

R rất mạnh trong việc biến đổi dữ liệu và có rất nhiều package hỗ trợ cho công việc này. Tuy nhiên, thư viện nổi tiếng nhất trong R trong việc làm sạch và biến đổi dữ liệu là `dplyr`, một thư viện nổi tiếng với những tính năng chuyên cho việc xử lý, tổng hợp dữ liệu trước khi xây dựng mô hình phân tích dữ liệu. Chương này sẽ tập trung vào giới thiệu về những hàm cơ bản nhất của `dplyr`.

Trước khi bắt đầu nội dung bài giảng, chúng ta có thể download và gọi gói dplyr.

```
#install.packages("dplyr")
library(dplyr)
```

3.1 Giới thiệu về pipe operator

Khi viết các câu lệnh, thông thường ta có 2 cách viết phổ biến sau.

- Cách 1: Viết với các câu lệnh lồng vào nhau (nested). Với cách viết này, các hàm sẽ được viết lồng vào nhau và kết quả của hàm sẽ được tính toán theo thứ tự từ trong ra ngoài.
- Cách 2: Viết lưu dưới dạng các đối tượng trung gian. Với cách viết này, từng đối tượng sẽ được tính toán từng phần và kết quả sẽ được hiển thị một cách mạch lạc hơn. Tuy nhiên, nhược điểm của phương pháp này là sẽ tạo ra rất nhiều đối tượng trung gian, gây ra khó khăn trong việc theo dõi và quản lỹ.

Giả sử ta cần tính toán độ lệch chuẩn của véc-tơ x, công thức tính độ lệch chuẩn sẽ là

$$\sigma = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n-1}$$

Với hai cách viết code khác nhau, ta có thể tính độ lệch chuẩn theo hai cách.

```
# Cách 1
# Tạo vector x
x <- seq(2, 100, 2)
# Tính độ lệch chuẩn
sqrt(sum((x-mean(x))^2)/(length(x)-1))
```

```
## [1] 29.15
```

```
sd(x)
```

```
## [1] 29.15
```

```
# Cách 2
# Tạo vector x
x <- seq(2, 100, 2)
# Tính tổng bình phương
sum_sqr <- sum((x-mean(x))^2)
len_x <- length(x)
var <- sum_sqr/(len_x - 1)
sd <- var^(1/2)
sd
```

```
## [1] 29.15
```

Ta thấy kết quả ở hai cách tính là như nhau. Tuy nhiên, cách viết hai sẽ tạo ra nhiều đối tượng trung gian hơn cách viết 1 rất nhiều. Khi phân tích dữ liệu thực tế, ta sẽ phải áp dụng cả 2 cách viết code để có thể vận dụng linh hoạt trong từng trường hợp cụ thể.

Trong R, có cách viết code thứ ba, được gọi là cách suwr dụng `pipe operator (%>%)`. `Toán tử Pipe` cho phép viết code theo cách đơn giản và dễ theo dõi giúp cho người đọc và người viết code trên R có thể theo dõi được code một cách dễ dàng nhất. Câu trúc của pipe như sau

---

f(x, y) = x %>% f(., y)

Ví dụ của pipe.

```
# Cách 1
mean(x)
# Cách 2
x %>% mean
```

Ta có thể xem xét ví dụ phức tạp hơn.

```
# Cách 1 - dùng cách viết thường
summary(head(iris))
```

```
##   Sepal.Length   Sepal.Width    Petal.Length
##  Min.   :4.60   Min.   :3.00   Min.   :1.30
##  1st Qu.:4.75   1st Qu.:3.12   1st Qu.:1.40
##  Median :4.95   Median :3.35   Median :1.40
##  Mean   :4.95   Mean   :3.38   Mean   :1.45
##  3rd Qu.:5.08   3rd Qu.:3.58   3rd Qu.:1.48
##  Max.   :5.40   Max.   :3.90   Max.   :1.70
##   Petal.Width          Species
##  Min.   :0.200   setosa    :6
##  1st Qu.:0.200   versicolor:0
##  Median :0.200   virginica :0
##  Mean   :0.233
##  3rd Qu.:0.200
##  Max.   :0.400
```

```
# Cách 2 - dùng pipe
iris %>% head %>% summary
```

```
##   Sepal.Length   Sepal.Width    Petal.Length
##  Min.   :4.60   Min.   :3.00   Min.   :1.30
##  1st Qu.:4.75   1st Qu.:3.12   1st Qu.:1.40
##  Median :4.95   Median :3.35   Median :1.40
##  Mean   :4.95   Mean   :3.38   Mean   :1.45
##  3rd Qu.:5.08   3rd Qu.:3.58   3rd Qu.:1.48
##  Max.   :5.40   Max.   :3.90   Max.   :1.70
##   Petal.Width          Species
##  Min.   :0.200   setosa    :6
##  1st Qu.:0.200   versicolor:0
##  Median :0.200   virginica :0
##  Mean   :0.233
##  3rd Qu.:0.200
##  Max.   :0.400
```

Cả hai cách đều cho ra kết quả giống nhau. Tuy nhiên, cách hai sẽ dễ theo dõi, dễ đọc hơn cách 1 rất nhiều. Cách đọc hiểu quá trình thực hiện pipe như sau:

1. Gọi tập dữ liệu `iris` để phân tích
2. Thực hiện hàm `head` trên tập dữ liệu này, *được kết quả bao nhiêu...*
3. ... tiếp tục thực hiện hàm `summary`

Như ta thấy, cách viết theo phong cách của `pipe operator (%>%)` cho phép ta thực hiện các phép tính theo đúng mạch tư duy logic của bản thân. Điều này là một điểm rất mạnh mà hiện tại, mới chỉ ở R có toán tử `%>%` áp dụng được cho mọi hàm.

Một số đặc tính cơ bản của %>%:

1. Theo mặc định, Phía tay trái (LHS) sẽ được chuyển tiếp thành yếu tố đầu tiên của hàm được sử dụng phía tay phải (RHS), ví dụ:

```
mean(x)
```

```
## [1] 51
```

```
# Tương đương với:
x %>% mean
```

```
## [1] 51
```

2. Khi LHS không còn là yếu tố đầu tiên của một hàm RHS, thì dấu "." được sử dụng để định vị cho LHS, ví dụ:

```
library(dplyr)
# Cách 1
summary(lm(mpg ~ cyl, data = mtcars))

# Cách 2
mtcars %>%
  lm(mpg ~ cyl, data = .) %>%
  summary
```

Trong tình huống trên, tham số về dữ liệu trong hàm lm không phải là ở đầu, mà sau phần công thức, nên chúng ta sẽ dùng dấu "." như là đại diện của thực thể mtcars ở bên ngoài (LHS) của hàm lm.

3.2 Các hàm cơ bản trong dplyr

Trong công việc biến đổi dữ liệu, bất kỳ ngôn ngữ phân tích nào cũng có 3 nhóm hàm lớn.

- Nhóm 1 - các hàm truy vấn dữ liệu: Lấy dữ liệu theo dòng, theo cột và theo điều kiện. Trong `dplyr` sẽ là các hàm `select`, `filter` và `slice`
- Nhóm 2 - Các hàm tổng hợp dữ liệu: Tính toán tổng hợp dữ liệu theo chiều. Trong `dplyr` sẽ là các hàm `group_by`, `summarise`
- Nhóm 3 - Các hàm biến đổi dữ liệu: Tạo mới, biến đổi các dữ liệu cũ thành các dữ liệu mới. Trong `dplyr` sẽ là các hàm thuộc nhóm `mutate`, `join`, `bind`

Trong phần này, chúng ta sẽ giới thiệu nhanh các nhóm câu lệnh cơ bản trên.

3.2.1 Nhóm câu lệnh truy vấn dữ liệu

Khi truy vấn dữ liệu, ta thường phải thực hiện 3 nhóm công việc sau.

- Lấy theo cột
- Lấy theo dòng
- Lấy theo điều kiện

Xét về mặt bản chất, lấy theo điều kiện là một trường hợp đặc biệt của việc lấy theo dòng. Đối với các ngôn ngữ như SQL, sẽ không phân biệt hai loại này. Tuy nhiên, vì R lưu thứ tự của từng quan sát trong dataframe, nên việc phân biệt được hai loại truy vấn trên là cần thiết.

3.2.1.1 Lấy các cột trong dataframe với `select`

---

data %>% select(var1, var2, ...)

Trong đó, `var1`, `var2` là tên các cột cần truy vấn. Đặc biệt, R rất linh hoạt trong việc lọc theo cột. Ta có thể truy vấn theo tên, theo thứ tự hoặc thậm chí theo các khoảng thứ tự các biến. Xem ví dụ sau.

```
library(dplyr)
# Xêm tên các biến trong mtcars
mtcars %>% names
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec"
##  [8] "vs"   "am"   "gear" "carb"
```

```
# Chọn cột mpg và cyl
mtcars %>% select(mpg, cyl) %>% head
```

```
##                   mpg cyl
```

```
## Mazda RX4          21.0   6
## Mazda RX4 Wag      21.0   6
## Datsun 710         22.8   4
## Hornet 4 Drive     21.4   6
## Hornet Sportabout 18.7   8
## Valiant            18.1   6
```

```
# Chọn cột thứ nhất và thứ hai
mtcars %>% select(1,2) %>% head
```

```
##                    mpg cyl
## Mazda RX4          21.0   6
## Mazda RX4 Wag      21.0   6
## Datsun 710         22.8   4
## Hornet 4 Drive     21.4   6
## Hornet Sportabout 18.7   8
## Valiant            18.1   6
```

```
# Chọn cột thứ 3 đến cột thứ 6
mtcars %>% select(3:6) %>% head
```

```
##                   disp  hp drat    wt
## Mazda RX4          160 110 3.90 2.620
## Mazda RX4 Wag      160 110 3.90 2.875
## Datsun 710         108  93 3.85 2.320
## Hornet 4 Drive     258 110 3.08 3.215
## Hornet Sportabout  360 175 3.15 3.440
## Valiant            225 105 2.76 3.460
```

Ngoài ra, khi lấy chi tiết các cột (liệt kê từng cột) khi lấy dữ liệu trên 1 bảng, bạn có thể dùng một số hàm sau để hỗ trợ việc lấy trường dữ liệu được nhanh hơn:

- `starts_with("Ký tự là thông tin mong muốn")`: các cột dữ liệu ccó tên hứa các ký tự mong muốn đứng ở đầu của tên, ví dụ:

```
iris %>%
  select(starts_with("Petal")) %>%
  head
```

```
##   Petal.Length Petal.Width
## 1          1.4         0.2
## 2          1.4         0.2
## 3          1.3         0.2
## 4          1.5         0.2
## 5          1.4         0.2
## 6          1.7         0.4
```

- `ends_with("Ký tự là thông tin mong muốn")`: các cột dữ liệu có tên chứa các ký tự mong muốn ở cuối của tên, ví dụ:

```
iris %>%
  select(ends_with("Length")) %>%
  head
```

```
##   Sepal.Length Petal.Length
## 1          5.1          1.4
## 2          4.9          1.4
## 3          4.7          1.3
## 4          4.6          1.5
## 5          5.0          1.4
## 6          5.4          1.7
```

- `contains("Ký tự là thông tin mong muốn")`: các cột dữ liệu có tên chứa chính xác các ký tự mong muốn ở bất kỳ vị trí nào của tên, ví dụ:

```
iris %>%
  select(contains("etal")) %>%
  head
```

```
##   Petal.Length Petal.Width
## 1          1.4         0.2
## 2          1.4         0.2
## 3          1.3         0.2
## 4          1.5         0.2
## 5          1.4         0.2
## 6          1.7         0.4
```

- matches("Dạng ký tự là thông tin mong muốn"): các cột dữ liệu có tên chứa các ký tự có dạng ký tự mong muốn ở bất kỳ vị trí nào của tên, ví dụ:

```
iris %>%
  select(matches(".t.")) %>%
  head
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1          5.1         3.5          1.4         0.2
## 2          4.9         3.0          1.4         0.2
## 3          4.7         3.2          1.3         0.2
## 4          4.6         3.1          1.5         0.2
## 5          5.0         3.6          1.4         0.2
## 6          5.4         3.9          1.7         0.4
```

Trong ví dụ trên, R sẽ lấy tất cả các cột có tên chứa chữ t và có ký tự khác ở trước và sau (các ký tự chỉ chứa chữ t mà chữ t ở đâu hoặc cuối tên sẽ không được tính vào)

Thêm vào đó, ta có thể đổi tên biến ngay trong khi lựa chọn các biến với `select` như sau:

```
mtcars %>%
  select(`miles per gallon` = mpg
         , cylinder = cyl
         , weight = wt) %>%
  head
```

```
##                   miles per gallon cylinder weight
## Mazda RX4                     21.0        6  2.620
## Mazda RX4 Wag                 21.0        6  2.875
## Datsun 710                    22.8        4  2.320
## Hornet 4 Drive                21.4        6  3.215
## Hornet Sportabout             18.7        8  3.440
## Valiant                       18.1        6  3.460
```

3.2.1.2 Lấy các dòng trong dataframe với `slice`

---

data %>% slice(observation)

Tương tự như lấy theo cột, ta có thể lấy các dòng trong một dataframe. Tuy nhiên, lưu ý hàm `slice` chỉ cho phép điều kiện lấy quan sát là một véc-tơ. Xem ví dụ sau.

```
# Lấy dòng đầu tiên
mtcars %>% slice(1)
```

```
## # A tibble: 1 x 11
##     mpg   cyl  disp    hp  drat    wt  qsec    vs
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1   21.    6.  160.  110.  3.90  2.62  16.5    0.
## # ... with 3 more variables: am <dbl>, gear <dbl>,
## #   carb <dbl>
```

```
# Lấy dòng từ 1:3
mtcars %>% slice(1:3)
```

```
## # A tibble: 3 x 11
##     mpg   cyl  disp    hp  drat    wt  qsec    vs
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  21.0    6.  160.  110.  3.90  2.62  16.5    0.
```

```
## 2  21.0    6.  160.  110.  3.90  2.88  17.0    0.
## 3  22.8    4.  108.   93.  3.85  2.32  18.6    1.
## # ... with 3 more variables: am <dbl>, gear <dbl>,
## #   carb <dbl>
```

```
# Lấy dòng 1:3 và 5
mtcars %>% slice(c(1:3,5))
```

```
## # A tibble: 4 x 11
##     mpg   cyl  disp    hp  drat    wt  qsec    vs
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  21.0    6.  160.  110.  3.90  2.62  16.5    0.
## 2  21.0    6.  160.  110.  3.90  2.88  17.0    0.
## 3  22.8    4.  108.   93.  3.85  2.32  18.6    1.
## 4  18.7    8.  360.  175.  3.15  3.44  17.0    0.
## # ... with 3 more variables: am <dbl>, gear <dbl>,
## #   carb <dbl>
```

3.2.1.3 Lọc quan sát theo điều kiện với `filter`

---

data %>% filter(condition)

Hàm filter cho phép ta sử dụng các điều kiện phức tạp để truy xuất dữ liệu từ dataframe. Các điều kiện thường dùng bao gồm.

Dấu	Ký hiệu	Ví dụ
Bằng	`==`	`7==8`
Khác	`!=`	`7!=8`
Lớn hơn	`>`	`a > b`
Lớn hơn hoặc bằng	`>=`	`a >= b`
Nhỏ hơn	`<`	`a < b`
Nhỏ hơn hoặc bằng	`<=`	`a <= b`
Và	`&`	`a > 7 & b < 9`

Xem ví dụ sau.

```
# Lọc điều kiện mpg > 20
mtcars %>%
```

```
  filter(mpg > 20) %>%
  dim
```

```
## [1] 14 11
```

```
# Lọc điều kiện mpg >20 hoặc mpg <18
mtcars %>%
  filter(mpg > 20 | mpg < 18) %>%
  dim
```

```
## [1] 27 11
```

```
# Lọc điều kiện mpg >=20 và cyl = 6
mtcars %>%
  filter(mpg > 20 & cyl == 6)
```

```
##    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## 1 21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## 2 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## 3 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
```

Lưu ý: Khi điều kiện *hoặc* là chuỗi các giá trị rời rạc áp dụng cho cùng một trường, chúng ta có thể làm ngắn gọn hơn với cấu trúc "%in%" thay vì cấu phải liệt kê tất cả các điều kiện đơn lẻ và ngăn cách nhau bởi dấu "|":

```
mtcars %>%
 filter(carb == 4 | carb == 3 | carb == 1)
```

```
##     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## 1  21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## 2  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## 3  22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## 4  21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## 5  18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## 6  14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## 7  19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## 8  17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## 9  16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## 10 17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## 11 15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## 12 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## 13 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## 14 14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## 15 32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## 16 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## 17 21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## 18 13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## 19 27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
```

```
## 20 15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
```

Câu lệnh trên tương đương với:

```
mtcars %>%
  filter(carb %in% c(1, 3, 4))
```

3.2.1.4 Sắp xếp dữ liệu với arrange

---

data %>% arrange(var1, var2)

Ngoài việc lọc dữ liệu có điều kiện, chúng ta cũng thường xuyên thực hiện việc sắp xếp dữ liệu theo một trật tự nhất định nào đó khi xem dữ liệu. Hàm arrange() hỗ trợ công việc này. Cách thức sắp xếp dữ liệu mặc định là từ nhỏ đến lớn.

```
mtcars %>%
  arrange(mpg) %>%
  head
```

```
##    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## 1 10.4   8  472 205 2.93 5.250 17.98  0  0    3    4
## 2 10.4   8  460 215 3.00 5.424 17.82  0  0    3    4
## 3 13.3   8  350 245 3.73 3.840 15.41  0  0    3    4
## 4 14.3   8  360 245 3.21 3.570 15.84  0  0    3    4
## 5 14.7   8  440 230 3.23 5.345 17.42  0  0    3    4
## 6 15.0   8  301 335 3.54 3.570 14.60  0  1    5    8
```

Khi có nhiều biến cần được sắp xếp, hàm arrange sẽ ưu tiên các biến theo thứ tự từ trái sang phải.

```
mtcars %>% arrange(mpg, cyl)
```

```
##     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## 1  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## 2  10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## 3  13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## 4  14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## 5  14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## 6  15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## 7  15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## 8  15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
```

```
## 9  15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## 10 15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## 11 16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## 12 17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## 13 17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## 14 18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## 15 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## 16 19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## 17 19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## 18 19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## 19 21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## 20 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## 21 21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## 22 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## 23 21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## 24 22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## 25 22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## 26 24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## 27 26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## 28 27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## 29 30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## 30 30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## 31 32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## 32 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
```

3.2.1.5 Đổi tên biến với `rename`

---

data %>% rename(new_var = old_var)

```
mtcars %>%
  rename(displacement = disp,
         miles_per_gallon = mpg) %>%
  names
```

```
##  [1] "miles_per_gallon" "cyl"
##  [3] "displacement"     "hp"
##  [5] "drat"             "wt"
##  [7] "qsec"             "vs"
##  [9] "am"               "gear"
## [11] "carb"
```

3.2.2 Nhóm câu lệnh biến đổi dữ liệu

3.2.2.0.1 Tạo mới trường dữ liệu với `mutate`

Trong quá trình xử lý dữ liệu, ta thường xuyên phải tạo thêm các trường dữ liệu mới (trường dữ liệu phát sinh). Hàm mutate() được sử dụng để làm công việc này. Cấu trúc của hàm rất đơn giản như sau.

---

data %>% mutate(new_var = statement)

Xem ví dụ sau

```
mtcars %>%
  select(mpg) %>%
  mutate(new_mpg = mpg * 2) %>%
  head
```

```
##    mpg new_mpg
## 1 21.0    42.0
## 2 21.0    42.0
## 3 22.8    45.6
## 4 21.4    42.8
## 5 18.7    37.4
## 6 18.1    36.2
```

Trong một số trường hợp, khi ta không muốn lấy các trường thông tin cũ mà chỉ muốn lấy các trường thông tin mới tạo thì có thể sử dụng hàm `transmute()` với cấu trúc giống như hàm mutate.

```
mtcars %>%
  select(mpg) %>%
  transmute(new_mpg = mpg * 1.61) %>%
  head
```

```
##   new_mpg
## 1   33.81
## 2   33.81
## 3   36.71
## 4   34.45
## 5   30.11
## 6   29.14
```

3.2.2.0.2 Gộp nhiều bảng với nhóm hàm `join`

- Hàm `inner_join(x, y, by = "key")`: lấy tất cả dữ liệu có trên bảng hai bảng khi trùng key, ví dụ:

```
x <- data.frame(student_id = seq(1, 10, 1),
                maths = c(10, 8, 7, 6, 7.8, 4,
                          7.7, 9, 9.5, 6.5))
y <- data.frame(student_id = seq(2, 20, 2),
                physics = c(8, 9.5, 7.5, 6, 5.5,
                            6.5, 7.8, 8.2, 8, 7.5))
x
```

```
##    student_id maths
## 1           1  10.0
## 2           2   8.0
## 3           3   7.0
## 4           4   6.0
## 5           5   7.8
## 6           6   4.0
## 7           7   7.7
## 8           8   9.0
## 9           9   9.5
## 10         10   6.5
```

```
y
```

```
##    student_id physics
## 1           2     8.0
## 2           4     9.5
## 3           6     7.5
## 4           8     6.0
## 5          10     5.5
## 6          12     6.5
## 7          14     7.8
## 8          16     8.2
## 9          18     8.0
## 10         20     7.5
```

```
# gộp 2 bảng dữ liệu x và y theo student_id
x %>%
  inner_join(y, by = "student_id")
```

```
##   student_id maths physics
## 1          2   8.0     8.0
## 2          4   6.0     9.5
```

```
## 3           6   4.0     7.5
## 4           8   9.0     6.0
## 5          10   6.5     5.5
```

- **full_join**: lấy tất cả dữ liệu có cả trên bảng x, y.

---

full_join(x, y, by = "key")

```
x %>%
  full_join(y, by = "student_id")
```

```
##    student_id maths physics
## 1           1  10.0      NA
## 2           2   8.0     8.0
## 3           3   7.0      NA
## 4           4   6.0     9.5
## 5           5   7.8      NA
## 6           6   4.0     7.5
## 7           7   7.7      NA
## 8           8   9.0     6.0
## 9           9   9.5      NA
## 10         10   6.5     5.5
## 11         12    NA     6.5
## 12         14    NA     7.8
## 13         16    NA     8.2
## 14         18    NA     8.0
## 15         20    NA     7.5
```

Trong ví dụ trên, các giá trị về điểm toán (maths) sẽ trả về NA cho các student_id không tồn tại trên bảng y và ngược lại cho bảng x với các giá trị điểm vật lý (physics) của các student_id không tồn tại trên bảng x.

- Hàm **left_join**: lấy dữ liệu chỉ có trên bảng x, ví dụ:

---

left_join(x, y, by = "var")

```
x %>%
  left_join(y, by = "student_id")
```

```
##    student_id maths physics
## 1           1  10.0      NA
## 2           2   8.0     8.0
## 3           3   7.0      NA
## 4           4   6.0     9.5
## 5           5   7.8      NA
## 6           6   4.0     7.5
## 7           7   7.7      NA
## 8           8   9.0     6.0
## 9           9   9.5      NA
## 10         10   6.5     5.5
```

Với các `student_id` không có giá trị trên bảng y, cột physics sẽ trả về giá trị NA

- Hàm **right_join** : lấy dữ liệu chỉ có trên bảng y, ví dụ:

right_join(x, y, by = "var")

```
x %>%
  right_join(y, by = "student_id")
```

```
##    student_id maths physics
## 1           2   8.0     8.0
## 2           4   6.0     9.5
## 3           6   4.0     7.5
## 4           8   9.0     6.0
## 5          10   6.5     5.5
## 6          12    NA     6.5
## 7          14    NA     7.8
## 8          16    NA     8.2
## 9          18    NA     8.0
## 10         20    NA     7.5
```

Với các `student_id` không có giá trị trên bảng x, cột maths sẽ trả về giá trị NA

Lưu ý: Trong trường hợp cột dữ liệu dùng để nối các bảng có tên khác nhau, ta có thể sử dụng cấu trúc sau:

---

left_join(x, y, by = c("key_x" = "key_y"))

Xem ví dụ sau:

```
names(x)[1] <- "student_id1"
names(y)[1] <- "student_id2"

x %>%
  inner_join(y, by = c("student_id1" = "student_id2"))
```

```
##   student_id1 maths physics
## 1           2   8.0     8.0
## 2           4   6.0     9.5
## 3           6   4.0     7.5
## 4           8   9.0     6.0
## 5          10   6.5     5.5
```

3.2.2.0.3 Ghép nhiều bảng theo dòng hoặc cột với nhóm hàm `bind`

Bên cạnh các hàm `join`, khi xử lý dữ liệu trong thực tiễn, ta có thể phải *ghép* các bảng dữ liệu theo hàng hoặc cột. Trong dplyr, có hai hàm rất hữu dụng trong hai trường hợp trên là `bind_col` và `bind_rows`

---

bind_cols(data1, data2) bind_rows(data1, data2)

Xem hai ví dụ sau.

```
df1 <- data.frame(id = 1:3,
                  income = 8:10)
df2 <- data.frame(id = 4:9,
                  income = 8:13)
df3 <- data.frame(id = 1:3,
                  gender = c("F", "F", "M"))

# Nối theo dòng
df1 %>% bind_rows(df2)
```

```
##   id income
## 1  1      8
## 2  2      9
## 3  3     10
## 4  4      8
## 5  5      9
## 6  6     10
## 7  7     11
## 8  8     12
## 9  9     13
```

```
# Nối theo cột
df1 %>% bind_cols(df3)
```

```
##   id income id1 gender
## 1  1      8   1      F
## 2  2      9   2      F
## 3  3     10   3      M
```

3.2.3 Nhóm hàm tổng hợp dữ liệu với `summarise`

Trong quá trình xử lý dữ liệu, ta thường xuyên phải tổng hợp dữ liệu theo các cách như: tính tổng, tính số dư bình quân, phương sai, tổng số lượng quan sát... Với `dplyr`, ta có thể sử dụng hàm `summarise()` để thực hiện công việc này.

---

data %>% summarise(var_name = calculate_stats(var))

```
mtcars %>%
  summarise(mean_mpg = mean(mpg),
            sd_mpg = sd(mpg))
```

```
##   mean_mpg sd_mpg
## 1    20.09  6.027
```

Đây là ví dụ đơn giản nhất với `summarise` mà ta có thể thay thế bằng `summary()` trên R base. Tuy nhiên, kết hợp giữa hàm `summarise()` và hàm group_by() trên dplyr sẽ cho chúng ta có cái nhìn về dữ liệu tổng hợp một cách đa chiều hơn. Hàm group_by() cho phép dữ liệu tổng hợp được gộp lại theo một hoặc nhiều trường thông tin khác nhau, giúp người phân tích có thể nhìn dữ liệu theo từ chiều riêng biệt hoặc gộp các chiều thông tin với nhau.

```
mtcars %>%
  group_by(cyl) %>%
  summarise(mean_mpg = mean(mpg),
            mean_disp = mean(disp))
```

```
## # A tibble: 3 x 3
##     cyl mean_mpg mean_disp
##   <dbl>    <dbl>     <dbl>
## 1    4.     26.7      105.
## 2    6.     19.7      183.
## 3    8.     15.1      353.
```

3.3 Các hàm nâng cao trong dplyr

Bên cạnh các nhóm hàm cơ bản đã trình bày ở phần trên, `dplyr` còn có một số hàm nâng cao khác đặc biệt hữu dụng trong quá trình biến đổi, tổng hợp dữ liệu, bao gồm `case_when`, `mutate_at` & `summarise_at`

3.3.1 Điều kiện phân nhóm với `case_when`

Trong quá trình phân tích và xử lý dữ liệu, chúng ta thường phải tạo thêm các trường mới hoặc tính toán dữ liệu dựa vào từng điều kiện khác nhau để đưa ra giá trị của trường hoặc cách tính cho dữ liệu. Ví dụ, khi ta muốn tính thưởng cho KH thì sẽ phải dùng nhiều công thức khác nhau như KH thuộc VIP sẽ nhân 1 tỷ lệ, KH thuộc nhóm trung bình sẽ có 1 tỷ lệ khác, hay KH thông thường thì sẽ 1 tỷ lệ khác….

Trong dplyr, hàm `case_when()` xử lý các trường hợp trên rất nhanh chóng.

---

data %>% mutate(new_var = case_when(condition_1 ~ "value_1", condition_2 ~ "value_2",..., TRUE ~ "value_n"))

Ta xem ví dụ sau:

```
df <- data.frame(number = 1:10)

df %>% mutate(nhom = case_when(
  number <= 5 ~ "nhom_1", # nhóm 1: số từ 1 đến 5
  number > 5 & number <= 8 ~ "nhom_2", # nhóm 2: số từ 6 đến 8
  TRUE ~ "nhom_3" # các số còn lại
  ))
```

```
##    number   nhom
## 1       1 nhom_1
## 2       2 nhom_1
## 3       3 nhom_1
## 4       4 nhom_1
## 5       5 nhom_1
## 6       6 nhom_2
## 7       7 nhom_2
## 8       8 nhom_2
## 9       9 nhom_3
## 10     10 nhom_3
```

3.3.2 Tạo thêm biến mới theo điều kiện với `mutate_if` & `mutate_at`

Khi phân tích, ta có thể tạo thêm biến mới khi các biến trong dataframe thỏa mãn điều kiện nào đó.

---

data %>% mutate_if(condition, function)

```
df <- data.frame(
  id = 1:5,
  gender = c("F", "M", "M", "F", "F"),
  income = c(4,5,3,6,7)
)
df %>% summary
```

```
##        id    gender     income
##  Min.   :1   F:3    Min.   :3
##  1st Qu.:2   M:2    1st Qu.:4
##  Median :3          Median :5
##  Mean   :3          Mean   :5
##  3rd Qu.:4          3rd Qu.:6
##  Max.   :5          Max.   :7
```

```
# Biến đổi các biến factor thành character
df %>% mutate_if(is.factor,
                 as.character) %>%
  summary
```

```
##        id       gender              income
##  Min.   :1   Length:5           Min.   :3
##  1st Qu.:2   Class :character   1st Qu.:4
##  Median :3   Mode  :character   Median :5
##  Mean   :3                      Mean   :5
##  3rd Qu.:4                      3rd Qu.:6
##  Max.   :5                      Max.   :7
```

Ngoài ra, ta có thể tự tạo các hàm mới và áp dụng với `mutate_if`. Xem ví dụ sau.

```
my_func <- function(x){x*100}
# Nhân các biến numeric lên 100 lần
df %>%
  mutate_if(is.numeric, my_func)
```

```
##    id gender income
## 1 100      F    400
## 2 200      M    500
## 3 300      M    300
## 4 400      F    600
## 5 500      F    700
```

Đối với `mutate_at`, ta cũng có thể thực hiện tương tự. Cấu trúc tổng quát của `mutate_at` như sau.

data %>% mutate_at(vars(var1, var2, ...), function)

```
df
```

```
##   id gender income
## 1  1      F      4
## 2  2      M      5
## 3  3      M      3
## 4  4      F      6
## 5  5      F      7
```

```
# Nhân biến income lên 100 lần
df %>% mutate_at(vars(income), my_func)
```

```
##   id gender income
## 1  1      F    400
## 2  2      M    500
## 3  3      M    300
## 4  4      F    600
## 5  5      F    700
```

3.3.3 Tổng hợp dữ liệu theo điều kiện với `summarise_at` và `summarise_if`

Tương tự như `mutate_at` và `mutate_if`, ta có thể tổng hợp nhanh dữ liệu theo điều kiện.

Cấu trúc tổng quát của `summarise_at`

---

data %>% group_by(var) (không bắt buộc) summarise_at(vars(variables), funs(functions))

Xem ví dụ sau

```
mtcars %>%
  group_by(am) %>%
  summarise_at(vars(mpg, disp),
               funs(mean, max, median))
```

```
## # A tibble: 2 x 7
##      am mpg_mean disp_mean mpg_max disp_max mpg_median
##   <dbl>    <dbl>     <dbl>   <dbl>    <dbl>      <dbl>
## 1    0.     17.1      290.    24.4     472.       17.3
## 2    1.     24.4      144.    33.9     351.       22.8
## # ... with 1 more variable: disp_median <dbl>
```

Tương tự, ta có cấu trúc tổng quát của `summarise_if`

---

data %>% group_by(var) (không bắt buộc) summarise_if(condition, funs(functions))

```
iris %>%
  select(Species, Sepal.Length, Sepal.Width) %>%
  group_by(Species) %>%
  summarise_if(is.numeric,
               funs(mean, median))
```

```
## # A tibble: 3 x 5
##   Species    Sepal.Length_mean Sepal.Width_mean
##   <fct>                  <dbl>            <dbl>
## 1 setosa                  5.01             3.43
## 2 versicolor              5.94             2.77
## 3 virginica               6.59             2.97
## # ... with 2 more variables:
## #   Sepal.Length_median <dbl>,
## #   Sepal.Width_median <dbl>
```

4

Phân rã và xoay chiều dữ liệu

Khi phân tích dữ liệu, dữ liệu sau khi được làm sạch thường cơ bản có hai dạng.

- Dạng ngang: Mỗi dòng ứng với 1 quan sát và nhiều biến
- Dạng dọc: Nhiều dòng có thể chứa cùng một quan sát nhưng với các biến khác nhau.

Xem hai ví dụ về dữ liệu dạng ngang và dọc ở dưới đây.

id	Species	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
1	setosa	5.1	3.5	1.4	0.2
2	setosa	4.9	3.0	1.4	0.2
3	setosa	4.7	3.2	1.3	0.2
4	setosa	4.6	3.1	1.5	0.2
5	setosa	5.0	3.6	1.4	0.2
6	setosa	5.4	3.9	1.7	0.4

id	Species	Measurement	Value
1	setosa	Sepal.Length	5.1
1	setosa	Sepal.Width	3.5
1	setosa	Petal.Length	1.4
1	setosa	Petal.Width	0.2
2	setosa	Sepal.Length	4.9
2	setosa	Sepal.Width	3.0
2	setosa	Petal.Length	1.4
2	setosa	Petal.Width	0.2

Trong thực tế, chúng ta phải sử dụng rất linh hoạt cả hai định dạng dữ liệu này (dữ liệu ngang và dữ liệu dọc). Trong chương này, chúng ta sẽ học cách sử dụng và biến đổi dữ liệu giữa hai định dạng với `tidyr`.

4.1 Phân rã dữ liệu thành dạng dọc với gather

```
library(tidyr)
data %>%
  gather(key = name_of_key,
         value = name_of_value_variable,
         gather = c(list_of_var))
```

Xem ví dụ sau.

```
library(dplyr)
df <- iris %>%
  head(2) %>%
  mutate(id = 1:nrow(.)) %>%
  select(6, 5, 1:4)
df
```

```
##   id Species Sepal.Length Sepal.Width Petal.Length
## 1  1  setosa          5.1         3.5          1.4
## 2  2  setosa          4.9         3.0          1.4
##   Petal.Width
## 1         0.2
## 2         0.2
```

```
# Xoay dữ liệu sang dạng dọc
df2 <- df %>%
  gather(key = Measurement,
       value = Value,
       c(3:6)) # Các biến được phân rã
df2
```

```
##   id Species  Measurement Value
## 1  1  setosa Sepal.Length   5.1
## 2  2  setosa Sepal.Length   4.9
## 3  1  setosa  Sepal.Width   3.5
## 4  2  setosa  Sepal.Width   3.0
## 5  1  setosa Petal.Length   1.4
## 6  2  setosa Petal.Length   1.4
## 7  1  setosa  Petal.Width   0.2
## 8  2  setosa  Petal.Width   0.2
```

Ở ví dụ trên, khi phân rã dữ liệu sang dạng dọc, các biến được phân rã là 4 biến ở vị trí từ `3 đến 6`. Do dữ liệu gốc `df` chỉ có 2 quan sát, nên dữ liệu mới sau khi phân rã sẽ có 8 quan sát.

4.2 Xoay chiều dữ liệu với spread

Ngược lại với phân rã dữ liệu là xoay chiều dữ liệu. Trong `tidyr`, ta có thể sử dụng hàm `spread`. Công thức tổng quát để xoay chiều dữ liệu như sau.

```
data %>%
  #Biến được xoay thành cột
  spread(key = key_variable,
         # Biến giá trị
         value = value_variable)
```

Ta quay trở lại ví dụ ở phần trước với dữ liệu `df2` đã được phân rã.

```
df2
```

```
##   id Species  Measurement Value
## 1  1  setosa Sepal.Length   5.1
## 2  2  setosa Sepal.Length   4.9
## 3  1  setosa  Sepal.Width   3.5
## 4  2  setosa  Sepal.Width   3.0
## 5  1  setosa Petal.Length   1.4
## 6  2  setosa Petal.Length   1.4
## 7  1  setosa  Petal.Width   0.2
## 8  2  setosa  Petal.Width   0.2
```

Ta có thể xoay chiều dữ liệu lại như sau.

```
df2 %>%
  spread(key = Measurement,
         value = Value)
```

```
##   id Species Petal.Length Petal.Width Sepal.Length
## 1  1  setosa          1.4         0.2          5.1
## 2  2  setosa          1.4         0.2          4.9
##   Sepal.Width
## 1         3.5
## 2         3.0
```

4.3 Tách một biến thành nhiều biến với `separate`

Khi phân tích dữ liệu, ta thường xuyên phải tách một biến thành nhiều biến. Khi đó, việc tách biến sẽ trở nên rất đơn giản với hàm `separate`. Công thức tổng quát của `separate` như sau:

```
data %>%
  separate(var_to_spread, c("new_var1",
                            "new_var2", ...))
```

```
df <- data.frame(date = c(NA,
                          "2018-07-01",
                          "2018-09-02"))
df
```

```
##         date
## 1       <NA>
## 2 2018-07-01
## 3 2018-09-02
```

```
# Tách biến date thành 3 biến
df %>%
  separate(date, c("year", "month", "date"))
```

```
##   year month date
## 1 <NA>  <NA> <NA>
## 2 2018    07   01
## 3 2018    09   02
```

4.4 Gộp nhiều biến thành một biến với `unite`

Ngược lại với `spread`, ta có thể gộp nhiều biến thành một với `unite`. Công thức tổng quát của `unite` như sau.

```
data %>%
  unite(new_var, var_1, var2,...)
```

Trong đó, `var_1`, `var_2` là tên các biến sẽ được gộp. `new_var` là tên biến mới được tạo thành.

Quay trở lại ví dụ trên.

```
df <- data.frame(date = c("2018-07-01", "2018-09-02")) %>%   separate(date, c("year", "mon
df
```

```
##   year month date
## 1 2018    07   01
## 2 2018    09   02
```

```
# Gộp nhiều biến
df %>%
  unite(full_date, 1:3,
        sep = "/",
        remove = F)
```

```
##    full_date year month date
## 1 2018/07/01 2018    07   01
## 2 2018/09/02 2018    09   02
```

5

Lập trình chức năng hàm với purrr

Khi phân tích dữ liệu phức tạp, ta thường xuyên phải thực hiện một nhóm các phân tích tương tự nhau cho các nhóm dữ liệu khác nhau. Việc sử dụng các hàm làm đơn vị thao tác cơ bản và phối hợp các hàm với nhau được gọi là lập trình chức năng hàm (functional programming). Để đơn giản, ta xét ví dụ sau.

Sử dụng tập dữ liệu `iris`, với mỗi nhóm của `Species`, xây dựng mô hình hồi quy giữa `Sepal.Length` và `Petal.Length`, so sánh giá trị `r.squared` giữa các mô hình.

Với cách làm thông thường, ta sẽ phải thức hiện theo thứ tự sau:

- Tạo các data.frame cho từng giá trị của Species
- Với mỗi data.frame vừa tạo, xây dựng mô hình `lm`
- Với mỗi mô hình vừa tạo, chiết xuất giá trị `r.squared` và lưu vào một data.frame

Cách triển khai trên có thể sử dụng vòng lặp trong R với phương án như sau

```
library(dplyr)

category <- iris$Species %>% levels %>% as.character()
model_result <- data.frame()
for (i in category){
  df <- iris %>% filter(Species == i)
  model <- lm(Sepal.Length ~ Sepal.Width, data = df)
  model_summary <- summary(model)
  df_temp <- data.frame(species = i,
                        r.square = model_summary$r.squared)
  model_result <- bind_rows(model_result, df_temp)
}
```

Tuy nhiên, với lập trình chức năng hàm, ta có thể làm rất đơn giản như sau.

```
library(purrr)
iris %>%
  split(.$Species) %>%
  map(~lm(Sepal.Length ~ Sepal.Width, data = .)) %>%
```

```
  map(summary) %>%
  map_dbl("r.squared")
```

```
##     setosa versicolor  virginica
##     0.5514     0.2766     0.2091
```

Trong chương này, chúng ta sẽ tìm hiểu các cách thức cơ bản lập trình chức năng hàm với R qua package `purrr`. Việc nắm vững kiến thức và kỹ năng lập trình hàm có rất nhiều ứng dụng trong công việc phân tích, giúp giảm thiểu rất lớn thời gian phân tích, làm cho quá trình phân tích mạch lạc hơn rất nhiều trong các bài toán khám phá dữ liệu

5.1 Nhóm hàm map

Công thức tổng quát của nhóm hàm map

```
map(.x, .f, ...)
```

Giải thích: Với mỗi giá trị của `.x`, thực hiện `.f`. Trong đó, x là một list.

Hàm map làm hàm tổng quát, ngoài ra, `map` còn có các biến thể chính sau

Câu lênh	Kết quả
map	list
map_dbl	vector dạng double
map_int	vector dạng int
map_chr	vector dạng character
map_df	data.frame

```
# Dạng list
iris %>% map(class)
```

```
## $Sepal.Length
## [1] "numeric"
##
## $Sepal.Width
## [1] "numeric"
##
## $Petal.Length
## [1] "numeric"
##
```

```
## $Petal.Width
## [1] "numeric"
##
## $Species
## [1] "factor"
```

```
# Dạng char
iris %>% map_chr(class)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##    "numeric"    "numeric"    "numeric"    "numeric"
##      Species
##     "factor"
```

```
# Dạng data.frame
iris %>% map_df(class)
```

```
## # A tibble: 1 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
##   <chr>        <chr>       <chr>        <chr>
## 1 numeric      numeric     numeric      numeric
## # ... with 1 more variable: Species <chr>
```

Map theo điều kiện với map_if và map_at

Tương tự với `map`, nhóm `map_if` và `map_at` cho phép tính toán theo điều kiện hoặc vị trí của list. Xem ví dụ sau.

```
# map_if
iris %>%
  map_if(is.numeric, as.character) %>%
  as.data.frame %>%
  str
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: Factor w/ 35 levels "4.3","4.4","4.5",..: 9 7 5 4 8 12 4 8 2 7 ...
##  $ Sepal.Width : Factor w/ 23 levels "2","2.2","2.3",..: 15 10 12 11 16 19 14 14 9 11 .
##  $ Petal.Length: Factor w/ 43 levels "1","1.1","1.2",..: 5 5 4 6 5 8 5 6 5 6 ...
##  $ Petal.Width : Factor w/ 22 levels "0.1","0.2","0.3",..: 2 2 2 2 2 4 3 2 2 1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
# map_at
iris %>%
  map_at(c(1,2), as.character) %>%
  str
```

```
## List of 5
##  $ Sepal.Length: chr [1:150] "5.1" "4.9" "4.7" "4.6" ...
##  $ Sepal.Width : chr [1:150] "3.5" "3" "3.2" "3.1" ...
```

```
##  $ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num [1:150] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

5.2 Sửa đổi giá trị với `modify`

Tương tự như `map`, `modify` cho áp dụng hàm vào một nhóm các list. Tuy nhiên, khác với `map`, `modify` cho ra kết quả với cấu trúc dữ liệu ban đầu.

```
# map đổi cấu trúc của dataframe
iris %>%
  map_if(is.factor, as.character) %>%
  str
```

```
## List of 5
##  $ Sepal.Length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num [1:150] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num [1:150] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : chr [1:150] "setosa" "setosa" "setosa" "setosa" ...
```

```
# modify giữ nguyên cấu trúc
iris %>%
  modify_if(is.factor, as.character) %>%
  str
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : chr  "setosa" "setosa" "setosa" "setosa" ...
```

5.3 Ứng dụng

5.3.1 Biến đổi dữ liệu với `modify` và `map_df`

Khi phân tích dữ liệu, đôi khi ta cần chuẩn hóa dữ liệu cho tất cả các biến numeric trong data.frame. Với nhóm hàm của purrr, ta có thể thực hiện như sau

```
# Tạo hàm
standardize_data <- function(x){
  x <- (x - min(x))/(max(x) - min(x))
  return(x)
}
# Sử dụng map_df
df <- iris
df[, 1:4] <- df[, 1:4] %>% map_df(standardize_data)
df %>% head
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1      0.22222      0.6250      0.06780     0.04167
## 2      0.16667      0.4167      0.06780     0.04167
## 3      0.11111      0.5000      0.05085     0.04167
## 4      0.08333      0.4583      0.08475     0.04167
## 5      0.19444      0.6667      0.06780     0.04167
## 6      0.30556      0.7917      0.11864     0.12500
##   Species
## 1  setosa
## 2  setosa
## 3  setosa
## 4  setosa
## 5  setosa
## 6  setosa
```

```
# Sử dụng modify
# Sử dụng map_df
df <- iris
df <- df %>% modify_if(is.numeric, standardize_data)
df %>% head
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1      0.22222      0.6250      0.06780     0.04167
## 2      0.16667      0.4167      0.06780     0.04167
## 3      0.11111      0.5000      0.05085     0.04167
## 4      0.08333      0.4583      0.08475     0.04167
```

```
## 5          0.19444       0.6667        0.06780     0.04167
## 6          0.30556       0.7917        0.11864     0.12500
##    Species
## 1   setosa
## 2   setosa
## 3   setosa
## 4   setosa
## 5   setosa
## 6   setosa
```

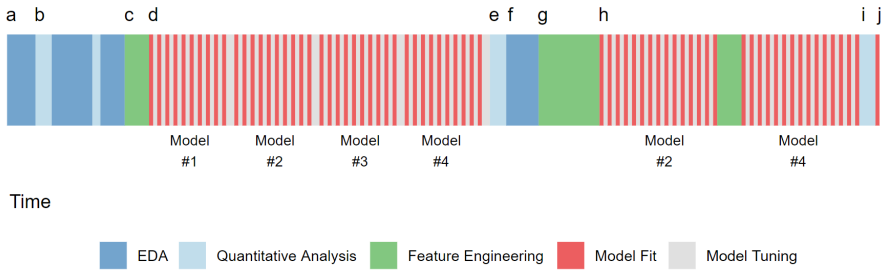
Phần III

Machine Learning

6

Nguyên lý dự báo

Quá trình dự báo: Trong thực tế, quá trình dự báo diễn ra như sau



6.1 Các nhóm thuật ngữ cần nhớ

- **Độ biến động (variance)** vs. **độ chuẩn xác (bias)**:
  - `variance` là độ biến động của mô hình so với dữ liệu, đo độ nhạy cảm của các tham số trong mô hình khi thay đổi dữ liệu. Nói cách khác, một mô hình được gọi là biến động lớn khi dữ liệu mô hình thay đổi sẽ dẫn đến một sự thay đổi lớn trong mô hình. Ví dụ, mô hình sử dụng trung vị làm biến dự báo sẽ ít biến động hơn khi sử dụng giá trị trung bình.
  - `bias` đo độ `chuẩn xác` của mô hình. Một mô hình được gọi là `bias` thấp khi kết quả dự báo gần với kết quả thực tế và ngược lại. Các mô hình có `bias` thấp có khả năng thích ứng với dữ liệu tốt, các mô hình như cây quyết đinh, neural network thuộc dạng này.

6.2 Lưu ý

- Khi xây dựng mô hình, hiệu ứng của các biến dự báo (predictor) có thể lớn hơn thuật toán rất nhiều
- Với cùng một nhóm các biến dự báo đúng thực tế, các thuật toán khác nhau có thể đưa ra các kết quả tương tự nhau.

7

Feature Engineering

Khi xây dựng mô hình dự báo, ta phải cân bằng giữa độ chính xác và khả năng giải thích của mô hình. Trong nhiều trường hợp, khả năng giải thích được ưu tiên hơn, thể hiện rất rõ trong các mô hình score card của rủi ro trong hệ thống ngân hàng. Tuy nhiên, ta không thể hy sinh độ chính xác để lấy khả năng giải thích nếu độ chính xác của mô hình không đạt đến một ngưỡng nhất định. Khi đó, có hai cách tiếp cận:

- Cho thêm biến vào mô hình.
- Thay đổi các biến có sẵn bằng các nhóm biến phái sinh để mô hình tốt hơn - cách tiếp cận này gọi là `feature engineering`.

`Feature Engineering` là quá trình thể hiện các biến đầu vào (variables) với những cách thức khác nhau để giúp cho tăng độ chính xác của mô hình dự báo. Ví dụ:

- Địa điểm của khách hàng có thể được thể hiện bằng ZIP code hoặc cùng lúc 2 biến, kinh độ và vĩ độ.
- Biến dự báo `age` có thể cho ra kết quả tốt hơn trong mô hình khi ta dùng biến $\frac{1}{age}$

Với mỗi mô hình, thuật toán khác nhau sẽ yêu cầu những cách thức triển khai và thay đổi biến đầu vào khác nhau. Do đó, khi lựa chọn cách thức biến đổi biến đầu vào, ta phải nắm rất rõ thuật toán và cách thức biến đổi dữ liệu theo từng trường hợp khác nhau. Đối với `feature engineering`, ta có thể chia làm 2 nhóm chính.

- Biến đổi các biến định dạng nhóm (categorical)
- Biến đổi các biến định dạng số (numeric)

7.1 Feature engineering cho các biến nhóm

7.1.1 Tạo dữ liệu giả (dummy data) cho biến không phân biệt thứ tự

Trong phương pháp này, toàn bộ các dữ liệu gốc được chuyển sang dạng 0-1. Tuy nhiên, dữ liệu mới được tạo ra sẽ ít hơn dữ liệu gốc 1 trường hợp. Bởi lẽ khi biết giá trị của 6 biến, ta có thể biết được giá trị của biến cuối cùng.

Biến gốc	Mon	Tues	Wed	Thurs	Fri	Sat
Sun	0	0	0	0	0	0
Mon	1	0	0	0	0	0
Tues	0	1	0	0	0	0
Wed	0	0	1	0	0	0
Thurs	0	0	0	1	0	0
Fri	0	0	0	0	1	0
Sat	0	0	0	0	0	1

zero-variance predictor: là biến chỉ có một giá trị. Khi xây dựng mô hình, ta cần loại biến này.

7.1.2 Dữ liệu có rất nhiều nhóm

Đối với các biến có rất nhiều nhóm (ví dụ: 200 chi nhánh trong ngân hàng), ta có 2 cách tiếp cận.

- Cách một, dựa vào kiến thức nghiệp vụ tự nhóm. Ví dụ, các chi nhánh ở Hà Nội sẽ đánh dấu là HN, ở Hồ Chí Minh là HCM, các chi nhánh còn lại là `Others`.
- Cách hai, sử dụng `hash function`. Trong trường hợp này, các biến category sẽ được tạo thành một biến hoàn toàn mới có giá trị số. Xem ví dụ dưới đây.

Giá trị	Hash
belvedere tiburon	582753783
berkeley	1166288024

Lưu ý: Nhiều thí nghiệm đã được sử dụng để so sánh sự khác biệt giữa việc

dùng factor và encoding 0-1 trong dữ liệu. Kết quả cho thấy không có nhiều sự khác biệt giữa hai cách.

7.2 Các biến liên tục

Đối với các biến liên tục, khi xây dựng mô hình, ta sẽ gặp phải các vấn đề sau.

- Các biến có các đơn vị khác nhau. Ví dụ, tuổi có giá trị từ 15-75, thu nhập có giá trị từ 2 triệu VND đến 200 triệu VND
- Các biến bị lệch sang phải (skewness)
- Các biến có xuất hiện giá trị ngoại lai (outliers)
- Các biến có thể bị chặn trai hoặc chặn phải. Ví dụ, độ tuổi có giá trị không quá 80

Đối với các biến số, có ba nhóm kỹ thuật lớn biến đổi dữ liệu.

- Biến đổi 1:1 - một biến được biến đổi thành một biến khác
- Biến đổi 1:n - một biến được biến đổi thành nhiều biến khác nhau
- Biến đổi n:n - n biến gốc được biến đổi cùng lúc thành n biến khác

7.2.1 Biến đổi 1:1

Trong biến đổi 1:1, có rất nhiều cách khác nhau.

- Biến đổi theo scale của dữ liệu: log, Box-Cox

$$x^* = \begin{cases} \frac{x^\lambda - 1}{\lambda \tilde{x}^{\lambda-1}}, & \lambda \neq 0 \\ \tilde{x} \log x, & \lambda = 0 \end{cases}$$

Phần IV

Chuỗi thời gian

8

Giới thiệu về chuỗi thời gian

Khi phân tích dữ liệu, có hai khái niệm đều được gọi là *dự báo* khi được dịch sang tiếng Việt, đó là `forcast` và `predict`. Tuy nhiên, hai khái niệm này rất khác nhau.

- **Predict**: Thường được dùng để chỉ việc dự báo xác suất xảy ra các sự kiện. Ví dụ, xác suất vỡ nợ, xác suất khách hàng churn,...
- **Forecast**: Thường dùng trong việc dự báo chuỗi thời gian. Ví dụ, dựa vào lịch sử biến động của tổng khách hàng uống cafe theo tuần, ta có thể *dự báo* được số lượng khách hàng uống cafe của 1 tuần tới, hai tuần tới,...

Trong phần trước, chúng ta đã bàn nhiều về nhóm `predict`, trong phần này, ta sẽ bàn thêm về nhóm `forecast`, về các cấu phần và cách thức dự báo đối với chuỗi thời gian. Do đó, trong phần này, khi nói về dự báo, chúng ta đang bàn về vấn đề `forecast`.

Phân biệt `dự báo`, `mục tiêu` và `kế hoạch`

Dự báo: là quá trình sử dụng các thông tin hiện hữu đưa ra các nhận định chính xác nhất có thể có trong tương lai trong một khoảng thời gian xác định về một chỉ số nào đó.

Mục tiêu: là thứ mà cá nhân, tổ chức mong muốn đạt được trong một khoảng thời gian xác định trong tương lai. Thường thì mục tiêu được đặt mà không quan tâm đến bất kỳ đến việc dự báo nào cả. Ví dụ, mục tiêu tăng trưởng của doanh nghiệp thường là năm sau cao gấp đôi năm trước trong khi dự báo chỉ có thể tăng được 30%.

Kế hoạch: Là phản ứng của tổ chức, cá nhân đối với *dự báo* và *mục tiêu*. Việc lập kế hoạch đòi hỏi nhiều hành động cụ thể để điều hướng dự báo sát sát với mục tiêu (hoặc vượt mục tiêu)

Xét về yếu tố thời gian, việc dự báo có thể chia thành dự báo ngắn hạn, trung hạn và dài hạn.

Các điểm khi dự báo

Khi dự báo, có hai điểm quan trọng chúng ta phải trả lời.

- Thứ nhất, ta cần dự báo điều gì? Dự báo với từng sản phẩm hay với cả

nhóm sản phẩm? Dự báo doanh số bán hàng của từng cửa hàng hay của toàn hệ thống?
- Thứ hai, yếu tố thời gian xét trong vấn đề dự báo này là gì? Ta cần dự báo trong bao lâu? Tần xuất như thế nào? Ví dụ, dư báo doanh số bán hàng mỗi tháng/tuần 1 lần trong 1 năm tới?

Lưu ý: Các bạn phân tích dữ liệu cần phải tìm hiểu các đơn vị nghiệp vụ sẽ sử dụng kết quả dự báo như thế nào, để tránh việc bỏ quá nhiều thời gian và công sức dự báo nhưng không ai sử dụng.

8.1 Thành phần của chuỗi thời gian

Trong bất cứ chuỗi thời gian nào, cũng có 3 thành phần sau.

- **Xu hướng** (trend) thể hiện chiều hướng tăng hay giảm dài hạn của chuỗi thời gian
- **Mùa vụ** (seasonal) thể hiện sự biến đổi của chuỗi thời gian theo chu kỳ biết trước. Ví dụ, vào cuối tuần, khách hàng có xu hướng đi ăn nhà hàng nhiều hơn.
- **Chu kỳ kinh doanh** (cyclic) thể hiện xu hướng biến đổi dài hạn của chuỗi thời gian, thường ít nhất hai năm. Chu kỳ kinh doanh khác với yếu tố mùa vụ ở chỗ, chu kỳ biến đổi của yếu tố mùa vụ thường là đã được biết trước và mang tính ngắn hạn.

Phần V

Case study

Quá trình phân tích dữ liệu không phải chỉ là việc áp dụng lý thuyết một cách máy móc và đòi hỏi sự sáng tạo và linh hoạt trong từng trường hợp cụ thể và rất khác nhau trong từng lĩnh vực. Ở phần này, tác giả sẽ trình bày các ứng dụng thực tiễn theo nhiều lĩnh vực khác nhau,

9

Trực quan hóa dữ liệu

9.1 Xây dựng phễu bán hàng theo từng nhóm

Trong quá trình phân tích bán hàng, phếu bán hàng (sale funnel) là một kỹ thuật rất hữu dụng để trực quan hóa kết quả kinh doanh theo từng nhóm. Tuy nhiên, hiện ít có biểu đồ nào thể hiện được phễu bán hàng một cách hiệu quả trên R.

Trong mục này, tác giả sẽ hướng dẫn một ví dụ thực tiễn trực quan hóa phễu bán hàng một cách hiệu quả.

Xem ví dụ điển hình về phễu bán hàng dưới đây

```
data <- read.table(textConnection(
          c("step;segment1;segment2;segment3;total
          1_visit;1806;11663;12641;26110
          2_register;1143;6476;5372;12991
          3_login;1806;11663;2694;16163
          4_subscribe;21;3322;2694;6037
          5_paid;259;422;41;722")),
        header = T, sep = ";")
# Dữ liệu
data
```

```
##                   step segment1 segment2 segment3
## 1              1_visit     1806    11663    12641
## 2           2_register     1143     6476     5372
## 3              3_login     1806    11663     2694
## 4          4_subscribe       21     3322     2694
## 5               5_paid      259      422       41
##   total
## 1 26110
## 2 12991
## 3 16163
## 4  6037
## 5   722
```

Trong tập dữ liệu trên, ta sẽ mô phỏng dữ liệu phễu bán hàng của 3 phân khúc khách hàng trên một trang thương mại điện tử mà trong đó, khách hàng sẽ đi qua năm bước khác nhau:

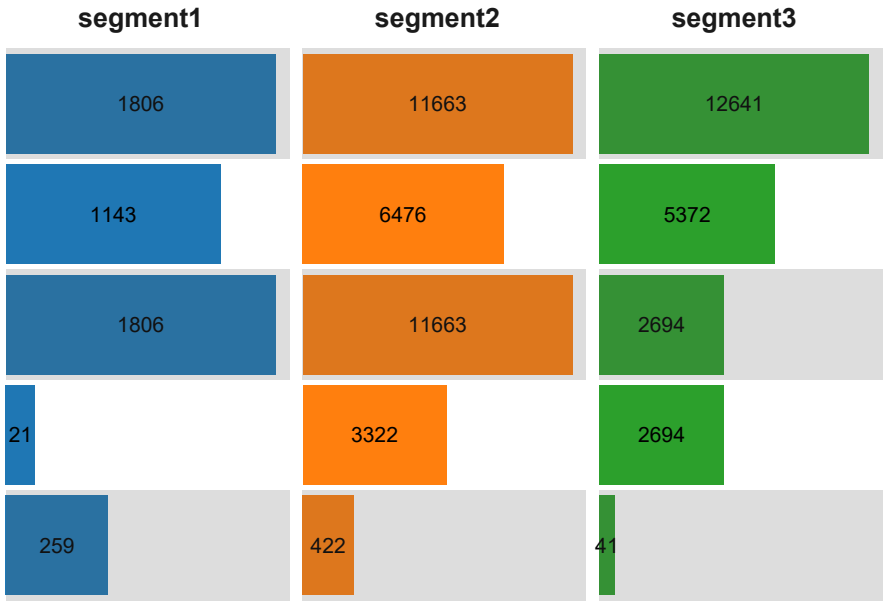
- Ghé thăm website (visit)
- Đăng ký (register)
- Đăng nhập (login)
- Đăng ký cập nhật các thông tin sản phẩm (subscribe)
- Mua hàng và trả tiền thành công (paid)

Để tạo một biểu đồ phễu bán hàng, ta sẽ thực hiện 3 bước lớn sau.

- Tạo `theme` cho biểu đồ
- Tạo các biểu đồ con cho phễu bán hàng
- Kết hợp các biểu đồ để tạo thành phễu bán hàng hoàn chỉnh

```
# Gọi library
library(tidyverse)
library(reshape2)
library(forcats)
library(ggthemes)

# Tạo theme trông cho chart
funnel_theme <- theme(axis.title = element_blank(),
        axis.ticks.x = element_blank(),
        axis.text.x = element_blank(),
        legend.position = "none",
        panel.grid = element_blank()
        )

# Phân rã dữ liệu
df <- data %>% melt(id.vars = "step")

# Tạo biểu đồ chính
p1 <- df %>%
  mutate(step = fct_rev(step)) %>%
  filter(variable != "total") %>%
  ggplot(aes(step, value)) +
  geom_bar(aes(fill = variable), stat = "identity") +
  facet_grid(~variable, scale = "free") +
  coord_flip() +
  geom_text(aes(label = value),
            position = position_stack(vjust = .5)) +
  scale_fill_tableau() +
  theme_minimal() +
  scale_y_sqrt() +
  funnel_theme +
```

```
  theme(plot.margin=grid::unit(c(0,0,0,0), "mm")) +
  theme(
  axis.text.y = element_blank(),
  strip.text = element_text(size = 14,
                            face = "bold")) +
  theme(
    panel.spacing = unit(0, "mm")) +
  annotate("rect", xmin = 0.5, xmax = 1.5, ymin = 0, ymax = Inf,
           alpha = .2) +
  annotate("rect", xmin = 2.5, xmax = 3.5, ymin = 0, ymax = Inf,
           alpha = .2) +
  annotate("rect", xmin = 4.5, xmax = 5.5, ymin = 0, ymax = Inf,
           alpha = .2) +
  theme(axis.text.y = element_blank())
p1
```



- Tạo thêm phần `label` tổng theo từng segment

```
df %>%
  mutate(step = fct_rev(step)) %>%
  filter(variable == "total") %>%
  ggplot(aes(step, 0)) +
  geom_label(aes(label = value),
             col = "white",
             fill = "darkred",
```

```
              size = 4) +
  coord_flip() +
  facet_wrap(~variable) +
  theme_minimal() +
  theme(axis.text = element_blank()) +
  funnel_theme +
  theme(
    strip.text.x = element_blank()
  ) -> p2
p2
```

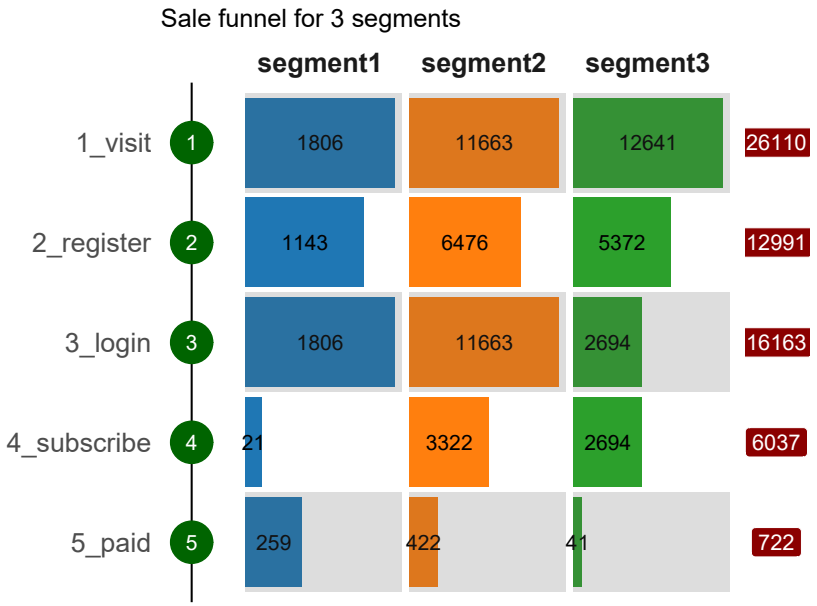
- Tạo thêm thứ tự các bước trong phễu bán hàng để dễ theo dõi hơn

```
df2 <- data.frame(step = data$step,
                  value = 1:5)
df2 %>%
  mutate(step = fct_rev(step)) %>%
  ggplot(aes(step, 1)) +
  geom_hline(yintercept = 1) +
  geom_point(size = 10, col = "darkgreen") +
  geom_text(aes(label = value),
            col = "white") +
  coord_flip() +
  theme_minimal() +
  funnel_theme +
```

```
  theme(
    axis.text = element_text(size = 14)
  ) -> p3
p3
```

- Cuối cùng, ta có thể tạo ghép các biểu đồ rời rạc để tạo thành phễu bán hàng hoàn chỉnh. Việc kết hợp các biểu đồ trên `ggplot2` có thể hoàn thành một cách đơn giản với `ggplot2`

```
#devtools::install_github("thomasp85/patchwork")
library(patchwork)
p3 +
  labs(title = "Sale funnel for 3 segments") +
  p1 + p2 +
  plot_layout(nrow = 1, widths = c(1, 8, 1))
```



Như vậy, chúng ta đã hoàn thành phễu bán hàng rất chuyên nghiệp với `ggplot2`. Phễu bán hàng này đặc biệt hiệu quả khi cùng lúc phải so sánh nhiều phân khúc khách hàng khác nhau trên toàn bộ chuỗi bán hàng.

9.2 Vẽ biểu đồ warterfall cho acive/inactive users

Trong kỷ nguyên số, chỉ số `active user` (tạm dịch: người dùng thường xuyên hoạt động) là chỉ số đặc biệt quan trọng với bất kỳ website/ app nào. Công thức tính chỉ số người dùng thường xuyên hoạt động tại khoảng thời gian t được tính như sau:

$$active_t = active_{t-1} + new_t - churn_t$$

Ví dụ về waterfall chart được lấy từ ví dụ của Tableau tại đường link: [https://public.tableau.com/views/CH24_BBOD_ChurnTurnover/SubscriberChurnAnalysis]

Trong case study này, chúng ta sẽ tìm cách xây dựng một biểu đồ waterfall chart tương tự

```
# Load library
library(tidyverse)
```

```
library(ggplot2)
library(reshape2)
library(lubridate)
library(grid)
library(gridExtra)

# Tạo dữ liệu giả lập
set.seed(123)
data <- data.frame(date = seq(1, 372, by = 31) %>% as_date)
data <- data %>%
  mutate(new = abs(rnorm(12, 100, 10)) %>% round(0)) %>%
  mutate(churn = abs(rnorm(12, 50, 30)) %>% round(0)) %>%
  mutate(net = new - churn)  %>%
  mutate(eop = cumsum(net)) %>%
  select(-net)

data
```

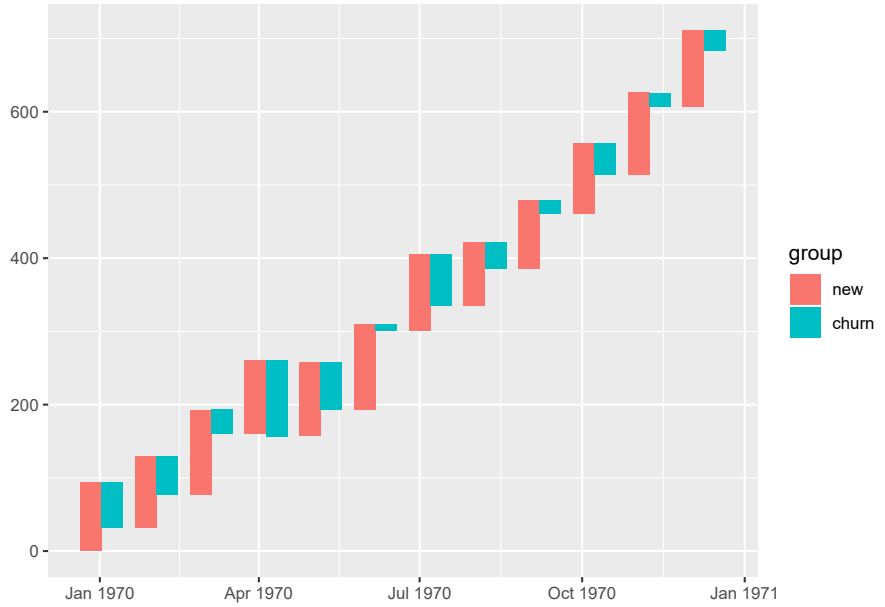
```
##          date new churn eop
## 1  1970-01-02  94    62  32
## 2  1970-02-02  98    53  77
## 3  1970-03-05 116    33 160
## 4  1970-04-05 101   104 157
## 5  1970-05-06 101    65 193
## 6  1970-06-06 117     9 301
## 7  1970-07-07 105    71 335
## 8  1970-08-07  87    36 386
## 9  1970-09-07  93    18 461
## 10 1970-10-08  96    43 514
## 11 1970-11-08 112    19 607
## 12 1970-12-09 104    28 683
```

Trong ví dụ này, dữ liệu được tạo ngẫu nhiên sao cho số lượng `active user` cuối kỳ (eop - end of period) bằng với số cuối kỳ trước, thêm số lượng mới và trừ đi lượng khách hàng rời bỏ (`churn`).

Để tạo waterfall chart, ta có thể sử dụng `geom_segment` trong ggplot2

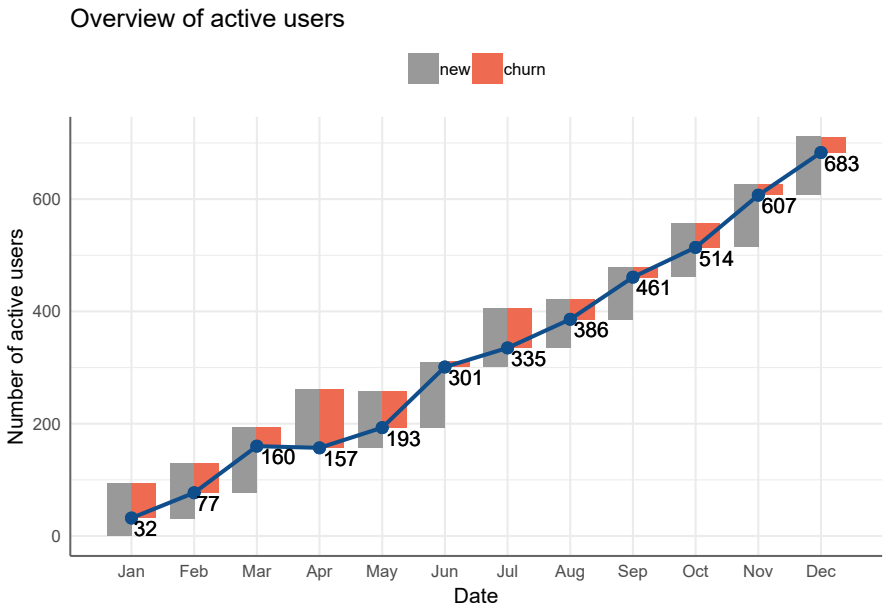
```
# Xác định độ rộng của segment
step <- 0.4*(max(data$date) - min(data$date))/(nrow(data) - 1)

# Xác định ymax
data <- data %>%
  mutate(ymax = eop + churn)

# Xác định ymin
```

```r
df <- data %>%
  melt(id.vars = c("date", "eop", "ymax")) %>%
  mutate(ymin = ymax - value) %>%
  rename(group = variable)

# Xác định xmin và xmax
df <- df %>%
  mutate(xmin = case_when(
    group == "new" ~ date - step,
    TRUE ~ date
  )) %>%
  mutate(xmax = case_when(
    group == "new" ~ date,
    TRUE ~ date + step
  ))


# Create waterfall chart
p1 <- df %>%
  arrange(date) %>%
  ggplot() +
  geom_rect(aes(xmin = xmin,
                xmax = xmax,
                ymin = ymin,
                ymax = ymax,
                fill = group))
p1
```



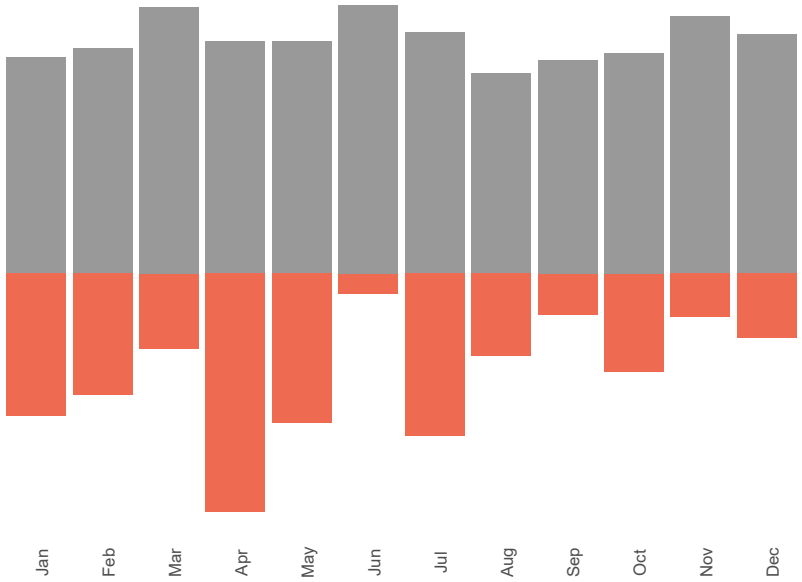
Như vậy, ta đã tạo xong biểu đồ water-fall đơn giản. Ở bước tiếp theo, chúng ta cần điều chỉnh lại các thành phần cho biểu đồ.

```
# Tạo dữ liệu cho biểu đồ đường
df2 <- df %>% select(date, eop) %>% distinct()

# Điều chỉnh theme
p2 <- p1  +
  geom_line(aes(date, eop), col = "dodgerblue4", size = 1) +
  geom_point(aes(date, eop), col = "dodgerblue4", size = 2.5) +
  geom_text(aes(date, eop, label = eop), vjust = 1.2,
            hjust = -0.1) +
  scale_fill_manual(values = c("grey60", "coral2")) +
  theme_minimal() +
  theme(
    axis.line = element_line(color = "gray40", size = 0.5),
    legend.position = "top") +
  scale_x_date(breaks = data$date,
               date_labels = "%b") +
  theme(panel.grid.minor.x = element_blank(),
        legend.title = element_blank()) +
  ggtitle("Overview of active users") +
  xlab("Date") +
  ylab("Number of active users")
p2
```



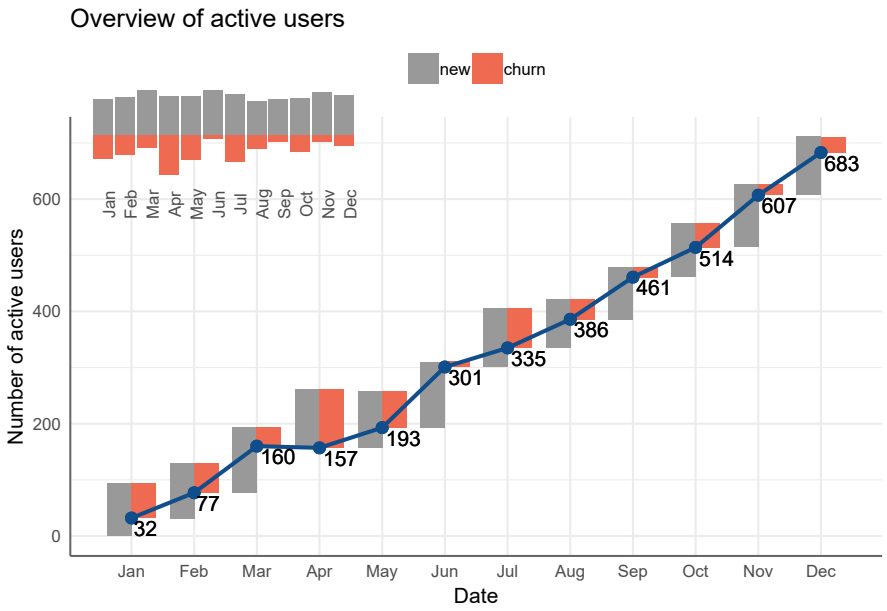
Bước tiếp theo, ta cần xây dựng biểu đồ bar đơn giản để có thể đưa vào góc phần tư bên trái của biểu đồ vừa tạo.

```
p3 <- df %>%
  mutate(value = case_when(
    group == "churn" ~ -1 * value,
    TRUE ~ value
  )) %>%
  ggplot(aes(date, value)) +
  geom_bar(aes(fill = group), stat = "identity") +
  scale_fill_manual(values = c("grey60", "coral2")) +
  theme_minimal() +
  theme(
    legend.position = "none",
    axis.title.x = element_blank(),
    axis.title.y = element_blank(),
    axis.ticks.y = element_blank(),
    axis.text.y = element_blank(),
    panel.grid.minor = element_blank(),
    panel.grid.major = element_blank(),
    axis.text.x = element_text(angle = 90)
  ) +
  scale_x_date(breaks = data$date,
               date_labels = "%b")
p3
```



Cuối cùng, ta có thể nhóm hai biểu đồ trên với `grid` & `gridExtra`.

```
grid.newpage()

# Xác định vị trí cho biểu đồ chính
position_1 <- viewport(width = 1, height = 1,
                       x = 0.5, y = 0.5)

# Vị trí cho biểu đồ phụ
position_2 <- viewport(width = 0.35, height = 0.25,
                       x = 0.25, y = 0.75)

print(p2, vp = position_1)
print(p3, vp = position_2)
```


Overview of active users
new
churn
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Number of active users
0
200
400
600
32
77
160
157
193
301
335
386
461
514
607
683
Date

A

More to Say

Yeah! I have finished my book, but I have more to say about some topics. Let me explain them in this appendix.

To know more about **bookdown**, see https://bookdown.org.

Tài liệu tham khảo

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2018). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.7.18.

Chỉ mục