Apostila: introdução ao pacote dplyr

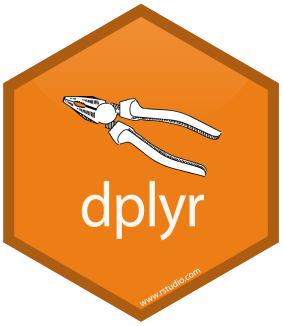
Me. Elisângela C. BiazattiDouglas Vinícius Jossivana Macedo

22 de outubro de 2019

Contents

Chapter 1

Prefácio



Este material foi elaborado com o proprósito de um minicurso, que tem como objetivo apresentar algumas ideias das funções básicas do pacote `dplyr`, podemos usar um computador e um pouco de criatividade para explorar essas idéias em uma variedade de situações. Usamos R com o RStudio para fazer todo o nosso trabalho.

O livro R for data science é o mais indicado para aprender sobre o universo `tidyverse`. Nesse minicurso abordamos mais sobre a gramática das funções básicas do `dplyr` alguns exemplos e exercícios abordados.

1.1 Público-alvo

- Estudantes de estatística que desejam ganhar tempo nos trabalhos da faculdade;
- Acadêmicos com interesse em tornar suas análises e códigos mais legíveis em R.

1.2 Conteúdo:

```
- Primeiro dia (22/10): Breve introdução ao R, organização de dados, exercícios;
- Segundo dia (23/10): select(), rename(), arrange(), mutate(), summarise(), exercícios
- Terceiro dia (24/10): agrupar dados, combinar conjuntos de dados.
```

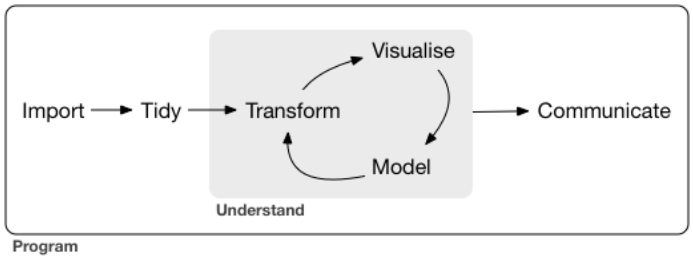
1.3 Pré-requisitos

Chapter 2

Introdução

> "Existem apenas dois tipos de idiomas: os que as pessoas reclamam e os que ninguém usa". - Bjarne Stroustrup

O modelo típico de análise de dados é similar:



Primeiramente, você deve **importar** seus dados para o R. Significa que você pega os dados armazenados em um arquivo, banco de dados ou API da Web e carrega-os em um data frames no R.

Logo após, a ideia é **organizá**-los. Significa armazená-los de forma consistente.

Depois de arrumar os dados, o próximo passo é **transformá**-los. Significa restringir observações de interesse, criar novas variáveis

Depois de organizar os dados com as variáveis necessárias, existem dois mecanismos principais de geração de conhecimento: visualização e modelagem. Eles têm pontos fortes e fracos complementares, portanto qualquer análise real se repetirá entre eles várias vezes.

A **visualização** é uma atividade fundamentalmente humana. Uma boa visualização mostrará coisas que você não esperava, ou fará novas perguntas sobre os dados.

Modelos são ferramentas complementares para visualização. Depois de fazer suas perguntas suficientemente precisas, você pode usar um modelo para respondê-las.

O último passo da ciência de dados é a **comunicação**, uma parte absolutamente crítica de qualquer projeto de análise de dados. Não importa o quão bem seus modelos e visualização levaram você a entender os dados, a menos que você também possa comunicar seus resultados a outras pessoas.

Ao redor de todas essas ferramentas está a programação. A programação é uma ferramenta transversal que você usa em todas as partes do projeto. Você não precisa ser um programador especialista para ser um cientista de dados, mas aprender mais sobre programação compensa, porque se tornar um programador melhor permite automatizar tarefas comuns e resolver novos problemas com maior facilidade.

2.1 R e RStudio

A primeira coisa que você precisa fazer para iniciar o R é instalá-lo no seu computador. O R funciona em praticamente todas as plataformas disponíveis, incluindo os sistemas Windows, Mac OS X e Linux amplamente disponíveis.

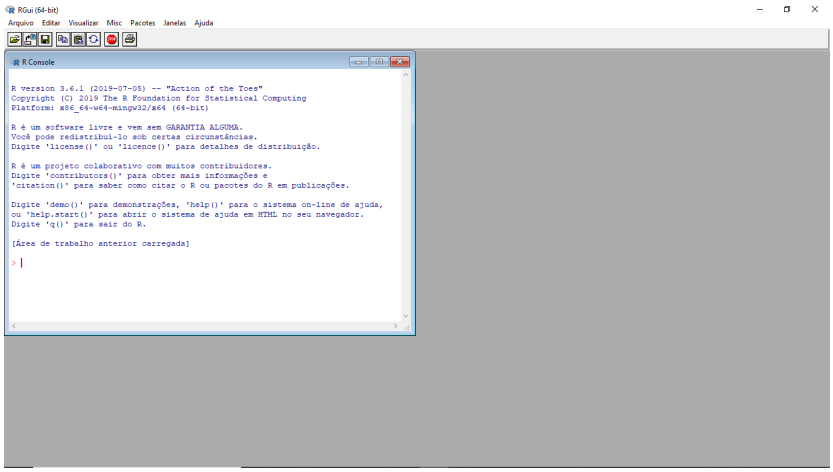
Uma nova versão principal do R sai uma vez por ano, e há 2 ou 3 versões menores a cada ano. É uma boa ideia atualizar regularmente. A atualização pode ser um pouco complicada, especialmente para as versões principais, que exigem a reinstalação de todos os seus pacotes.

Há também um ambiente de desenvolvimento integrado (IDE) disponível para o R, construído pelo RStudio. IDE, do inglês **Integrated Development Environment** ou Ambiente de Desenvolvimento Integrado, é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo. O RStudio é atualizado duas vezes por ano. Quando uma nova versão estiver disponível, o RStudio informará você.

Você pode ver como instalar o R e o RStudio aqui:

- Instalando o RStudio

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

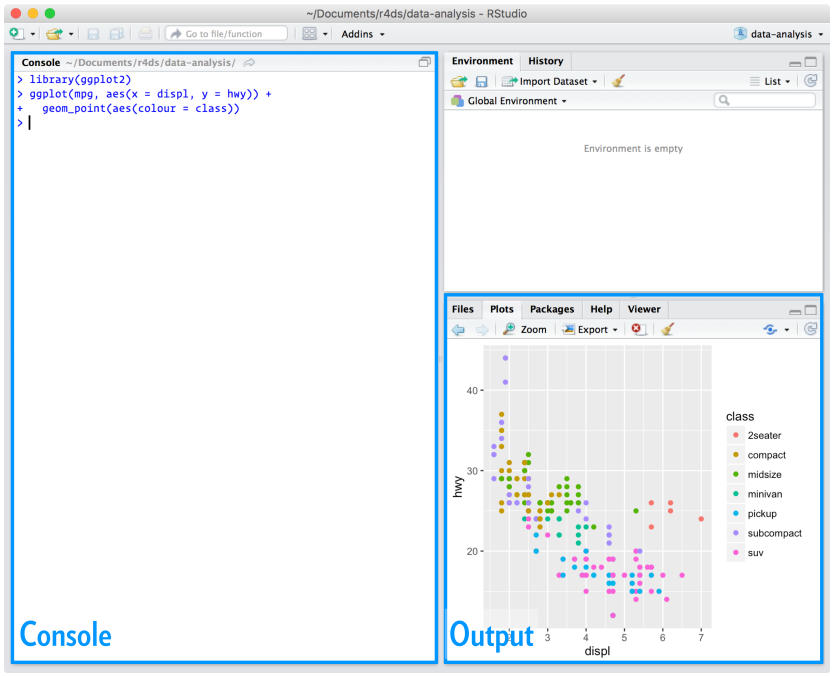


Experimente um comando: 2+2, cujo output é 4:

```
2 + 2
```

```
## [1] 4
```

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



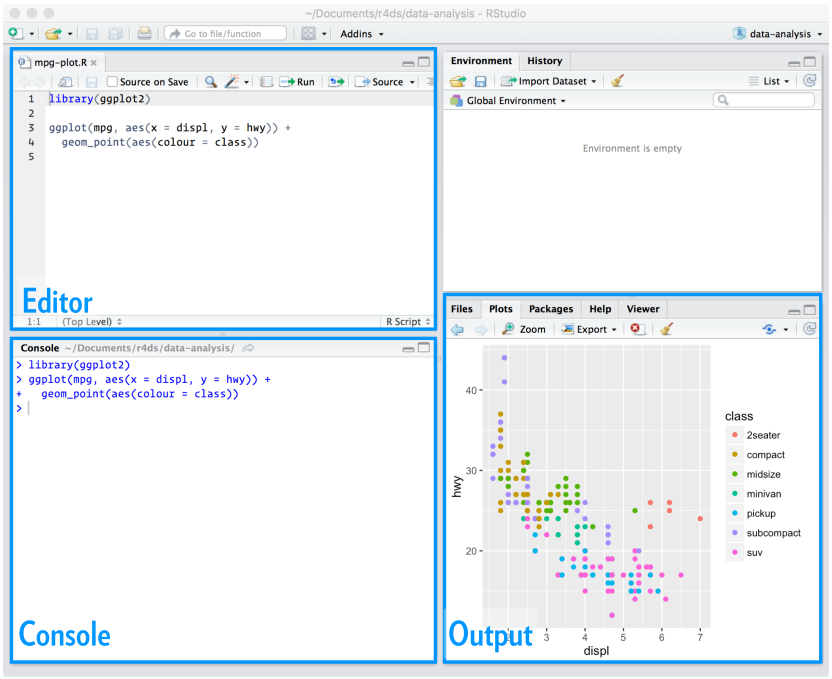
Do lado esquerdo fica o console, onde os comandos podem ser digitados e onde ficam os *outputs*.

No lado superior direito há duas abas:

-i) *Environment*, que é onde ficam armazendos os objetos criados, bases de dados importadas, etc; e

-ii) *History*, onde ficam o histórico dos comandos executados.

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:



No *script* você pode digitar comandos a serem executados e também comentários.

2.2 Objetos R

R possui cinco classes básicas ou "atômicas" de objetos:

- character
- numeric (real numbers)

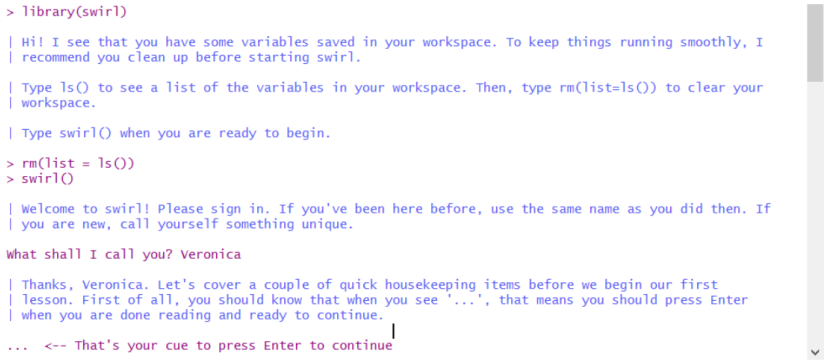
- integer
- complex
- logical (True/False)

O tipo mais básico de objeto R é um vetor. Vetores vazios podem ser criados com a função `vector()`. Existe apenas uma regra sobre vetores em R, que um vetor pode conter apenas objetos da mesma classe.

Mas é claro que, como qualquer boa regra, há uma exceção, que é uma lista , que abordaremos um pouco mais tarde. Uma lista é representada como um vetor, mas pode conter objetos de diferentes classes. De fato, geralmente é por isso que os usamos.

2.3 *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 do projeto, veja http://swirlstats.com/. Em http://swirlstats.com/students. Nestes endereços de 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.*

2.4 Universo `tidyverse`

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

Os princípios fundamentais do `tidyverse` são:

- 1.Reutilizar estruturas de dados existentes;
- 2.Organizar funções simples usando o pipe;
- 3.Aderir à programação funcional;
- 4.Projetado para ser usado por seres humanos.

Assim como o processo típico do passo a passo apresentando anteriormente para análise de dados, o `tidyverse` é a ferramenta que o ajuda eficientemente a executar este processo.

```
library(tidyverse) #Carregar o pacote.
tidyverse_logo() #Logo
```

```
## * __  _    __   .    o          *  .
##  / /_(_)__/ /_ ___  _____ _______ ___
## / __/ / _  / // / |/ / -_) __(_-</ -_)
## \__/_/\_,_/\_, /|___/\__/_/ /___/\__/
##      *  . /___/      o      .       *
```

Chapter 3

O que é Dados organizados?

> "Famílias felizes são todas iguais; toda família infeliz é infeliz à sua maneira." – Leo Tolstoi

> "Os conjuntos de dados organizados são todos iguais, mas todos os conjuntos de dados confusos são confusos à sua maneira." – Hadley Wickham

Você vai precisar instalar os pacotes `tidyr`, `devtools` e `DSR`. Para instalar `tidyr` e `devtools`, abra o RStudio e execute o comando:

```
install.packages(c("tidyr", "devtools"))
```

`DSR` é uma coleção de conjuntos de dados. Para instalar DSR, execute o comando:

```
devtools::install_github("garrettgman/DSR")
```

Os dados tabulares podem ser organizados de várias maneiras. Os conjuntos de dados abaixo mostram os mesmos dados organizados de quatro maneiras diferentes, sendo que possuem as mesmas variáveis: país, ano, população e casos. Mas cada conjunto organiza os valores em forma de layout diferente.

```
library(DSR)
# Primeiro conjunto de dados.
table1
```

```
## # A tibble: 6 x 4
```

```
##   country      year  cases population
##   <fct>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

```
# Segundo conjunto de dados.
table2
```

```
## # A tibble: 12 x 4
##    country      year key             value
##    <fct>       <int> <fct>           <int>
##  1 Afghanistan  1999 cases             745
##  2 Afghanistan  1999 population   19987071
##  3 Afghanistan  2000 cases            2666
##  4 Afghanistan  2000 population   20595360
##  5 Brazil       1999 cases           37737
##  6 Brazil       1999 population  172006362
##  7 Brazil       2000 cases           80488
##  8 Brazil       2000 population  174504898
##  9 China        1999 cases          212258
## 10 China        1999 population 1272915272
## 11 China        2000 cases          213766
## 12 China        2000 population 1280428583
```

```
# Terceiro conjunto de dados.
table3
```

```
## # A tibble: 6 x 3
##   country      year rate
##   <fct>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

O último conjunto de dados é uma coleção de duas tabelas.

```
# Quarto conjunto de dados.
table4 # cases
```

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
##   <fct>        <int>  <int>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

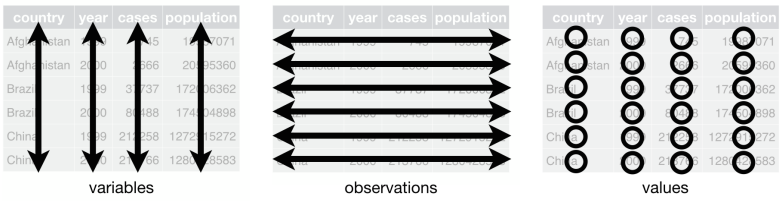
```
table5 # population
```

```
## # A tibble: 3 x 3
##   country          `1999`     `2000`
##   <fct>             <int>      <int>
## 1 Afghanistan   19987071   20595360
## 2 Brazil       172006362  174504898
## 3 China       1272915272 1280428583
```

R segue um conjunto de convenções que tornam um layout de dados tabulares muito mais fácil de trabalhar do que outros. Seus dados serão mais fáceis de trabalhar no R se seguirem três regras:

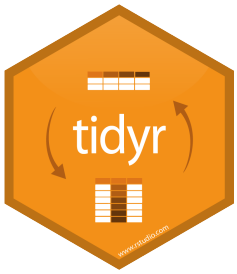
- 1.Cada variável no conjunto de dados é colocada em sua própria coluna;
- 2.Cada observação é colocada em sua própria linha;
- 3.Cada valor é colocado em sua própria célula.

Os dados que satisfazem essas regras são conhecidos como dados organizados. Observe que `table1` são dados organizados.



Em `table1`, cada variável é colocada em sua própria coluna, cada observação em sua própria linha e cada valor em sua própria célula.

3.1 Breve Introdução ao `tidyr`



O pacote tidyr tem como principal objetivo transformar um data frame para o formato tidy, ou limpo.

De acordo com as regras ditas anteriormente, um dado limpo é aquele com formato *long*, ou seja, com mais linhas. O outro formato é chamado de *wide*, com mais colunas. No caso deste exemplo, ano é uma variável, logo é necessário existir uma coluna com os valores de ano. O valor relacionado a UF naqueles anos também é outra variável, então precisa de uma coluna pra representá-lo. Além disso, a própria UF precisa de uma coluna.

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.

3.1.1 `gather`

Vamos criar um `tibble` no formato *wide* e transformá-lo em um dado *tidy*:

```
library(tibble)
tb <- tibble(uf = c("RJ", "SP"), `2017` = c(10, 11), `2018` = c(11, 10))
tb
```

```
## # A tibble: 2 x 3
##   uf    `2017` `2018`
##   <chr>  <dbl>  <dbl>
## 1 RJ        10     11
## 2 SP        11     10
```

Chapter 4

Manipulando Data Frames com `dplyr`

4.1 Data Frames

4.2 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

4.3 Gramática do `dplyr`

Alguns dos principais "verbos" fornecidos pelo `dplyr` são:

`-select`: retorna um subconjunto das colunas de um data.frames, usando uma notação flexível;

`-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.

4.4 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*.

4.5 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:

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

Para instalar a partir do GitHub, execute:

```
library(devtools) #carregar o pacote 'devtools' antes.
devtools::install_github("hadley/dplyr")
```

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

```
library(dplyr)
```

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.

4.6 select()

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.

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

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

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. Uma das formas para facilitar o processo de descompactação do arquivo pelo R é:

```
# objeto caracter, endereço do arquivo.
fileURL <- "http://www.biostat.jhsph.edu/~rpeng/leanpub/rprog/chicago_data.zip"

#Esta função pode ser usada para baixar um arquivo da Internet.
download.file(fileURL, destfile = "data/chicago.rds", method = "curl", extra='-L')
```

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.

```
dim(chicago)
```

```
## [1] 6940    8
```

```
str(chicago)
```

```
## 'data.frame':    6940 obs. of  8 variables:
##  $ city      : chr  "chic" "chic" "chic" "chic" ...
##  $ tmpd      : num  31.5 33 33 29 32 40 34.5 29 26.5 32.5 ...
##  $ dptp      : num  31.5 29.9 27.4 28.6 28.9 ...
##  $ date      : Date, format: "1987-01-01" "1987-01-02" ...
##  $ pm25tmean2: num  NA NA NA NA NA NA NA NA NA NA ...
##  $ pm10tmean2: num  34 NA 34.2 47 NA ...
##  $ o3tmean2  : num  4.25 3.3 3.33 4.38 4.75 ...
##  $ no2tmean2 : num  20 23.2 23.8 30.4 30.3 ...
```

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
```

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
```

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`.

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
```

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:

- 1.`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
```

- 2.`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
```

- 3.`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
```

- 4.`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.

```
subset6 <- select(chicago, matches(c(".m."), names(chicago)))
head(subset6)
```

```
##   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
```

- 5.`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 0.501  0.326   0.482 0.887  0.364  0.780  0.479  0.0956
##  2 0.262  0.680   0.558 0.955  0.344  0.675  0.696  0.572
##  3 0.151  0.00109 0.510 0.803  0.339  0.620  0.389  0.231
##  4 0.720  0.986   0.229 0.493  0.177  0.351  0.0609 0.933
```

```
##  5 0.324  0.760   0.942 0.479  0.608  0.446  0.732  0.811
##  6 0.225  0.0249  0.702 0.163  0.489  0.670  0.623  0.0737
##  7 0.248  0.329   0.630 0.758  0.268  0.619  0.130  0.0993
##  8 0.736  0.531   0.463 0.0824 0.811  0.0620 0.152  0.267
##  9 0.410  0.700   0.646 0.546  0.0725 0.440  0.344  0.849
## 10 0.0221 0.107   0.496 0.356  0.130  0.359  0.576  0.880
```

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

```
## # A tibble: 10 x 3
##        V4     V5     V6
##     <dbl>  <dbl>  <dbl>
##  1 0.501  0.780  0.0956
##  2 0.262  0.675  0.572
##  3 0.151  0.620  0.231
##  4 0.720  0.351  0.933
##  5 0.324  0.446  0.811
##  6 0.225  0.670  0.0737
##  7 0.248  0.619  0.0993
##  8 0.736  0.0620 0.267
##  9 0.410  0.440  0.849
## 10 0.0221 0.359  0.880
```

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
```

4.6.1 Exercícios

- 1. Obter um subconjunto com as seguintes variáveis selecionadas: `dep_time`, `dep_delay`, `arr_time`, e `arr_delayde` do banco `flights`.
- 2. O que acontece se você incluir o nome de uma variável várias vezes em uma `select()` chamada?
- 3. O que a `one_of()` função faz? Por que pode ser útil em conjunto com esse vetor?

vars <- c("year", "month", "day", "dep_delay", "arr_delay")

4.7 `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
```

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.

4.7.1 Exercícios

4.8 `mutate()`

Chapter 5

Transformação de Dados com `dplyr`

5.1 Introdução

A visualização é uma ferramenta importante para a geração de *insights*, mas é raro você obter os dados exatamente da forma correta de que precisa. Freqüentemente, você precisará criar algumas novas variáveis ou resumos, ou talvez apenas queira renomear as variáveis ou reordenar as observações para tornar os dados um pouco mais fáceis de trabalhar. Você aprenderá como fazer tudo isso (e muito mais!) Neste capítulo, que ensinará como transformar seus dados usando o pacote `dplyr` e um novo conjunto de dados em voos partindo de Nova York em 2013.

5.2 Pré-requisitos

Neste capítulo, vamos nos concentrar em como usar o pacote `dplyr`, outro membro central do `tidyverse`. Ilustraremos as ideias principais usando dados do pacote `nycflights13` e usaremos o `ggplot2` para nos ajudar a entender os dados.

```
library(nycflights13)
library(tidyverse) # ou isoladamente: library(dplyr).
```

5.3 nycflights13

Esse data frames contém todos os 336.776 vôos que partiram de Nova York em 2013. Os dados são do Bureau of Transportation Statistics dos EUA e estão documentados em `?flights`.

```
flights
```

```
## # A tibble: 336,776 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>
##  1  2013     1     1      517            515         2      830
##  2  2013     1     1      533            529         4      850
##  3  2013     1     1      542            540         2      923
##  4  2013     1     1      544            545        -1     1004
##  5  2013     1     1      554            600        -6      812
##  6  2013     1     1      554            558        -4      740
##  7  2013     1     1      555            600        -5      913
##  8  2013     1     1      557            600        -3      709
##  9  2013     1     1      557            600        -3      838
## 10  2013     1     1      558            600        -2      753
## # ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
## #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
## #   minute <dbl>, time_hour <dttm>
```

Para ver todo o conjunto de dados, você pode executar o `View(flights)` que abrirá o conjunto de dados no visualizador do RStudio. Imprime de forma distinta do data frame, porque é um `tibble`. O que é um *tibble*? *Tibbles* são similares aos *data frames*, porém diferentes em dois aspectos: **impressão** e **indexação**

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. Além disso, cada coluna apresenta o seu tipo, algo semelhante ao apresentado quando utilizamos a função `str()`. A segunda diferença, não menos importante, é a forma de indexação. Para indexar um `tibble` devemos utilizar o nome completo da variável que desejamos. Caso contrário, ocorrerá um erro.

Ainda sobre a indexação, sempre que indexarmos um `tibble` usando `[`, o resultado será outro `tibble`. Usando `[[` o resultados será um vetor.

Abreviações de letras sob os nomes das colunas. Eles descrevem o tipo de cada variável:

-`int` significa números inteiros;

`-dbl` significa números duplos ou reais;

`-chr` significa vetores de caracteres ou seqüências de caracteres;

`-dttm` significa data e hora (uma data + uma hora).

`-lgl` significa vetores lógicos que contêm apenas `TRUE` ou `FALSE`;

`-fctr` significa fatores, que R usa para representar variáveis categóricas com valores possíveis fixos.

`-data` significa data.

Existem outros tipos comuns de variáveis que não são usadas neste conjunto de dados.

Em síntese, *data frames* são tabelas de dados. Em seu formato, são bem parecidos com as matrizes, no entanto, possuem algumas diferenças significativas. Podemos idealizar os *data frames* como sendo matrizes em que cada coluna pode armazenar um tipo de dado diferente. Logo, estamos lidando com um objeto bem mais versátil do que as matrizes e os vetores.

Uma das funções básicas mais importantes para começarmos a trabalhar com *data frames* é a `str()`. Essa função dá uma visão clara da estrutura do nosso objeto, bem como informa os tipos de dados existentes.

5.3.1 Exercícios

- 1.Qual a diferença entre uma matriz e um data frame no R?
- 2.Os data frames podem ser indexados com a mesma sintaxe utilizada para matrizes?
- 3.Qual função básica que utilizamos para verificar a estrutura dos dados de um data frame?

5.4 Operador *pipe* %>%

Os tubos são uma ferramenta poderosa para expressar claramente uma sequência de várias operações. O pipe, `%>%` vem do pacote `magrittr` de Stefan Milton

Bache. Pacotes no `tidyverse` carregam `%>%` automaticamente, para que normalmente não carregue o magrittr explicitamente.

Para começar a utilizar o *pipe*, instale e carregue o pacote `magrittr`.

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

Para mais informações sobre o *pipe*, outros operadores relacionados e exemplos de utilização, visite a página Ceci n'est pas un pipe

5.4.1 Exercícios

- 1. Reescreva a expressão abaixo utilizando o `%>%`.

round(mean(sum(1:10)/3), digits = 1)

Dica: utilize a função `magrittr::divide_by()`. Veja o help da função para mais informações.

- 2. Reescreva o código abaixo utilizando o `%>%`.

x <- rnorm(100) x.pos <- x[x>0] media <- mean(x.pos) saida <- round(media, 1)

- 3. Sem rodar, diga qual a saída do código abaixo. Consulte o `help` das funções caso precise.

2 %>% add(2) %>% c(6, NA) %>% mean(na.rm = T) %>% equals(5)

5.5 `filter()`

`filter()` permite agrupar observações com base em seus valores. O primeiro argumento da função é o nome do data frames. Por exemplo, podemos selecionar todos os valores

5.5.1 Comparações

5.5.2 Operadores Lógicos

5.5.3 Valores Ausentes

5.5.4 Exercícios

5.6 `arrange()`

5.6.1 Exercícios

5.7 `select()`

5.7.1 Exercícios

Chapter 6

Conclusão

Peng, R.D. and Welty, L.J. (2004) The NMMAPSdata package. R News 4(2).

Wood, S.N. (2006, 2017) Generalized Additive Models: An Introduction with R.

Peng, R.D., Exploratory Data Analysis with R (2016) This version was published on 2016-07-20. This book is for sale at http://leanpub.com/exdata