O pacote magrittr e o operador %>%

Vítor Wilher

Cientista de Dados | Mestre em Economia

Plano de Voo

Introdução

`Pipes` (ou tubos) são uma ferramenta poderosa para expressar claramente uma sequência de várias operações. Até agora, você os utiliza sem saber como eles funcionam ou quais são as alternativas. Agora, nessa subseção, é hora de explorar o pipe com mais detalhes. Você aprenderá as alternativas ao pipe, quando não o usar, e algumas ferramentas relacionadas úteis.[1]

[1] Essa seção está baseada em Grolemund and Wickham [2017].

Introdução

`Pipes`, %>%, são carregados através do pacote *magrittr* como abaixo.

```
library(magrittr)
```

Alternativas ao `pipe`

O objetivo do `pipe` é ajudá-lo a escrever código de uma maneira que seja mais fácil de ler e entender. Para ver por que o `pipe` é tão útil, exploraremos várias maneiras de escrever o mesmo código. Vamos usar o código para contar uma história sobre um coelhinho chamado Foo Foo:

Little bunny Foo Foo
Went hopping through the forest
Scooping up the field mice
And bopping them on the head

Alternativas ao `pipe`

Nós começamos definindo um objeto que representa *little bunny Foo Foo*:

```
foo_foo <- little_bunny()
```

Posteriormente, nós usamos uma função para cada verbo-chave: `hop()`, `scoop()` e `bop()`.

Alternativas ao `pipe`

O meio mais mais simples é salvar cada passo como um novo objeto, como abaixo:

```
foo_foo_1 <- hop(foo_foo, through = forest)
foo_foo_2 <- scoop(foo_foo_1, up = field_mice)
foo_foo_3 <- bop(foo_foo_2, on = head)
```

Alternativas ao `pipe`

A principal desvantagem dessa forma de escrever código é que ela te obriga a nomear cada elemento intermediário. Se houver nomes naturais, é uma boa ideia e você deve fazê-lo. Mas muitas vezes, como este neste exemplo, não existem nomes naturais e você adiciona sufixos numéricos para tornar os nomes exclusivos. Isso leva a dois problemas:

- O código está cheio de nomes sem importância;
- Você precisa incrementar cuidadosamente o sufixo em cada linha.

Alternativas ao `pipe`

Vamos ver agora um exemplo mais interessante. Como adicionar uma coluna em um data frame usando dados do mesmo.

```
library(tidyverse)
diamonds <- ggplot2::diamonds
diamonds2 = mutate(diamonds, price_per_carat = price / carat)
```

Usando o pipe

E agora usando o pipe:

```
diamonds2 <- diamonds %>%
  dplyr::mutate(price_per_carat = price / carat)
```

Usando o pipe

Podemos usar o pipe agora para codar nossa música:

```
foo_foo %>%
  hop(through = forest) %>%
  scoop(up = field_mice) %>%
  bop(on = head)
```

Ao invés de:

```
foo_foo_1 <- hop(foo_foo, through = forest)
foo_foo_2 <- scoop(foo_foo_1, up = field_mice)
foo_foo_3 <- bop(foo_foo_2, on = head)
```

Usando o `pipe`

Observe que nos concentramos nos verbos, não nos substantivos. Você pode ler essa série de composições de funções como se fosse um conjunto de ações imperativas. Foo Foo pula, depois escava e depois bate. A desvantagem, é claro, é que você precisa estar familiarizado com o `pipe`. Se você nunca viu %>% antes, não tem idéia do que esse código faz. Felizmente, a maioria das pessoas adota a ideia muito rapidamente, portanto, quando você compartilha seu código com outras pessoas que não estão familiarizadas com o `pipe`, você pode ensiná-las facilmente.

Usando o `pipe`

O `pipe` funciona executando uma "transformação lexical": nos bastidores, o `magrittr` remonta o código no `pipe` a um formato que funciona substituindo um objeto intermediário. Quando você executa um pipe como o acima, o `magrittr` faz algo assim:

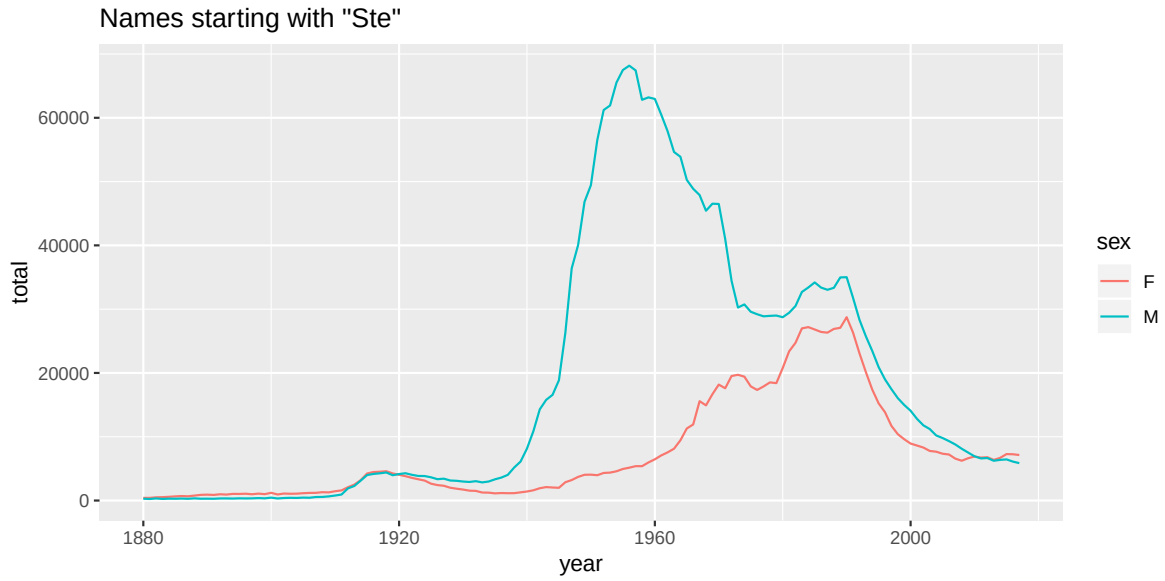
```
my_pipe <- function(.) {
  . <- hop(., through = forest)
  . <- scoop(., up = field_mice)
  bop(., on = head)
}

my_pipe(foo_foo)
```

Usando o pipe

```
library(babynames) # data package
library(dplyr)     # provides data manipulating functions.
library(magrittr)  # ceci n'est pas un pipe
library(ggplot2)   # for graphics

babynames %>%
    filter(name %>% substr(1, 3) %>% equals("Ste")) %>%
    group_by(year, sex) %>%
    summarize(total = sum(n)) %>%
    qplot(year, total, color = sex, data = ., geom = "line") %>%
    add(ggtitle('Names starting with "Ste"')) %>%
    print
```



Usando o `pipe`

A sequência de códigos contida no `pipe` pode ser descrita como uma espécie de receita:

1. pegue os dados do bebê e depois
2. filtre-o de modo que o nome da sub-string do caractere 1 a 3 seja igual a “Ste” e, em seguida,
3. agrupe-o por ano e sexo, depois
4. resuma-o calculando a soma total de cada grupo e, em seguida,
5. traçar os resultados, colorindo por sexo, depois
6. adicione um título e, em seguida,
7. imprima na tela.

Usando o `pipe`

O exemplo ilustra alguns recursos do %>%. Em primeiro lugar, as funções dplyr `filter`, `group_by` e `summarise`, todos tomam como primeiro argumento um objeto de dados e, por padrão, é onde o %>% o colocará no seu lado esquerdo. Os dados de nomes de bebês são inseridos como primeiro argumento na chamada para `filter`. Quando a filtragem é concluída, o resultado é passado como o primeiro argumento para `group_by` e da mesma forma para `summarise`. No entanto, nem sempre é uma sorte que uma função seja projetada para aceitar os dados (ou o que quer que você esteja transmitindo) como seu primeiro argumento (as funções dplyr são projetadas com %>% de operações em mente). É o caso de, por exemplo, qplot, mas observe os dados =. argumento. Isso indica a %>% para colocar o lado esquerdo lá, e não como o primeiro argumento. Essa é uma maneira simples e natural de acomodar a falta de consistência das assinaturas de funções e permite que o lado esquerdo vá para qualquer lugar da chamada no lado direito.

Usando o `pipe`

Você também pode ter notado que `print` é usada sem parênteses; isso é para tornar o código ainda mais limpo quando apenas um lado esquerdo for necessário como entrada. Por fim, observe que %>% pode ser usado de forma aninhada (uma cadeia separada é encontrada na chamada de `filter`) e que o `magrittr` possui aliases para operadores comumente usados, como adicionar para + e igual a == usado acima. Isso torna as cadeias de `pipes` mais legíveis (não necessariamente menores).

Quando não usar `pipes`

O `pipe` é uma ferramenta poderosa, mas não é a única ferramenta à sua disposição, e não resolve todos os problemas! Os `pipes` são mais úteis para reescrever uma sequência linear bastante curta de operações. Eu acho que você deve procurar outra ferramenta quando:

- Seus `pipes` são maiores que (digamos) dez etapas. Nesse caso, crie objetos intermediários com nomes significativos. Isso facilitará a depuração, porque você poderá verificar com mais facilidade os resultados intermediários e facilitará a compreensão do seu código, porque os nomes das variáveis podem ajudar a comunicar a intenção;
- Você tem várias entradas ou saídas. Se não houver um objeto principal sendo transformado, mas dois ou mais objetos combinados, não use o pipe.

Quando não usar `pipes`

- Você está começando a pensar em um gráfico direcionado com uma estrutura de dependência complexa. Os `pipes` são fundamentalmente lineares e a expressão de relacionamentos complexos com eles normalmente gera código confuso.

G. Grolemund and H. Wickham. *R for Data Science*. O'Reilly Media, 2017.