Introdução ao Pacote dplyr

R Básico + dplyr

Me Elisangela, Douglas Vinícius, Jossivana Macedo
Universidade Federal de Rondônia
22 de Outubro de 2019

Partes

1. Introdução sobre R e RStudio.
    - Universo `tidyverse`.
2. Tibbles x Data Frames.
    - O que é Dados Organizados?
    - Operador *pipe* `%>%`.
    - Breve introdução ao Tidyr.
        - gather(); spread(); separate(); unite().
3. Manipulando Data Frames com `dplyr`.
    - select(); rename(); mutate().

R & RStudio

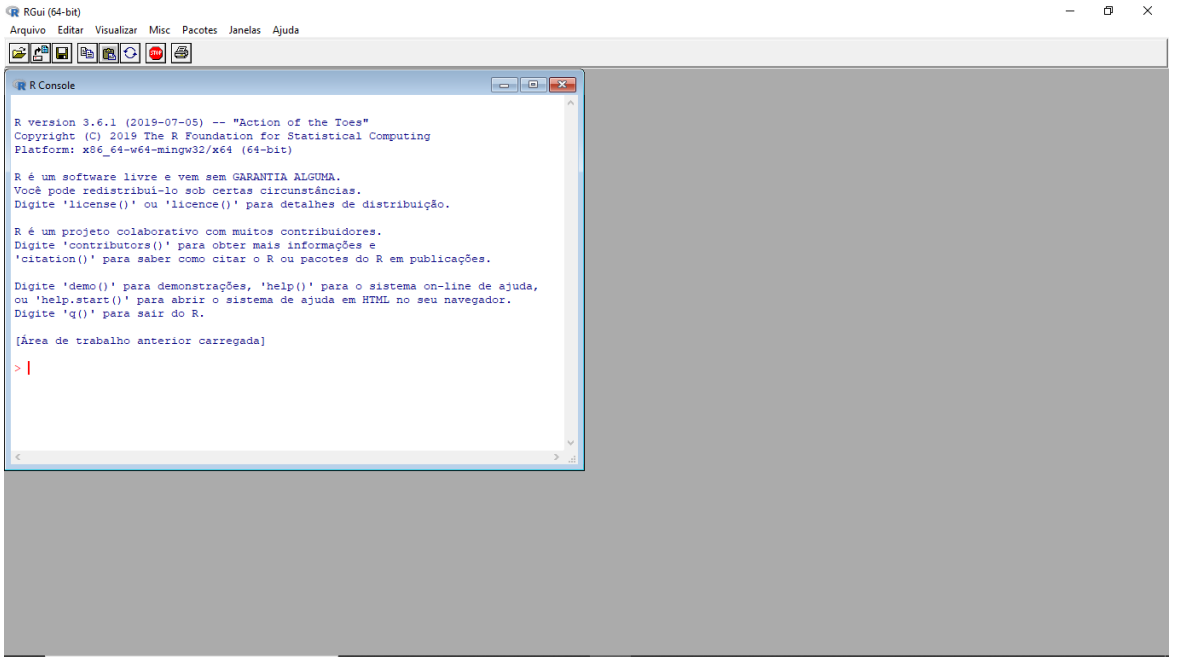
O ambiente R

R é uma linguagem de programação estatística que vem passando por diversas evoluções e se tornando cada vez mais uma linguagem de amplos objetivos. Excelente para manipulação de dados, cálculo e exibição gráfica. Segue alguns motivos para aprender o R:

- É completamente gratuito e de livre distribuição;
- Curva de aprendizado bastante amigável, sendo muito fácil de se aprender;
- Enorme quantidade de tutoriais e ajuda disponíveis gratuitamente na internet;
- É excelente para criar rotinas e sistematizar tarefas repetitivas;
- Amplamente utilizado pela comunidade acadêmica e pelo mercado;
- Quantidade enorme de pacotes, para diversos tipos de necessidades;
- Ótima ferramenta para criar relatórios e gráficos.

R

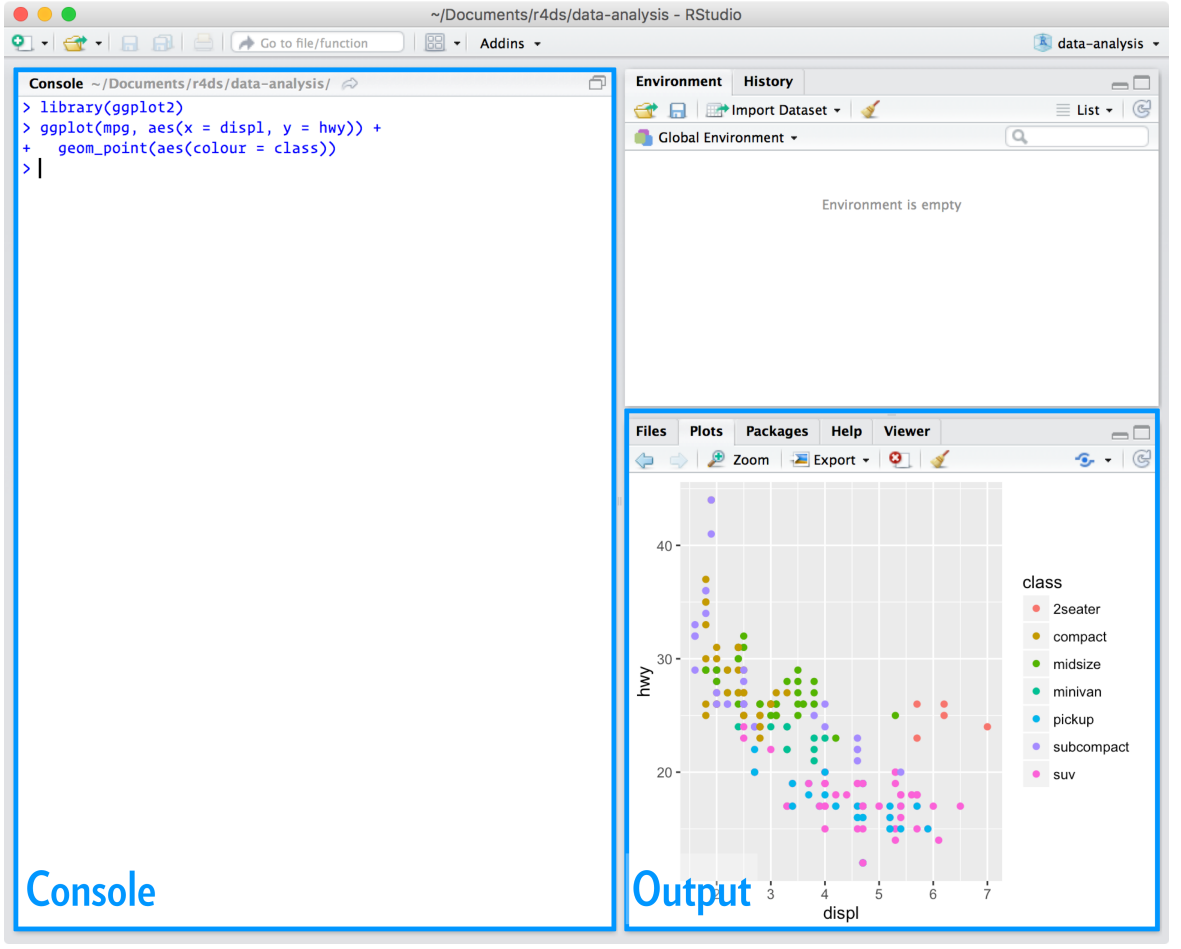
Após instalado, o R tem uma interface assim, com apenas o console para digitar comandos.



Experimente um comando: 2+2, cujo output é 4.

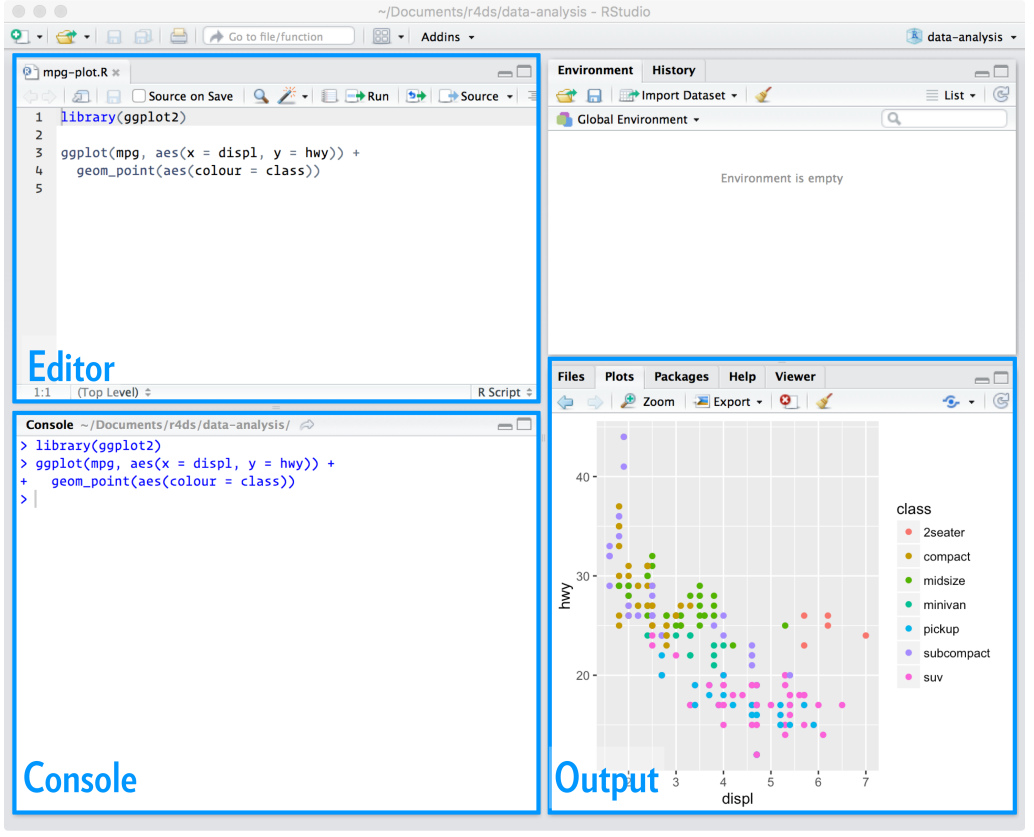
RStudio

E a interface do RStudio é dividida, inicialmente, em 3 partes.



RStudio

A forma mais eficiente e prática de usar o R ou o RStudio é através de um script. No RStudio, vá em *File* → *New File* → *R Script*. A interface agora fica dividida em 4 partes.



Classes de Objetos no R

R possui 5 classes básicas de objetos, também chamados de objetos "atômicas":

- character;
- numeric (real numbers);
- integer;
- complex;
- logical (True/False).

O tipo mais básico de objeto R é um vetor. Um vetor só pode conter elementos de uma mesma classe. Mas há uma exceção, que é uma lista. Uma lista é representada como um vetor, mas pode conter objetos de diferentes classes.

Atributos

Os objetos R podem ter atributos, como metadados para o objeto. Esses metadados podem ser muito úteis, pois ajudam a descrever o objeto. Por exemplo, nomes de colunas em um quadro de dados ajudam a nos dizer quais dados estão contidos em cada uma das colunas. Alguns exemplos de atributos de objeto R são:

- nomes, dimnames;
- dimensões (por exemplo, matrizes, matrizes);
- classe (por exemplo, inteiro, numérico);
- comprimento;
- outros atributos/metadados definidos pelo usuário.

Os atributos de um objeto (se houver) podem ser acessados usando a função `attributes()`. Nem todos os objetos R contêm atributos; nesse caso, a `attributes()` retorna NULL.

Pacote `swirl`

O `swirl` é um pacote do R construído para transformar o console em uma ferramenta interativa para aprender R. `swirl` ensina programação de R e ciência de dados interativamente, no seu próprio ritmo e diretamente no console do R. Para entender melhor o projeto, veja http://swirlstats.com/ e http://swirlstats.com/students. Nestes endereços são dados os detalhes sobre como usar o `swirl`. Uma vez intalado e carregado o pacote, você é levado a efetuar tarefas.

O `swirl` dá acesso às tarefas de cursos de R que estão disponíveis também no Coursera, como o R Programming: The basics of programming in R, em https://pt.coursera.org/learn/r-programming. Além deste, estão disponíveis no `swirl`: Regression Models: The basics of regression modeling in R, Statistical Inference: The basics of statistical inference in R, e Exploratory Data Analysis: The basics of exploring data in R.

Tibbles x Data frames

Mas afinal, como dois quadros de dados aparecem totalmente diferentes no console?

O que é um tibble? Tibbles são similares aos data frames, porém diferentes em dois aspectos: **impressão** e **indexação**.

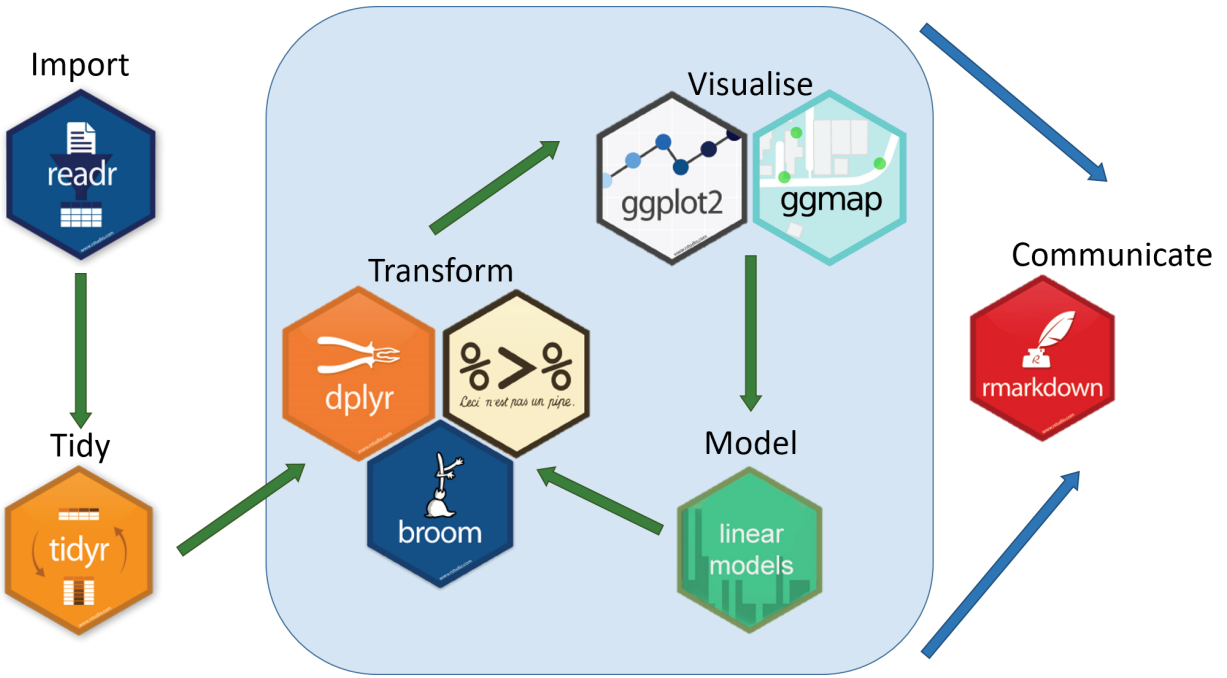
Sempre que indexarmos um tibble usando `[`, o resultado será outro tibble. Usando `[[` o resultados será um vetor.

Na impressão no console, os *tibbles* apresentam apenas as dez primeiras linhas e todas as colunas que cabem na tela, tornando mais fácil o trabalho com grandes volumes de dados.

tidyverse

Universo *tidyverse*

> O tidyverse é uma coleção de pacotes R projetados para ciência de dados. Todos os pacotes compartilham uma filosofia de design, gramática e estruturas de dados subjacentes.



%>%


%>%
Ceci n'est pas un pipe.
www.tidyverse.org

Operador *pipe* %>%

O pacote `magrittr` tem dois objetivos: diminuir o tempo de desenvolvimento e melhorar a legibilidade e a manutenção do código. Para começar a utilizar o *pipe*, instale e carregue o pacote `magrittr`.

```
install.packages("magrittr")
library(magrittr)
```

O operador do **pipeline** `%>%` é muito útil para reunir várias funções em uma sequência de operações.

Tubulação básica:

- x `%>%` *f* é equivalente a *f(x)*;
- x `%>%` *f(y)* é equivalente a *f(x, y)*;
- x `%>%` *f* `%>%` *g* `%>%` *h* é equivalente a *h(g(f(x)))*.

%>%

Observe abaixo que toda vez que desejamos aplicar mais de uma função, a sequência é ocultada em uma sequência de chamadas de funções aninhadas difíceis de ler, ou seja:

> third(second(first(x)))

Esse aninhamento não é uma maneira natural de pensar em uma sequência de operações. O `%>%` permite que você encadeie operações da esquerda para a direita, ou seja:

> first(x) %>% second() %>% third()

Por exemplo:

%>%

```
x <- c(0.109, 0.359, 0.63, 0.996, 0.515, 0.142, 0.017, 0.829, 0.907)

round(exp(diff(log(x))), 1)
```

```
## [1]  3.3  1.8  1.6  0.5  0.3  0.1 48.8  1.1
```

Com ajuda do *pipe*:

```
x %>%
  log() %>%
  diff() %>%
  exp() %>%
  round(1)
```

```
## [1]  3.3  1.8  1.6  0.5  0.3  0.1 48.8  1.1
```

%>%

Em resumo, aqui estão quatro razões pelas quais você deve usar tubos ou pipe no R:

- Estruturara a sequência de suas operações de dados da esquerda para a direita, ao contrário de dentro para fora;
- evita chamadas de função aninhadas;
- minimiza a necessidade de variáveis locais e definições de funções;
- facilita a adição de etapas em qualquer lugar da sequência de operações.

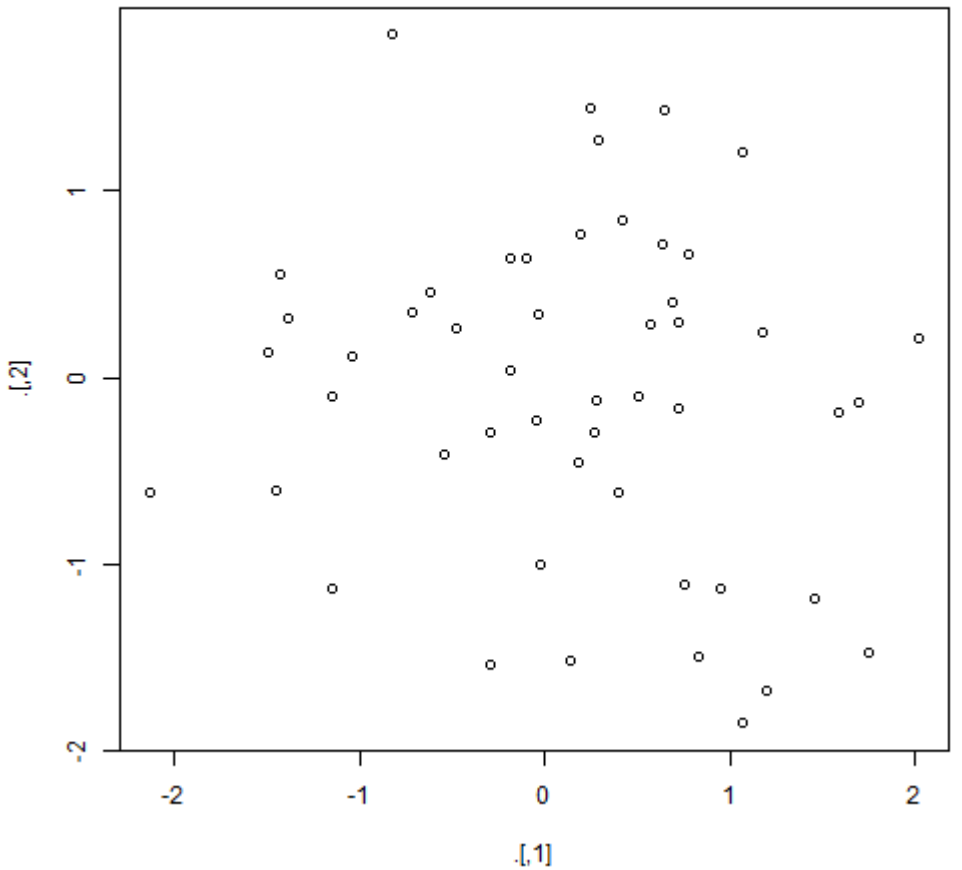
%T>%

Mesmo sendo `%>%` o operador de tubulação (principal) do pacote magrittr, existem alguns outros operadores que fazem parte do mesmo pacote:

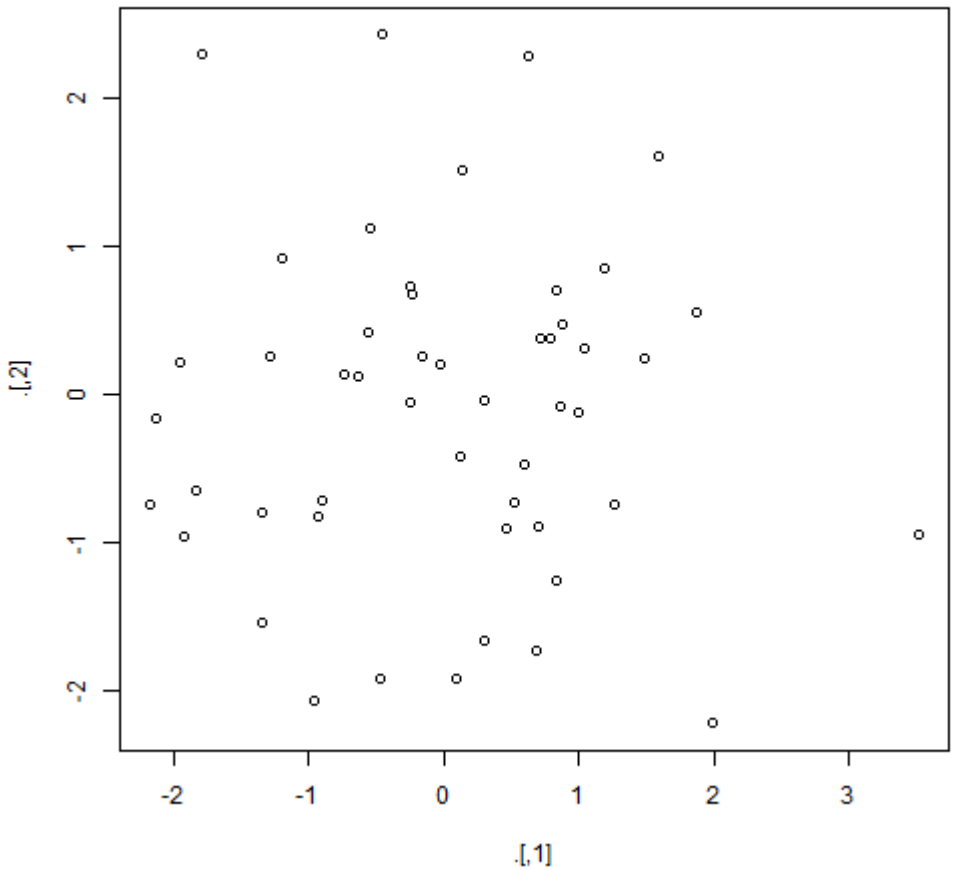
Ao trabalhar com tubos mais complexos, às vezes é útil chamar uma função por seus efeitos colaterais. Talvez você queira imprimir o objeto atual, plotá-lo ou salvá-lo em disco. Muitas vezes, essas funções não retornam nada, efetivamente encerrando o pipe.

Para contornar esse problema, usar-se o tubo *"tee"* `%T>%` funciona como `%>%` exceto que retorna o lado esquerdo em vez do lado direito. É chamado de *"tee"* porque é como um tubo em forma de T literal.

```
rnorm(100) %>%
  matrix(ncol = 2) %>%
  plot() %>%
  str()
```



```
rnorm(100) %>%
  matrix(ncol = 2) %T>%
  plot() %>%
  str()
```



Dados organizados

Costuma-se dizer que 80% da análise de dados é gasta no processo de limpeza e preparação dos dados (Dasu e Johnson 2003).

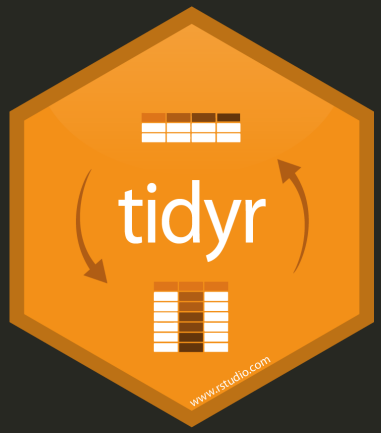
Por que garantir que seus dados estejam organizados? Existem duas vantagens principais:

- Há uma vantagem geral em escolher uma maneira consistente de armazenar dados. Se você possui uma estrutura de dados consistente, é mais fácil aprender as ferramentas que funcionam com ela porque elas têm uma uniformidade subjacente.

- Há uma vantagem específica em colocar variáveis em colunas porque permite que a natureza vetorizada de R seja eficiente. A maioria das funções R trabalha com vetores de valores. Isso faz com que a transformação de dados organizados pareça particularmente natural.

O `dplyr`, o `ggplot2` e todos os outros pacotes no tidyverse foram projetados para trabalhar com dados organizados.

tidyr

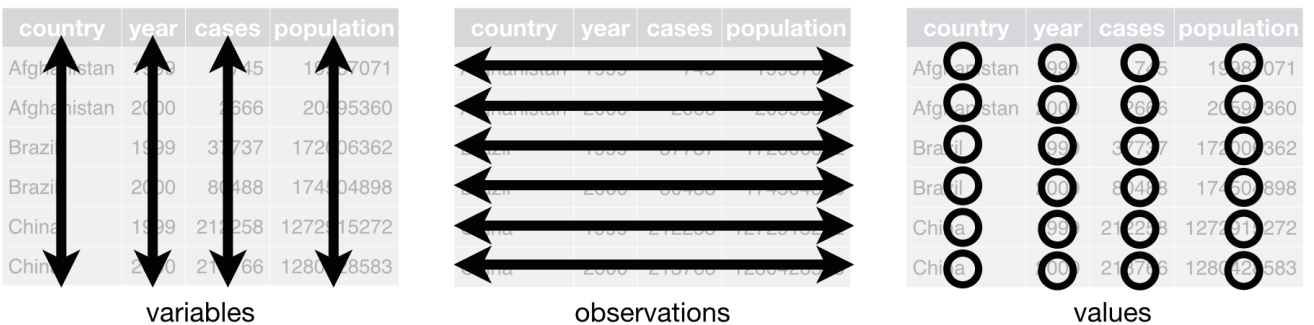


tidyr

O objetivo do tidyr é ajudá-lo a criar dados organizados.

- 1.Cada variável no conjunto de dados é colocada em sua própria coluna;
- 2.Cada observação é colocada em sua própria linha;
- 3.Cada valor é colocado em sua própria célula.

Os dados que satisfazem essas regras são conhecidos como dados organizados



tidyr

Vamos instalar o pacote `EDAWR` e carregá-lo:

```
devtools::install_github("rstudio/EDAWR")
library(EDAWR)
```

Quais desses quadros de dados são mais similares a dados organizados?

```
EDAWR::storms
```

```
##     storm wind pressure       date
## 1 Alberto  110     1007 2000-08-03
## 2    Alex   45     1009 1998-07-27
## 3 Allison   65     1005 1995-06-03
## 4     Ana   40     1013 1997-06-30
## 5  Arlene   50     1010 1999-06-11
## 6  Arthur   45     1010 1996-06-17
```

```
EDAWR::cases
```

```
##   country  2011  2012  2013
## 1      FR  7000  6900  7000
## 2      DE  5800  6000  6200
## 3      US 15000 14000 13000
```

```
EDAWR::pollution
```

```
##       city  size amount
## 1 New York large     23
## 2 New York small     14
## 3   London large     22
## 4   London small     16
## 5  Beijing large    121
## 6  Beijing small     56
```

tidyr

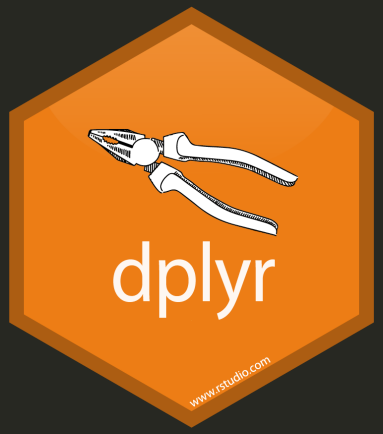
O `tidyr` possui duas funções principais:

`gather`: Transforma um `tibble` *wide* em *long*, ou seja, transforma os dados no formato *tidy*.

`spread`: Transforma um `tibble` *long* em *wide*, ou seja, transforma dados que estão no formato *tidy* em formato não *tidy*.

Além disso, existem duas funções que podem ser importantes na nossa análise: `separate` e `unite`, que separa uma coluna em duas e vice versa.

dplyr



O Pacote `dplyr`

O pacote `dplyr` foi desenvolvido por Hadley Wickham, cientista chefe do RStudio. É uma versão otimizada do pacote `plyr`. O pacote `dplyr` não fornece nenhuma funcionalidade "nova" ao R, pois já é feito com base no R, mas simplifica **bastante** a funcionalidade no R.

Uma contribuição importante do `dplyr` é que ele fornece uma "gramática" (em particular, verbos) para manipulação

Gramática do `dplyr`

Alguns dos principais "verbos" básicos de tabela única fornecidos pelo `dplyr` são:

- `select`: retorna um subconjunto das colunas de um data.frames, usando uma notação flexível;
- `pull()`: retire uma única variável;
- `filter`: extrair um subconjunto de linhas(observações) de um data.frames com base em condições lógicas;
- `arrange`: reordenar linhas de um data.frames;
- `rename`: renomear variáveis em um data.frames;
- `mutate`: adiciona novas variáveis/colunas ou transforme variáveis existentes;
- `summarise/summarize`: gera estatísticas resumidas de diferentes variáveis no data.frames, possivelmente dentro dos estratos.

Propriedades das funções do `dplyr`

As funções têm algumas características comuns:

- 1.O primeiro argumento é um data.frames;
- 2.Os argumentos subsequentes descrevem o que fazer com o data.frames especificado no primeiro argumento;
- 3.O resultado de retorno de uma função é um novo data.frames;
- 4.Os data.frames devem devidamente formatados e anotados para que tudo isso seja útil. Em particular, os dados devem estar *organizados*.

Instalando o Pacote `dplyr`

O pacote pode ser instalado a partir do CRAN ou do GitHub usando o pacote `devtools` com a função `install_github()`. O repositório GitHub normalmente contém as versões mais atualizadas dos pacotes.

Para instalar a partir do CRAN, bastar executar:

Para instalar a partir do GitHub, execute:

Após a instalação do pacote, carregá-lo com a função `library()`:

Ao carregar o pacote você pode receber alguns avisos, porque há funções no `dplyr`que têm o mesmo nome que as funções em outros pacotes. Por enquanto pode ignorar os avisos.

Para melhor apresentar as funcionalidades da função, usaremos um conjunto de dados diários sobre poluição do ar e taxa de mortalidade da cidade de Chicago, nos EUA. Este banco de dados encontra no seguinte endereço: http://www.biostat.jhsph.edu/~rpeng/leanpub/rprog/chicago_data.zip e está em um arquivo zipado.

Você pode carregar os dados no R usando a função `readRDS()`:

```
chicago <- readRDS("data/chicago.rds")
```

Descrição do banco: tem 8 colunas e 6940 linhas. Cada linha refere-se a um dia. As colunas são:

- city: cidade, neste campo tem apenas "chic" referenciando a cidade de Chicago.
- tmpd: temperatura em Fahrenheit.
- dptp: temperatura do ponto de orvalho.
- date: tempo em dias.
- pm25tmean2: partículas médias < 2,5mg por m cúbico (mais perigoso).
- pm10tmean2: partículas médias em 2,5^-10 por m cúbico.
- o3tmean2: Ozônio em partes por bilhão.
- no2tmean2: Medição mediana de dióxido de sulfato.

Umas das formas de ter informações do seu banco de dados é utilizar as seguintes funções `dim()` e `str()`. A primeira especifica a dimensão do seu banco e a segunda a estrutura do seu banco de dados.

select()

Muitas vezes teremos um data.frames contendo um grande número de dados. Com isso, a função `select()` permite obter as poucas colunas que você pode precisar.

Suponhamos que desejássemos pegar as 3 primeiras colunas. Há algumas maneiras de fazer isto. Poderíamos, por exemplo, usar o índices númericos. Mas também podemos usar os nomes diretamente.

```
names(chicago[1:3])
```

```
## [1] "city" "tmpd" "dptp"
```

```
subset1 <- select(chicago, city:dptp)
head(subset1)
```

```
##   city tmpd   dptp
## 1 chic 31.5 31.500
## 2 chic 33.0 29.875
## 3 chic 33.0 27.375
## 4 chic 29.0 28.625
## 5 chic 32.0 28.875
## 6 chic 40.0 35.125
```

select()

Normalmente : não pode ser usado com nomes ou sequências de caracteres, mas dentro da função `select()` pode usá-lo para especificar um intervalo de nomes de variáveis.

Pode **omitir** variáveis usando a função `select()` usando o sinal negativo.

```
subset2 <- select(chicago, -(city:dptp))
head(subset2)
```

```
##         date pm25tmean2 pm10tmean2 o3tmean2 no2tmean2
## 1 1987-01-01         NA   34.00000 4.250000  19.98810
## 2 1987-01-02         NA         NA 3.304348  23.19099
## 3 1987-01-03         NA   34.16667 3.333333  23.81548
## 4 1987-01-04         NA   47.00000 4.375000  30.43452
## 5 1987-01-05         NA         NA 4.750000  30.33333
## 6 1987-01-06         NA   48.00000 5.833333  25.77233
```

select()

o que indica que estamos incluindo todas as variáveis, exceto as variáveis `city` até `dptp`.

O código equivalente ao anterior sem o uso do pacote seria:

```
i <- match("city", names(chicago))
j <- match("dptp", names(chicago))
head(chicago[, -(i:j)])
```

```
##         date pm25tmean2 pm10tmean2 o3tmean2 no2tmean2
## 1 1987-01-01         NA   34.00000 4.250000  19.98810
## 2 1987-01-02         NA         NA 3.304348  23.19099
## 3 1987-01-03         NA   34.16667 3.333333  23.81548
## 4 1987-01-04         NA   47.00000 4.375000  30.43452
## 5 1987-01-05         NA         NA 4.750000  30.33333
## 6 1987-01-06         NA   48.00000 5.833333  25.77233
```

A função de correspondência `mathc()` retorna um vetor das posições das (primeiras) correspondências de seu primeiro argumento no segundo. De acordo com a Documentação R, a função é equivalente ao operador `%in%` que indica se uma correspondência foi localizada para o vetor1 no vetor2. O valor do resultado será VERDADEIRO ou FALSO, mas nunca NA. Portanto, o operador `%in%` pode ser útil em condições `if`. Por exemplos:

```
#função math().
v1 <- c("a1","b2","c1","d2")
v2 <- c("g1","x2","d2","e2","f1","a1","c2","b2","a2")
x <- match(v1,v2)
x
```

```
## [1]  6  8 NA  3
```

```
#com o operador %in%.
v1 <- c("a1","b2","c1","d2")
v2 <- c("g1","x2","d2","e2","f1","a1","c2","b2","a2")
v1 %in% v2
```

```
## [1]  TRUE  TRUE FALSE  TRUE
```

starts_with()

A função `select()` permite uma sintaxe especial que especifica nomes de variáveis com base em padrões. Por exemplo, há várias funções auxiliares que você pode usar:

`starts_with("abc")`: corresponde aos nomes que começam com "abc";

```
#Queremos manter todas as variáveis que começam com um "d":
subset3 <- select(chicago, starts_with("d"))
head(subset3)
```

```
##     dptp       date
## 1 31.500 1987-01-01
## 2 29.875 1987-01-02
## 3 27.375 1987-01-03
## 4 28.625 1987-01-04
## 5 28.875 1987-01-05
## 6 35.125 1987-01-06
```

ends_with()

`ends_with("xyz")`: corresponde aos nomes que terminam com "xyz";

```
subset4 <- select(chicago, ends_with("2"))
head(subset4)
```

```
##   pm25tmean2 pm10tmean2 o3tmean2 no2tmean2
## 1         NA   34.00000 4.250000  19.98810
## 2         NA         NA 3.304348  23.19099
## 3         NA   34.16667 3.333333  23.81548
## 4         NA   47.00000 4.375000  30.43452
## 5         NA         NA 4.750000  30.33333
## 6         NA   48.00000 5.833333  25.77233
```

contains()

`contains("ijk")`: corresponde aos nomes que contêm "ijk";

```
subset5 <- select(chicago, contains("tmean"))
head(subset5)
```

```
##   pm25tmean2 pm10tmean2 o3tmean2 no2tmean2
## 1         NA   34.00000 4.250000  19.98810
## 2         NA         NA 3.304348  23.19099
## 3         NA   34.16667 3.333333  23.81548
## 4         NA   47.00000 4.375000  30.43452
## 5         NA         NA 4.750000  30.33333
## 6         NA   48.00000 5.833333  25.77233
```

matches("(.)\\1")

`matches("(.)\\1")`: selecionar variáveis que correspondem a uma expressão regular. Esta corresponde a qualquer variável que contenha caracteres repetidos. Você aprenderá mais sobre expressões regulares no capítulo Strings do livro R for data science.

```
##   tmpd pm25tmean2 pm10tmean2 o3tmean2 no2tmean2
## 1 31.5         NA   34.00000 4.250000  19.98810
## 2 33.0         NA         NA 3.304348  23.19099
## 3 33.0         NA   34.16667 3.333333  23.81548
## 4 29.0         NA   47.00000 4.375000  30.43452
## 5 32.0         NA         NA 4.750000  30.33333
## 6 40.0         NA   48.00000 5.833333  25.77233
```

num_range("x", 1:3)

`num_range("x", 1:3)`: Corresponde x1, x2 e x3.

```
#Criando um objeto df que é um data frame
df <- as.data.frame(matrix(runif(100), nrow = 10))
df <- tbl_df(df[c(3, 4, 7, 1, 9, 8, 5, 2, 6, 10)])
select(df, V4:V6)
```

```
## # A tibble: 10 x 8
##        V4     V7     V1     V9     V8     V5      V2     V6
##     <dbl>  <dbl>  <dbl>  <dbl> <dbl>  <dbl>   <dbl> <dbl>
##  1 0.979  0.673  0.171  0.735  0.419 0.109  0.850   0.141
##  2 0.405  0.258  0.717  0.360  0.952 0.403  0.227   0.613
##  3 0.208  0.959  0.417  0.154  0.531 0.336  0.792   0.602
##  4 0.0433 0.442  0.0702 0.278  0.834 0.266  0.492   0.948
##  5 0.315  0.733  0.590  0.142  0.850 0.838  0.587   0.247
##  6 0.800  0.393  0.500  0.925  0.824 0.660  0.00136 0.234
##  7 0.378  0.725  0.0938 0.890  0.857 0.0142 0.451   0.759
##  8 0.0704 0.607  0.909  0.961  0.127 0.823  0.0141  0.577
##  9 0.869  0.0987 0.110  0.516  0.446 0.592  0.101   0.349
## 10 0.249  0.243  0.289  0.0874 0.529 0.643  0.375   0.350
```

```
select(df, num_range("V", 4:6))
```

```
## # A tibble: 10 x 3
##        V4     V5    V6
##     <dbl>  <dbl> <dbl>
##  1 0.979  0.109  0.141
```

Você também pode usar expressões regulares mais gerais, se necessário. Veja a página de ajuda (?select) para mais detalhes.

`select()` pode ser usado para renomear variáveis, mas raramente é útil porque descarta todas as variáveis não mencionadas explicitamente. Em vez disso, use `rename()`, que é uma variante de `select()` que mantém todas as variáveis que não são mencionadas explicitamente.

Outra opção é usar `select()` em conjunto com o `everything()` auxiliar. Isso é útil se você tiver um punhado de variáveis que deseja mover para o início do quadro de dados.

```
subset7 <- select(chicago, o3tmean2, no2tmean2, everything())
head(subset7)
```

```
##   o3tmean2 no2tmean2 city tmpd   dptp       date pm25tmean2 pm10tmean2
## 1 4.250000  19.98810 chic 31.5 31.500 1987-01-01         NA   34.00000
## 2 3.304348  23.19099 chic 33.0 29.875 1987-01-02         NA         NA
## 3 3.333333  23.81548 chic 33.0 27.375 1987-01-03         NA   34.16667
## 4 4.375000  30.43452 chic 29.0 28.625 1987-01-04         NA   47.00000
## 5 4.750000  30.33333 chic 32.0 28.875 1987-01-05         NA         NA
## 6 5.833333  25.77233 chic 40.0 35.125 1987-01-06         NA   48.00000
```

rename()

Para renomear variáveis em uma data.frames em R não é tão prático. E a função `rename()` foi projetada para facilitar esse processo.

Os nomes das cinco primeiras variáveis do data frame chicago.

```
#Imprimir às 3 primeiras linhas da primeira a quinta coluna.
head(chicago[, 1:5], 3)
```

```
##   city tmpd   dptp       date pm25tmean2
## 1 chic 31.5 31.500 1987-01-01         NA
## 2 chic 33.0 29.875 1987-01-02         NA
## 3 chic 33.0 27.375 1987-01-03         NA
```

rename()

A coluna `dptp` deve representar a temperatura do ponto de orvalho e a coluna `pm25tmean2` fornece os dados do PM2.5. No entanto, esses nomes são bastante obscuros ou estranhos e provavelmente serão renomeados para algo mais sensato.

```
chicago <- rename(chicago, Temp_Orv = dptp, pm25 = pm25tmean2)
head(chicago[, 1:5], 3)
```

```
##   city tmpd Temp_Orv       date pm25
## 1 chic 31.5   31.500 1987-01-01   NA
## 2 chic 33.0   29.875 1987-01-02   NA
## 3 chic 33.0   27.375 1987-01-03   NA
```

A sintaxe dentro da `rename()` função é ter o novo nome no lado esquerdo do `=` sinal e o nome antigo no lado direito.

mutate()

Em certas situações é útil adicionar novas colunas/variáveis que são funções de colunas existentes no data frames, ou seja, criar novas variáveis derivadas de variáveis existentes. Esse é o trabalho de `mutate()`. Esta função adiciona novas colunas no final do seu conjunto de dados. `mutate()` fornece uma interface limpa para fazer isso. Lembre-se de que, quando você está no RStudio, a maneira mais fácil de ver todas as colunas é `View()`.

Por exemplo, com os dados de poluição do ar, subtraindo a média dos dados. Dessa forma, podemos verificar se o nível de poluição do ar de um determinado dia é maior ou menor que a média (em oposição a observar seu nível absoluto).

mutate()

Aqui, criamos uma variável `pm25difmean` que subtrai a média da variável `pm25`.

```
chicago <- mutate(chicago, pm25difmean = pm25 - mean(pm25, na.rm = TRUE))
head(chicago)
```

```
##   city tmpd Temp_Orv       date pm25 pm10tmean2 o3tmean2 no2tmean2
## 1 chic 31.5   31.500 1987-01-01   NA   34.00000 4.250000  19.98810
## 2 chic 33.0   29.875 1987-01-02   NA         NA 3.304348  23.19099
## 3 chic 33.0   27.375 1987-01-03   NA   34.16667 3.333333  23.81548
## 4 chic 29.0   28.625 1987-01-04   NA   47.00000 4.375000  30.43452
## 5 chic 32.0   28.875 1987-01-05   NA         NA 4.750000  30.33333
## 6 chic 40.0   35.125 1987-01-06   NA   48.00000 5.833333  25.77233
##   pm25difmean
## 1          NA
## 2          NA
## 3          NA
## 4          NA
## 5          NA
## 6          NA
```

mutate()

Há também a função relacionada `transmute()`, que faz a mesma coisa que `mutate()`, mas elimina todas as variáveis não transformadas.

Aqui, desprezamos as variáveis PM10 e ozônio (O3).

```
head(transmute(chicago,
               pm10difmean = pm10tmean2 - mean(pm10tmean2, na.rm = TRUE),
               O3difmean = o3tmean2 - mean(o3tmean2, na.rm = TRUE)))
```

```
##   pm10difmean O3difmean
## 1   0.1047939 -15.18551
## 2          NA -16.13117
## 3   0.2714605 -16.10218
## 4  13.1047939 -15.06051
## 5          NA -14.68551
## 6  14.1047939 -13.60218
```

Observe que existem apenas duas colunas no quadro de dados transformados.

Há inúmeras funções que pode ser feita, a propriedade é que a função deva ser vetorizada: ela deve pegar um vetor de valores como entrada, retornar um vetor com o mesmo número de valores que a saída. Não há como listar todas as funções possíveis que você possa usar, mas aqui está uma seleção de funções frequentemente úteis:

Obrigado!

Continua...

@Doug7Vinicius