Capítulo **2**

NESTE CAPÍTULO

- **Obtendo R e RStudio**
- **Trabalhando com RStudio**
- **Aprendendo as funções R**
- **Aprendendo as estruturas de R**
- **Trabalhando com pacotes**
- **Formando as fórmulas de R**
- **Lendo e escrevendo arquivos**

R: O que Faz e Como o Faz

R é uma linguagem de computador. É uma ferramenta para fazer computações e cálculos que estabelecem o cenário da análise estatística e da tomada de decisão. Um aspecto importante da análise estatística é apresentar os resultados de maneira compreensível. Por isso, os gráficos são um componente importante de R.

Ross Ihaka e Robert Gentleman desenvolveram R na década de 1990 na Universidade de Auckland, Nova Zelândia, apoiados pela Foundation for Statistical Computing, e R está cada dia mais popular.

O RStudio é um ambiente de desenvolvimento integrado (IDE) de código aberto para criar e executar código R. Ele está disponível em versões para Windows, Mac e Linux. Embora uma IDE não seja necessária para trabalhar com R, o RStudio facilita *muito* a vida.

Baixando o R e o RStudio

Vamos começar pelo começo. Faça o download de R no Comprehensive R Archive Network (CRAN). Em seu navegador, digite este endereço (conteúdo em inglês), se usar Windows:

```
cran.r-project.org/bin/windows/base/
```

Digite este se usar Mac (conteúdo em inglês):

```
cran.r-project.org/bin/macosx/
```

Clique no link para baixar R. Isso coloca o arquivo `win.exe` em seu computador, se usar Windows, ou o arquivo `.pkg`, se usar Mac. Em qualquer caso, siga os procedimentos normais de instalação. Quando estiver concluída, os usuários de Windows verão um ícone R no desktop e os de Mac o verão na pasta Application.

DICA

Ambos os endereços fornecem links úteis para FAQs. A URL relacionada ao Windows também fornece link para "Instalação e outras instruções".

Agora o RStudio. Eis o endereço (conteúdo em inglês):

www.rstudio.com/products/rstudio/download

Clique no link para baixar o instalador para seu computador e, novamente, siga os procedimentos normais de instalação.

Depois de concluir a instalação do RStudio, clique no ícone para abrir a janela mostrada na Figura 2-1.

DICA

Se você já tiver uma versão mais antiga do RStudio e fizer esse procedimento de instalação, ele atualizará para a versão mais recente (e não é preciso desinstalar a versão anterior).

O grande painel Console à esquerda executa o código R. Uma maneira de executar o código é digitá-lo diretamente no painel. Mostrarei outra maneira daqui a pouco.

Os outros dois painéis fornecem informações úteis durante o trabalho com R. O painel Environment e o painel History estão na parte superior do lado direito. A aba Environment acompanha as coisas que você cria (que R chama de *objetos*) enquanto trabalha. A aba History acompanha o código R que você digita.

DICA

Acostume-se com a palavra *objeto.* Tudo em R é um objeto.

As abas Files, Plots, Packages e Help estão no painel na parte inferior do lado direito. A aba Files mostra os arquivos criados. A aba Plots fica com os gráficos que você cria a partir dos dados. A aba Packages mostra os add-ons (chamados de *pacotes*) baixados como parte da instalação de R. Lembre-se de que "baixado" não significa "pronto para usar". Para usar os recursos dos pacotes é necessário mais um passo. E acredite, você quer usar esses pacotes.

A Figura 2-2 mostra a aba Packages. Os pacotes estão na biblioteca do usuário (que você pode ver na figura) ou na biblioteca do sistema (que é preciso rolar para ver). Eu falo mais sobre pacotes posteriormente neste capítulo.

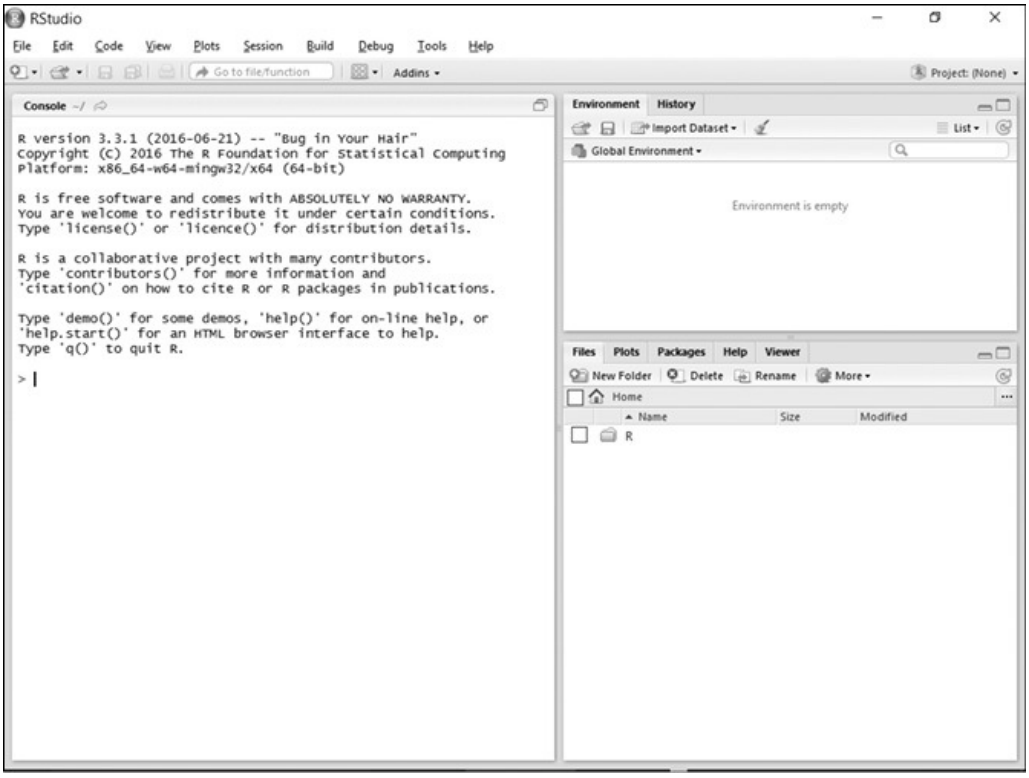


FIGURA 2-1:
O RStudio, imediatamente depois da instalação e da abertura.

A aba Help, mostrada na Figura 2-3, fornece links para muitas informações sobre R e RStudio.

Para aproveitar todo a capacidade do RStudio como IDE, clique no maior dos dois ícones no canto direito superior do painel Console. Isso muda a aparência do RStudio para que fique como na Figura 2-4.

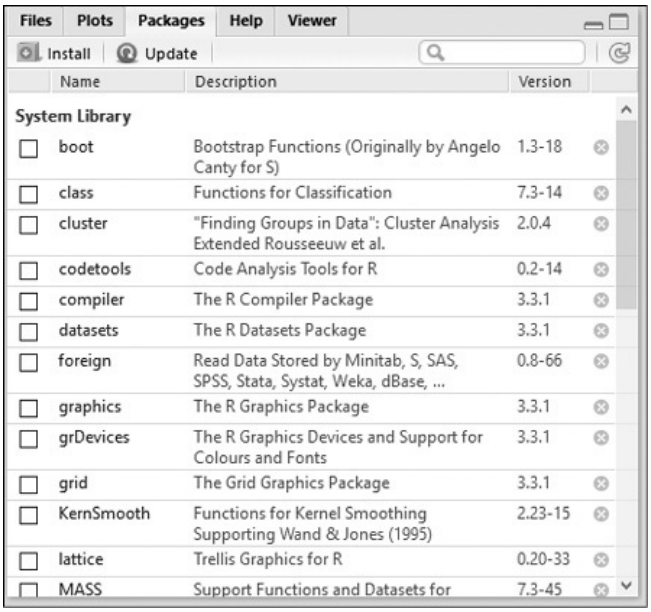


FIGURA 2-2:
Aba Packages do RStudio.

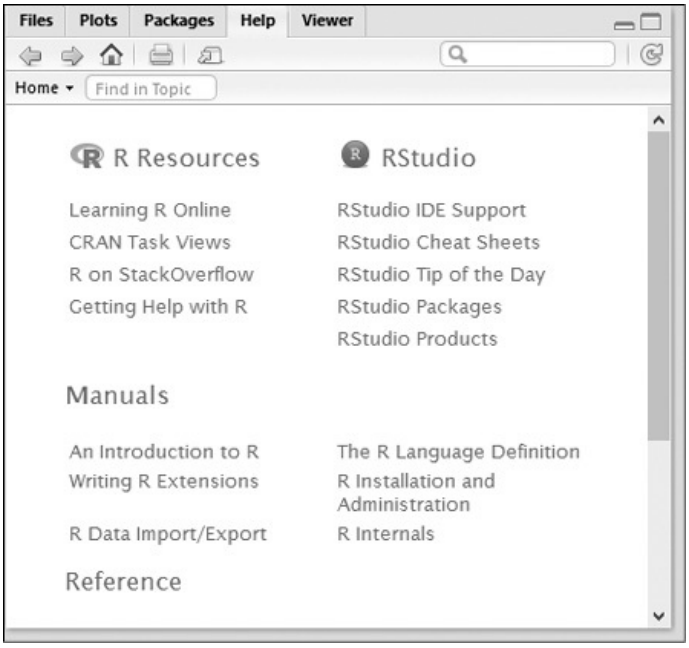


FIGURA 2-3:
Aba Help do RStudio.

O topo do painel Console é realocado para a parte inferior esquerda. O novo painel na parte superior esquerda é o Scripts. Nele você digita, edita o código e pressiona Ctrl+R (Command+Enter, no Mac), assim, o código é executado no painel Console.

DICA

Ctrl+Enter funciona da mesma forma que Ctrl+R. Também é possível selecionar

Code ⇨ Run Selected Line(s)

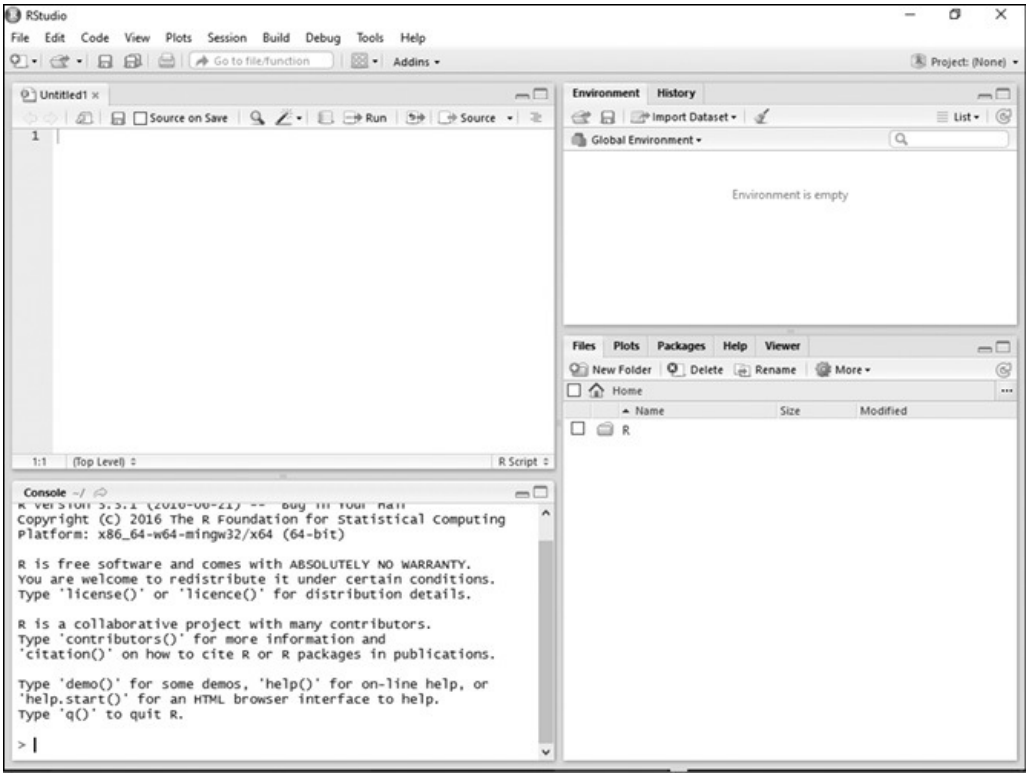


FIGURA 2-4:
RStudio depois de clicar no ícone maior no canto superior direito do painel Console.

Uma Sessão com R

Antes de começar a trabalhar, selecione

File ⇨ Save As. . .

então salve como My First R Session. Isso renomeia a aba no painel Scripts com o nome do arquivo e adiciona a extensão .R. Também faz com que o nome do arquivo (e a extensão .R) apareça na aba Files.

Diretório de trabalho

O que exatamente R salva e onde? R salva o que é chamado de *área de trabalho,* que é o ambiente no qual você trabalha. R salva a área de trabalho no *diretório de trabalho.* No Windows, o diretório de trabalho padrão é

```
C:\Users\<Nome Usuário>\Documents
```

Se algum dia você esquecer do caminho de seu diretório de trabalho, digite

```
> getwd()
```

no painel Console e R retornará o caminho na tela.

DICA

No painel Console você não digita o sinal "maior que" no começo da linha. Ele é um prompt.

Meu diretório de trabalho é este:

```
> getwd()
[1] “C:/Users/Joseph Schmuller/Documents
```

Observe para qual lado as barras estão inclinadas. Elas são o oposto do que você normalmente vê nos caminhos de arquivo do Windows. Isso porque R usa \ como um *caractere de escape*, o que significa que tudo o que vier depois de \ será algo diferente do que o normal. Por exemplo, \t em R significa *tecla Tab.*

Você também pode escrever um caminho de arquivo do Windows em R como

```
C:\\Users\\<Nome Usuário>\\Documents
```

Se quiser, é possível mudar o diretório de trabalho:

```
> setwd(<caminho arquivo>)
```

Outra maneira de mudar o diretório de trabalho é selecionar

```
Session ⇨ Set Working Directory ⇨ Choose Directory
```

Então vamos começar logo

E agora um pouco de R! Na janela Script digite

```
x <- c(3,4,5)
```

então pressione Ctrl+R.

Isso coloca a seguinte linha no painel Console:

```
> x <- c(3,4,5)
```

Como mencionei em uma Dica anterior, o sinal de "maior que" é um prompt que R fornece no painel Console. Você não o vê no painel Scripts.

O que R acabou de fazer? A flecha informa que x recebe o que está à direita. Então a flecha é o *operador de atribuição* de R.

À direita da flecha, *c* representa *concatenar,* um jeito bonito de dizer "pegue o que estiver entre parênteses e junte tudo". Então o conjunto de números 3, 4, 5 está agora atribuído a x.

R se refere a um conjunto de números como este como um *vetor*. (Eu falo mais sobre isso na seção "Estruturas de R".)

Leia essa linha de código R como "x recebe o vetor 3, 4, 5".

Digite **x** no painel Scripts e pressione Ctrl+Enter. Aqui está o que você vê no painel Console:

```
> x
[1] 3 4 5
```

O 1 entre colchetes é o rótulo do primeiro valor na linha de saída. É claro que aqui existe apenas um valor. O que acontece quando R produz diversos valores em várias linhas? Cada linha recebe um rótulo numérico entre colchetes e o número corresponde ao primeiro valor da linha. Por exemplo, se a saída consistir em 21 valores e o 18º valor for o primeiro da segunda linha, ela começará com [18].

Criar o vetor x faz a aba Environment parecer com a Figura 2-5.

FIGURA 2-5:
Aba Environment do RStudio depois de criar o vetor x.

Outra maneira de ver os objetos no ambiente é digitar

```
> ls()
```

Agora você pode trabalhar com x. Primeiro some todos os números no vetor. Digitar

```
sum(x)
```

no painel Scripts (lembre-se de pressionar Ctrl+Enter em seguida) executa a seguinte linha no painel Console:

```
> sum(x)
[1] 12
```

E a média dos números no vetor x?

É só digitar

```
mean(x)
```

no painel Scripts, que (quando seguido de Ctrl+Enter) será executado

```
> mean(x)
[1] 4
```

no painel Console.

Quando você digitar no painel Scripts ou Console, notará que surgem algumas informações úteis. À medida que ganhar experiência com o RStudio, aprenderá a usar essas informações.

Como mostro no Capítulo 5, a *variância* é uma medida do quanto um conjunto de números difere de sua média. O que exatamente é a variância e como você a calcula? Deixarei isso para o Capítulo 5. Por enquanto, aqui está como usar R para calculá-la:

```
> var(x)
[1] 1
```

Em cada caso, você digita um comando, R o avalia e exibe o resultado.

A Figura 2-6 mostra como fica o RStudio depois de todos esses comandos.

Para finalizar uma sessão, selecione File ⇨ Quit Session ou pressione Ctrl+Q. Como a Figura 2-7 mostra, uma caixa de diálogo aparece e pergunta o que você quer salvar dessa sessão. Salvar as seleções permite reabrir a sessão de onde parou na próxima vez em que abrir o RStudio (embora o painel Console não salve seu trabalho).

Bem útil esse RStudio.

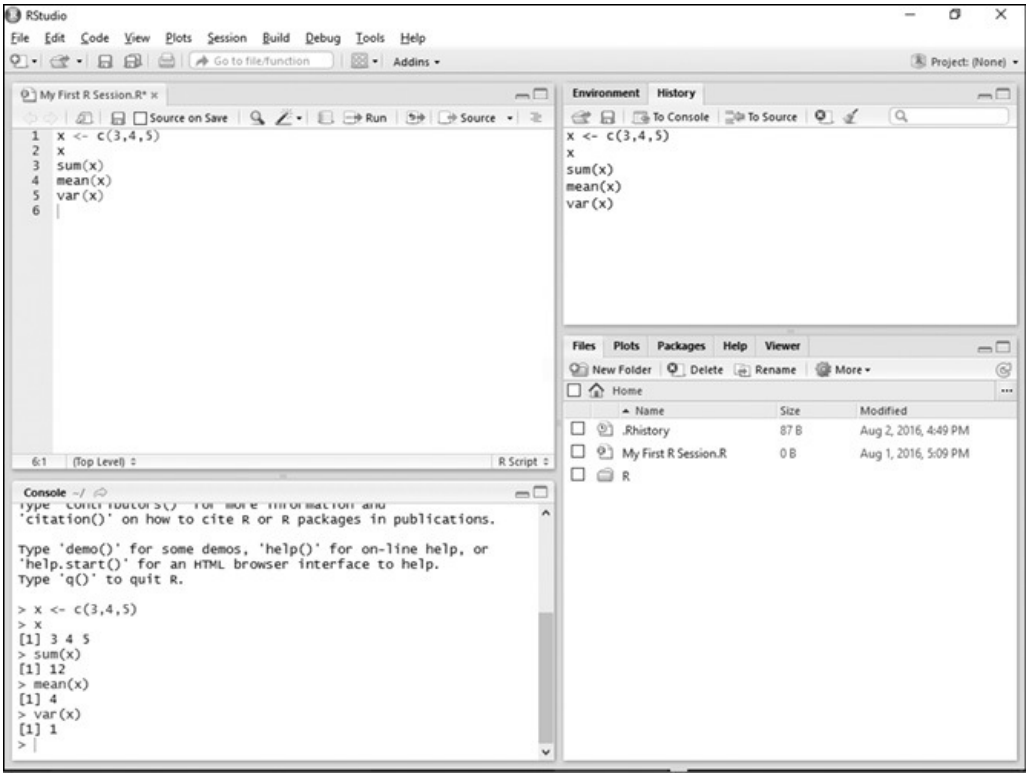


FIGURA 2-6:
O RStudio depois de criar e trabalhar com um vetor.

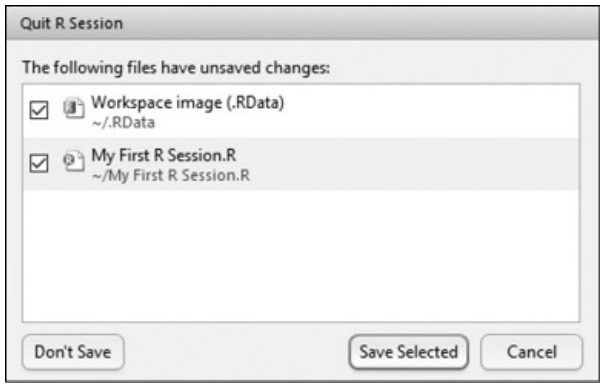


FIGURA 2-7:
Caixa de diálogo Quit R Session.

LEMBRE-SE

Avançando, na maior parte do tempo eu não informo "digite este código R no painel Scripts e pressione Ctrl+Enter" sempre que passamos por um exemplo. Só mostro o código e o resultado, como no exemplo var().

LEMBRE-SE

Além disso, às vezes eu mostro o código com o prompt > e outras sem. Geralmente mostro o prompt quando quero que você veja o código R e seus resultados. Eu não mostro o prompt quando só quero que você veja o código que criei no painel Scripts.

Dados faltantes

Nos exemplos de análise estatística que forneço, normalmente lido com os melhores cenários em que os conjuntos de dados estão em boa forma e têm todos os dados que deveriam ter.

Contudo, no mundo real, as coisas nem sempre são tão fáceis. Muitas vezes encontramos conjuntos de dados com valores faltantes por uma ou outra razão. O R indica um valor faltante como NA (Not Available ou Indisponível).

Por exemplo, aqui estão alguns dados (de um conjunto de dados bem maior) sobre a capacidade de bagagem, em metros cúbicos, de nove veículos:

```
capacity <- c(14,13,14,13,16,NA,NA,20,NA)
```

Três veículos são vans e o termo *capacidade de bagagem* não se aplica a eles, por isso os três casos de NA. Aqui está o que acontece quando você tenta descobrir a média desse grupo:

```
> mean(capacity)
[1] NA
```

Para encontrar a média, é preciso remover os NAs antes de calcular:

```
> mean(capacity, na.rm=TRUE)
[1] 15
```

Então o rm em na.rm significa "remover" e =TRUE significa "faça isso".

Caso algum dia você tenha que verificar um conjunto de pontuações de dados faltantes, a função is.na() fará isso por você:

```
> is.na(capacity)
[1] FALSE FALSE FALSE FALSE FALSE TRUE TRUE FALSE
 TRUE
```

Funções de R

Na seção anterior eu usei c(), sum(), mean() e var(). Esses são exemplos de *funções* incorporadas em R. Cada uma consiste em um nome de função seguido imediatamente de parênteses. Entre dos parênteses ficam os *argumentos*. Nesse contexto, "argumento" não significa "discussão", "confronto" ou nada parecido. É só o termo matemático para qualquer coisa em que a função opera.

LEMBRE-SE

Mesmo que uma função não tenha argumentos, ainda se devem incluir parênteses.

As quatro funções de R que mostrei são bem simples em termos de seus argumentos e saídas. No entanto, à medida que trabalhar com R, você encontrará funções que recebem mais de um argumento.

O R fornece algumas maneiras de lidar com funções de múltiplos argumentos. Uma delas é listar os argumentos na ordem em que aparecem na definição da função. O R chama isso de *correspondência posicional*.

Aqui está o que quero dizer: a função substr() recebe três argumentos. O primeiro é uma string de caracteres como "abcdefg", a que R se refere como *vetor de caracteres.* O segundo é uma posição *inicial* dentro da string (1 é a primeira posição, 2 é a segunda, e assim por diante). O terceiro é uma posição de *parada* dentro da string (um número maior ou igual à posição inicial). Na verdade, se você digitar **substr** no painel Scripts, verá uma mensagem pop-up útil como esta:

```
substr(x, start, stop)
Extract or replace substrings in a character vector
```

onde x representa o vetor de caracteres.

Essa função retorna a substring, que consiste nos caracteres entre as posições de início e fim.

Veja um exemplo:

```
> substr("abcdefg",2,4)
[1] "bcd"
```

O que acontecerá se você trocar 2 e 4?

```
> substr("abcdefg",4,2)
[1] ""
```

Este resultado é completamente compreensível. Nenhuma substring pode começar na quarta posição e terminar na segunda.

Porém se você *nomear* os argumentos, não importará a ordem na qual os coloca:

```
> substr("abcdefg",stop=4,start=2)
[1] "bcd"
```

Até isto funciona:

```
> substr(stop=4, start=2,"abcdefg")
[1] "bcd"
```

Então, quando usamos uma função, podemos colocar seus argumentos fora de ordem, se os nomeamos. O R chama isso de *correspondência de palavras-chave*, que é muito útil quando utilizamos uma função de R com muitos argumentos. Se não conseguimos lembrar da ordem, é só usar seus nomes e a função funcionará.

Se algum dia você precisar de ajuda com uma função específica, substr(), por exemplo, digite **?substr** e veja informações úteis aparecerem na aba Help.

Funções Definidas pelo Usuário

A rigor, este não é um livro sobre programação R. Mas, para ser completo, achei que deveria, pelo menos, mostrar que é possível criar suas próprias funções em R e o básico de como fazê-lo.

A forma de uma função R é

```
myfunction <- function(argument1, argument2, ... ){ statements
 return(object) }
```

Veja uma função simples para calcular a soma dos quadrados de três números:

```
sumofsquares <- function(x,y,z){
sumsq <- sum(c(x^2,y^2,z^2))
return(sumsq)
}
```

Digite esse trecho no painel Scripts e destaque-o. Depois pressione Ctrl+Enter. O trecho a seguir aparecerá no painel Console:

```
> sumofsquares <- function(x,y,z ){
+ sumsq <- sum(c(x^2,y^2,z^2))
+ return(sumsq)
+ }
```

Cada sinal de mais é um *prompt de continuação*. Indica que a linha é uma continuação da linha anterior.

E é assim que se usa a função:

```
> sumofsquares(3,4,5)
[1] 50
```

Comentários

Um *comentário* é uma maneira de fazer anotações no código. Comece um comentário com o símbolo #, que, claro, é uma *cerquilha*. (O que você está

dizendo? "Hashtag"? Certamente está de brincadeira.) Esse símbolo informa a R para ignorar tudo o que está à direita dele.

Os comentários são muito úteis para alguém que precisa ler o código que você escreveu. Por exemplo:

```
sumofsquares <- function(x,y,z){ # lista os argumentos
sumsq <- sum(c(x^2,y^2,z^2)) # realiza as operações
return(sumsq) # retorna o valor
}
```

Um aviso: Eu não adiciono comentários às linhas de código deste livro, mas forneço descrições detalhadas. Em um livro como este, acho que é a maneira mais eficaz de transmitir uma mensagem.

Como você pode imaginar, escrever funções R pode englobar MUITO mais do que expliquei aqui. Para aprender mais, confira *R For Dummies*, de Andrie de Vries e Joris Meys (John Wiley & Sons — sem publicação no Brasil).

Estruturas de R

Eu menciono na seção "Funções de R", anteriormente neste capítulo, que uma função R pode ter muitos argumentos. Também é possível que uma função R tenha muitas saídas. Para entender as possíveis saídas (e entradas) é preciso compreender as estruturas com as quais R trabalha.

Vetores

O *vetor* é a estrutura fundamental de R e eu o mostrei em exemplos anteriores. É um array de elementos de dados do mesmo tipo. Os elementos de dados em um vetor são chamados de *componentes*. Para criar um vetor, use a função c(), como fiz no exemplo anterior:

```
> x <- c(3,4,5)
```

Aqui, é claro, os componentes são números.

Em um vetor de caracteres, os componentes são strings de texto entre aspas ("Moe", "Larry", "Curly"):

```
> stooges <- c("Moe","Larry", "Curly")
```

A rigor, no exemplo substr(), “abcdefg” é um vetor de caracteres com um elemento.

Também é possível ter um vetor *lógico*, cujos elementos são TRUE e FALSE ou as abreviações T e F:

```
> z <- c(T,F,T,F,T,T)
```

Para se referir a um componente específico de um vetor, siga o nome do vetor com um número entre colchetes:

```
> stooges[2]
[1] "Larry"
```

Vetores numéricos

Além de c(), R fornece seq() e rep() como atalho para a criação de vetores numéricos.

Suponha que você queira criar um vetor de números de 10 a 30, mas não quer digitar todos eles. Veja como fazê-lo:

```
> y <- seq(10,30)
> y
[1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
[18] 27 28 29 30
```

Na minha tela, e provavelmente na sua também, todos os elementos em y aparecem em uma linha. A página impressa, no entanto, não é tão grande quanto o painel Console. Assim, separei a saída em duas linhas. Faço isso ao longo do livro, quando necessário.

R tem uma sintaxe especial para um vetor numérico cujos elementos são aumentados em 1:

```
> y <- 10:30
> y
[1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
[18] 27 28 29 30
```

Se quiser que os elementos aumentem em 2, use seq da seguinte forma:

```
> w <- seq(10,30,2)
> w
[1] 10 12 14 16 18 20 22 24 26 28 30
```

Você pode querer criar um vetor de valores repetidos. Se for o caso, rep() é a função a ser utilizada:

```
> trifecta <- c(6,8,2)
> repeated_trifecta <- rep(trifecta,4)
> repeated_trifecta
[1] 6 8 2 6 8 2 6 8 2 6 8 2
```

Outra maneira de usar rep() é fornecer um vetor como o segundo argumento. Lembre-se do exemplo anterior em que x é o vetor (3,4,5). O que acontecerá se fornecermos x como o segundo argumento de rep()?

```
> repeated_trifecta <- rep(trifecta,x)
> repeated_trifecta
[1] 6 6 6 8 8 8 8 2 2 2 2 2
```

O primeiro elemento se repetirá três vezes; o segundo, quatro vezes; e o terceiro, cinco vezes.

Matrizes

Uma *matriz* é um array bidimensional de elementos de dados do mesmo tipo. Em Estatística, as matrizes são úteis como tabelas que mantêm dados. (A Estatística avançada tem outras aplicações para matrizes que vão além do escopo deste livro.)

Há a matriz de números:

```
5  3  5  80
   0  5

1  3  6  85
0  5  0

1  4  6  90
5  0  5

2  4  7  95
0  5  0

2  5  7  10
5  0  5  0
```

ou de strings de caracteres:

```
"Moe"      "Larry"  "Curly"  "Shemp
                             "

"Groucho   "Harpo   "Chico"  "Zeppo"
"          "

"Ace"      "King"   "Queen  "Jack"
                    "
```

Os números constituem uma matriz de 5 (linhas) X 4 (colunas); a matriz de strings de caracteres é 3 X 4.

Para criar a matriz numérica 5 X 4, primeiro crie o vetor de números de 5 a 100 em acréscimos de 5:

```
> num_matrix <- seq(5,100,5)
```

Depois use a função dim() para transformar o vetor em uma matriz bidimensional:

```
> dim(num_matrix) <-c(5,4)
> num_matrix
[,1] [,2] [,3] [,4]
[1,] 5 30 55 80
[2,] 10 35 60 85
[3,] 15 40 65 90
[4,] 20 45 70 95
[5,] 25 50 75 100
```

Veja como R exibe os números da linha entre colchetes ao lado e os números da coluna entre colchetes acima.

Transpor uma matriz troca a posição das linhas e das colunas. Em R, a função t() faz isso:

```
> t(num_matrix)
[,1] [,2] [,3] [,4] [,5]
[1,] 5 10 15 20 25
[2,] 30 35 40 45 50
[3,] 55 60 65 70 75
[4,] 80 85 90 95 100
```

A função matrix() fornece outra maneira de criar matrizes:

```
> num_matrix <- matrix(seq(5,100,5),nrow=5)
> num_matrix
[,1] [,2] [,3] [,4]
[1,] 5 30 55 80
[2,] 10 35 60 85
[3,] 15 40 65 90
[4,] 20 45 70 95
[5,] 25 50 75 100
```

Se adicionarmos o argumento byrow=T, R preencherá a matriz por linhas:

```
> num_matrix <- matrix(seq(5,100,5),nrow=5,byrow=T)
> num_matrix
[,1] [,2] [,3] [,4]
[1,] 5 10 15 20
```

```
[2,] 25 30 35 40
[3,] 45 50 55 60
[4,] 65 70 75 80
[5,] 85 90 95 100
```

Como fazer referência a um componente específico da matriz? Digite o nome da matriz e, entre colchetes, o número da linha, a vírgula e o número da coluna:

```
> num_matrix[5,4]
[1] 100
```

Fatores

No Capítulo 1, descrevo quatro tipos de dados: nominal, ordinal, intervalar e de razão. Nos dados nominais, os números são apenas rótulos e sua magnitude não tem significância.

Suponha que estejamos fazendo uma pesquisa sobre a cor dos olhos das pessoas. Quando registramos a cor do olho de alguém, registramos um número: 1 = amber (âmbar), 2 = blue (azul), 3 = brown (castanho), 4 = gray (acinzentado), 5 = green (verde) e 6 = hazel (mel). Uma maneira de pensar nesse processo é considerar que a cor do olho é um *fator* e cada cor é um *nível* desse fator. Então, nesse caso, o fator cor de olho tem seis níveis.

LEMBRE-SE

Fator é o termo de R para uma variável nominal (também conhecida como *variável categórica*).

Agora imagine que você tenha usado o código numérico para tabular as cores dos olhos de 14 pessoas e transformou esses códigos em um vetor:

```
> eye_color <- c(2,2,4,1,5,5,5,6,1,3,6,3,1,4)
```

Em seguida, usou a função factor() para transformar eye_color em um fator:

```
> feye_color <- factor(eye_color)
```

Finalmente, atribuiu níveis ao fator:

```
> levels(feye_color) <- c("amber","blue",
 "brown","gray","green","hazel")
```

Agora, se examinarmos os dados de cores dos olhos em termos de níveis de fatores, eles se parecerão com o seguinte:

```
> feye_color
[1] blue blue gray amber green green green hazel amber
[10] brown hazel brown amber gray
Levels: amber blue brown gray green hazel
```

Listas

Em R, *lista* é uma coleção de objetos que não são necessariamente do mesmo tipo. Suponha que, além da cor dos olhos de cada pessoa no exemplo da seção anterior, você colete uma "pontuação de empatia" baseada em um teste de personalidade. A escala vai de 0 (menos empático) a 100 (mais empático). Veja o vetor para os dados de empatia das pessoas:

```
> empathy_score <- c(15,21,45,32,61,74,53,92,83,22,67,55,42,44)
```

Você quer combinar o vetor da cor dos olhos de forma codificada, o vetor de cor dos olhos na forma de fator e o vetor de pontuação de empatia em uma coleção chamada eyes_and_empathy. Para isso, use a função list():

```
> eyes_and_empathy <- list(eyes_code=eye_color,
  eyes=feye_color, empathy=empathy_score)
```

Observe que cada argumento foi nomeado (eyes_code, eyes e empathy). Isso faz com que R use esses nomes como os nomes da lista de componentes.

E a lista se parece com o seguinte:

```
> eyes_and_empathy
$eyes_code
[1] 2 2 4 1 5 5 5 6 1 3 6 3 1 4
$eyes
[1] blue blue gray amber green green green hazel amber
[10] brown hazel brown amber gray
Levels: amber blue brown gray green hazel
$empathy
[1] 15 21 45 32 61 74 53 92 83 22 67 55 42 44
```

Como podemos ver, R usa o cifrão ($) para indicar cada componente da lista. Então, se você quiser se referir a um componente da lista, digite o nome da lista, o cifrão e o nome do componente:

```
> eyes_and_empathy$empathy
[1] 15 21 45 32 61 74 53 92 83 22 67 55 42 44
```

Que tal focar uma pontuação específica, como a quarta? Eu acho que você já entendeu para onde estamos indo:

```
> eyes_and_empathy$empathy[4]
[1] 32
```

Listas e estatísticas

As listas são importantes porque várias funções estatísticas retornam listas de objetos. Uma função estatística é t.test(). No Capítulo 10 eu explico esse teste e a teoria por trás dele. Por ora, apenas se concentre na saída.

Uso esse teste para ver se a média das pontuações de empatia difere de um número arbitrário, 30, por exemplo. Veja o teste:

```
> t.result <- t.test(eyes_and_empathy$empathy, mu = 30)
```

Vamos examinar a saída:

```
> t.result
One Sample t-test
data: eyes_and_empathy$empathy
t = 3.2549, df = 13, p-value = 0.006269
alternative hypothesis: true mean is not equal to 30
95 percent confidence interval:
36.86936 63.98778
sample estimates:
mean of x
50.42857
```

Sem entrar em detalhes, entenda que essa saída, t.result, é uma lista. Para exibir isso, use $ para focar alguns dos componentes:

```
> t.result$data.name
[1] "eyes_and_empathy$empathy"
> t.result$p.value
[1] 0.006269396
> t.result$statistic
t
3.254853
```

Data frames

Uma lista é uma boa maneira de coletar dados. Um data frame é ainda melhor. Por quê? Quando pensamos em dados para um grupo de indivíduos, como as 14 pessoas no exemplo da seção anterior, normalmente pensamos em termos de colunas que representam as variáveis de dados (como eyes_code, eyes e empathy) e linhas que representam os indivíduos. E isso é um data frame. Se os termos *conjunto de dados* ou *matriz de dados* vêm à mente, você entendeu a ideia.

A função data.frame() trabalha com os vetores existentes para realizar a tarefa:

```
> e <- data.frame(eye_color,feye_color,empathy_score)
> e
eye_color feye_color empathy_score
1 2 blue 15
2 2 blue 21
```

```
3 4 gray 45
4 1 amber 32
5 5 green 61
6 5 green 74
7 5 green 53
8 6 hazel 92
9 1 amber 83
10 3 brown 22
11 6 hazel 67
12 3 brown 55
13 1 amber 42
14 4 gray 44
```

Quer a pontuação de empatia da sétima pessoa? Ela é

```
> e[7,3]
[1] 53
```

E que tal todas as informações da sétima pessoa?

```
> e[7,]
eye_color feye_color empathy_score
7 5 green 53
```

Editando um data frame: Parece uma planilha (mas não é)

R fornece uma maneira de modificar rapidamente um data frame. A função edit() abre uma janela Data Editor, que se parece muito com uma planilha e é possível fazer mudanças nas células. A Figura 2-8 mostra o que acontece quando você digita

```
> edit(e)
```

Data Editor

File Edit Help

	eye_color	feye_color	empathy_score	var4	var5	var6
1	2	blue	15			
2	2	blue	21			
3	4	gray	45			
4	1	amber	32			
5	5	green	61			
6	5	green	74			
7	5	green	53			
8	6	hazel	92			
9	1	amber	83			
10	3	brown	22			
11	6	hazel	67			
12	3	brown	55			
13						
14						
15						
16						
17						
18						
19						

FIGURA 2-8:
A função edit() abre uma visualização parecida com a planilha de um data frame.

Você precisa fechar a janela Data Editor para continuar.

CUIDADO

Para usuários Mac: A versão Mac do RStudio requer o sistema X Window para que algumas funções, como edit(), funcionem. A Apple costumava incluir esse recurso no Mac, mas não o faz mais. Atualmente você precisa baixar e instalar o XQuartz.

Extraindo dados de um data frame

Suponha que você queira fazer uma verificação rápida da média das pontuações de empatia para pessoas com olhos azuis em comparação com pessoas com olhos verdes e com olhos cor de mel.

A primeira tarefa é extrair as pontuações de empatia para cada cor de olhos e criar vetores:

```
> e.blue <- e$empathy_score[e$feye_color=="blue"]
> e.green <- e$empathy_score[e$feye_color=="green"]
> e.hazel <- e$empathy_score[e$feye_color=="hazel"]
```

Observe o sinal de igual duplo (==) entre os colchetes. É um *operador lógico*. Pense nele como "se e$feye_color for igual a 'blue'".

LEMBRE-SE

O sinal de igual duplo (a==b) distingue o operador lógico ("se a for igual a b") do operador de atribuição (a=b; "defina a igual a b").

Em seguida, crie um vetor das médias:

```
> e.averages <- c(mean(e.blue),mean(e.green),mean(e.hazel))
```

Depois use length() para criar um vetor do número de pontuações em cada grupo de cor de olho:

```
> e.amounts <- c(length(e.blue), length(e.green), length(e.hazel))
```

E então crie um vetor das cores:

```
> colors <- c("blue","green","hazel")
```

Agora crie um data frame de três colunas, com a cor em uma coluna, a média de empatia correspondente na seguinte e o número de pontuações em cada grupo de cores dos olhos na última coluna:

```
> e.averages.frame <- data.frame(color=colors,
average=e.averages, n=e.amounts)
```

Como foi no caso das listas, nomear os argumentos atribui nomes aos componentes do data frame (os vetores, que aparecem na tela como colunas).

Veja como isso ficará:

```
> e.averages.frame
color average n
1 blue 18.00000 2
2 green 62.66667 3
3 hazel 79.50000 2
```

Pacotes

Um *pacote* é uma coleção de funções e dados que ampliam R. Se você for um aspirante a cientista de dados e estiver buscando dados com os quais trabalhar, encontrará muitos data frames nos pacotes R. Se estiver procurando uma função estatística especializada que não esteja na instalação básica de R, provavelmente a encontrará em um pacote.

R armazena os pacotes em um diretório chamado *biblioteca*. Como um pacote vai parar na biblioteca? Clique na aba Packages no painel Files, Plots, Packages e Help. (Veja a Figura 2-2.) No próximo exemplo uso o pacote MASS, muito conhecido, que contém mais de 150 data frames de vários setores.

Se quiser ver o que há no pacote MASS, clique em MASS na aba Packages. (Ele está na seção System Library dessa aba.) Será aberta uma página na aba Help, que aparece na Figura 2-9.

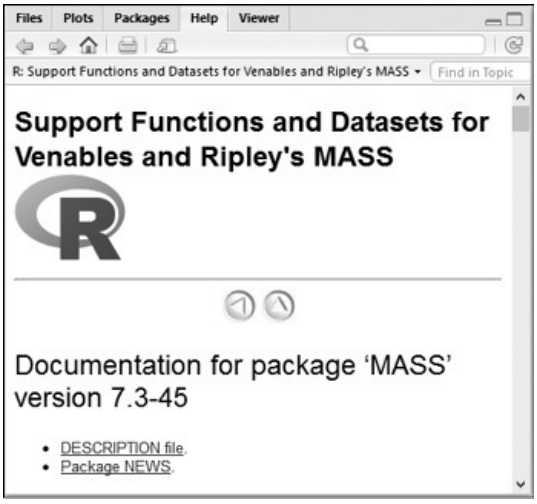


FIGURA 2-9:
Aba Help, mostrando informações sobre o pacote MASS.

Rolando a aba para baixo, é possível ver os nomes dos data frames e das funções. Clique no nome de um data frame e uma página de informações sobre ele se abrirá.

De volta à aba Packages, clique na caixa de verificação ao lado de MASS para instalar o pacote. Isso faz com que esta linha apareça na janela Console:

```
> library("MASS", lib.loc="C:/Program Files/R/R-3.3.1/library")
```

E o pacote MASS está instalado.

Um dos data frames em MASS se chama anorexia. Ele contém dados de peso de 72 jovens mulheres que são pacientes com anorexia. Cada paciente fez um dos três tipos de terapia. Como é esse data frame? Digite esta linha no painel Console

```
> edit(anorexia)
```

para abrir a janela Data Editor, mostrada na Figura 2-10.

Data Editor

File Edit Help

	Treat	Prewt	Postwt	var4	var5	var6	var7
1	Cont	80.7	80.2				
2	Cont	89.4	80.1				
3	Cont	91.8	86.4				
4	Cont	74	86.3				
5	Cont	78.1	76.1				
6	Cont	88.3	78.1				
7	Cont	87.3	75.1				
8	Cont	75.1	86.7				
9	Cont	80.6	73.5				
10	Cont	78.4	84.6				
11	Cont	77.6	77.4				
12	Cont	88.7	79.5				
13	Cont	81.3	89.6				
14	Cont	78.1	81.4				
15	Cont	70.5	81.8				
16	Cont	77.3	77.3				
17	Cont	85.2	84.2				
18	Cont	86	75.4				
19	Cont	84.1	79.5				

FIGURA 2-10:
O data frame anorexia no pacote MASS.

Parece que só está esperando que você o analise, não é? Eu não analisei nada sobre análise estatística ainda, mas você pode trabalhar um pouco nesse data frame com o que já mostrei.

O data frame fornece o peso de cada paciente antes (Prewt) e depois da terapia (Postwt). E a mudança de peso? R consegue calcular isso para cada paciente? É claro!

```
> anorexia$Postwt-anorexia$Prewt
[1] -0.5 -9.3 -5.4 12.3 -2.0 -10.2 -12.2 11.6 -7.1
[10] 6.2 -0.2 -9.2 8.3 3.3 11.3 0.0 -1.0 -10.6
[19] -4.6 -6.7 2.8 0.3 1.8 3.7 15.9 -10.2 1.7
[28] 0.7 -0.1 -0.7 -3.5 14.9 3.5 17.1 -7.6 1.6
[37] 11.7 6.1 1.1 -4.0 20.9 -9.1 2.1 -1.4 1.4
[46] -0.3 -3.7 -0.8 2.4 12.6 1.9 3.9 0.1 15.4
[55] -0.7 11.4 11.0 5.5 9.4 13.6 -2.9 -0.1 7.4
[64] 21.5 -5.3 -3.8 13.4 13.1 9.0 3.9 5.7 10.7
```

Hmmm... Você se lembra do t-test que mostrei anteriormente? Eu o utilizo aqui para ver se a mudança de peso antes/depois da terapia é diferente de 0. Espera-se que, em média, a mudança seja positiva. Aqui está o t-test:

```
> t.test(anorexia$Postwt-anorexia$Prewt, mu=0)
One Sample t-test
data: anorexia$Postwt - anorexia$Prewt
t = 2.9376, df = 71, p-value = 0.004458
```

```
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
0.8878354 4.6399424
sample estimates:
mean of x
2.763889
```

A saída do t-test mostra que a mudança média de peso foi positiva (2.763889 lbs). O alto valor de t (2.9376) junto com o baixo valor de p (0.004458), indicam que essa mudança é estatisticamente significativa. (O que *isso* significa?) Se eu falar mais, estarei colocando a carroça na frente dos bois. (Veja o Capítulo 10 para obter mais detalhes.)

E tem mais uma coisa: eu informei que cada paciente fez um dos três tipos de terapia. Uma terapia foi mais eficaz que as outras? Ou tiveram o mesmo efeito? Agora eu estaria *realmente* colocando a carroça na frente dos bois! (Essa explicação está no Capítulo 12, mas veja a seção "Fórmulas de R", um pouco mais adiante neste capítulo.)

Mais Pacotes

A comunidade R é extremamente ativa. Seus membros criam e contribuem com novos pacotes úteis o tempo todo na CRAN (Comprehensive R Archive Network). Então nem todos os pacotes R estão na aba Packages do RStudio.

Quando descobrir um novo pacote que você acha útil, será fácil instalá-lo em sua biblioteca. Eu ilustro isso ao instalar o ggplot2, um pacote útil que amplia os recursos gráficos de R.

Uma maneira de instalá-lo é com a aba Packages. (Veja a Figura 2-2.) Clique no ícone Install no canto superior esquerdo da aba. Isso abre a caixa de diálogo Install Packages, mostrada na Figura 2-11.

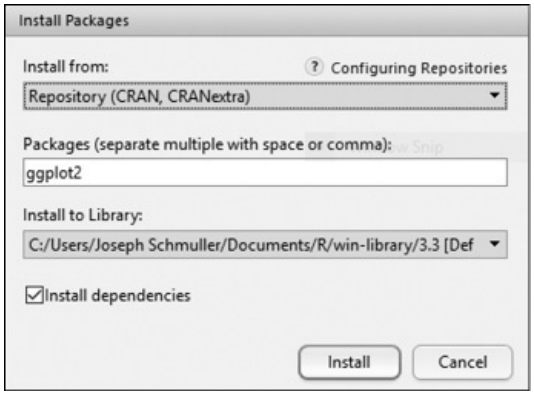


FIGURA 2-11:
Caixa de diálogo Install Packages.

DICA

Outra maneira de abrir a caixa de diálogo Install Packages é selecionar Install Packages no menu Tools na barra de menu na parte superior do RStudio.

No campo Packages, digitei ggplot2. Clique em Install e a seguinte linha aparecerá no painel Console:

```
> install.packages("ggplot2")
```

No entanto, é difícil ver essa linha, porque várias outras coisas acontecem imediatamente no painel Console e nas barras de status da tela. Quando tudo terminar, ggplot2 estará na aba Packages. O passo final é clicar na caixa de verificação ao lado de ggplot2 para colocá-lo na biblioteca. Depois você poderá usar o pacote. A Figura 2-12 mostra a aba Packages com ggplot2 e a caixa de verificação selecionada.

Ao clicar na caixa de verificação, a linha a seguir aparece no painel Console:

```
> library("ggplot2", lib.loc="~/R/win-library/3.3")
```

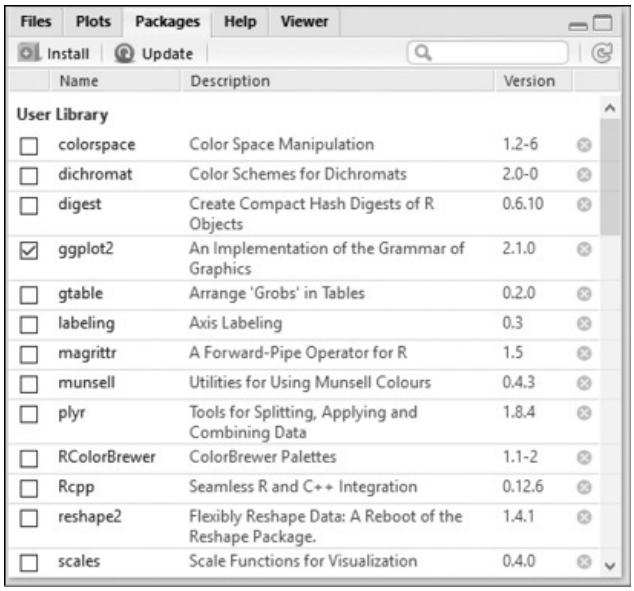


FIGURA 2-12:
A aba Packages depois de instalar ggplot2 e colocá-lo na biblioteca.

Outra maneira de começar o processo de instalação é digitar

```
> install.packages("ggplot2")
```

diretamente no painel Console.

Fórmulas de R

No Capítulo 1 eu menciono as variáveis independentes e dependentes. Aponto que, em um experimento, uma variável independente é o que um

pesquisador manipula e uma variável dependente é o que o pesquisador mede. No exemplo anterior da anorexia, Treat (tipo de terapia) é a variável independente e Postwt-Prewt (peso depois da terapia menos peso antes da terapia) é a variável dependente. Em termos práticos, "manipular" significa que o pesquisador atribuiu aleatoriamente cada paciente de anorexia a uma das três terapias.

Em outros tipos de estudos, o pesquisador não pode manipular uma variável independente. Em vez disso, ele anota os valores da variável independente que ocorrem naturalmente e avalia seus efeitos em uma variável dependente. No exemplo anterior da cor dos olhos e empatia, a cor é a variável independente e a pontuação de empatia é a variável dependente.

A *fórmula* de R incorpora esses conceitos e é a base de muitas funções estatísticas e gráficas de R. Esta é a estrutura básica de uma fórmula R:

```
function(dependent_var ~ independent_var, data=data_frame)
```

Leia o operador til (~) como "é dependente de".

O data frame anorexia fornece um exemplo. Para analisar a diferença na eficácia das três terapias de anorexia, eu usaria uma técnica chamada *análise de variância*. (Lá vou eu colocar a carroça na frente dos bois!) A função R para isso se chama aov() e é utilizada da seguinte forma:

```
> aov(Postwt-Prewt ~ Treat, data=anorexia)
```

Porém esse é só o começo da análise. O Capítulo 12 tem todos os detalhes, bem como o raciocínio estatístico por trás deles.

Lendo e Escrevendo

Antes de finalizar este capítulo sobre os recursos de R, preciso informar como importar dados de outros formatos e também como exportá-los para outros formatos.

A forma geral de uma função R para ler um arquivo é

```
> read.<format>(“File Name”, arg1, arg2, ...)
```

A forma geral de uma função R para escrever dados em um arquivo é

```
> write.<format>(dataframe, “File Name”, arg1, arg2, ...)
```

Nesta seção, trato de planilhas, arquivos CSV (valores separados por vírgulas) e arquivos de texto. <format> pode ser xlsx, csv ou table. Os argumentos depois de “File Name”são opcionais e variam dependendo do formato.

Planilhas

As informações nesta seção serão importantes se você leu meu livro clássico, *Análise Estatística com Excel Para Leigos* (Alta Books). (Tudo bem, essa foi uma citação descarada do meu clássico.) Se você tiver dados em planilhas e quiser analisá-los com R, preste muita atenção.

A primeira coisa a fazer é o download do pacote xlsx e colocá-lo na biblioteca. Confira a seção "Mais Pacotes", anteriormente neste capítulo, para obter mais informações sobre como fazer isso.

Em meu drive C, tenho uma planilha chamada Scores em uma pasta Spreadsheets. Está em Sheet1 da planilha. Ela contém as notas dos testes de Matemática e Ciência de dez alunos.

O código para ler essa planilha em R é

```
> scores_frame <- read.xlsx("C:/Spreadsheets/Scores.xlsx",
  sheetName="Sheet1")
```

Veja o data frame:

```
> scores_frame
Student Math_Score Science_Score
1 1 85 90
2 2 91 87
3 3 78 75
4 4 88 78
5 5 93 99
6 6 82 89
7 7 67 71
8 8 79 84
9 9 89 88
10 10 98 97
```

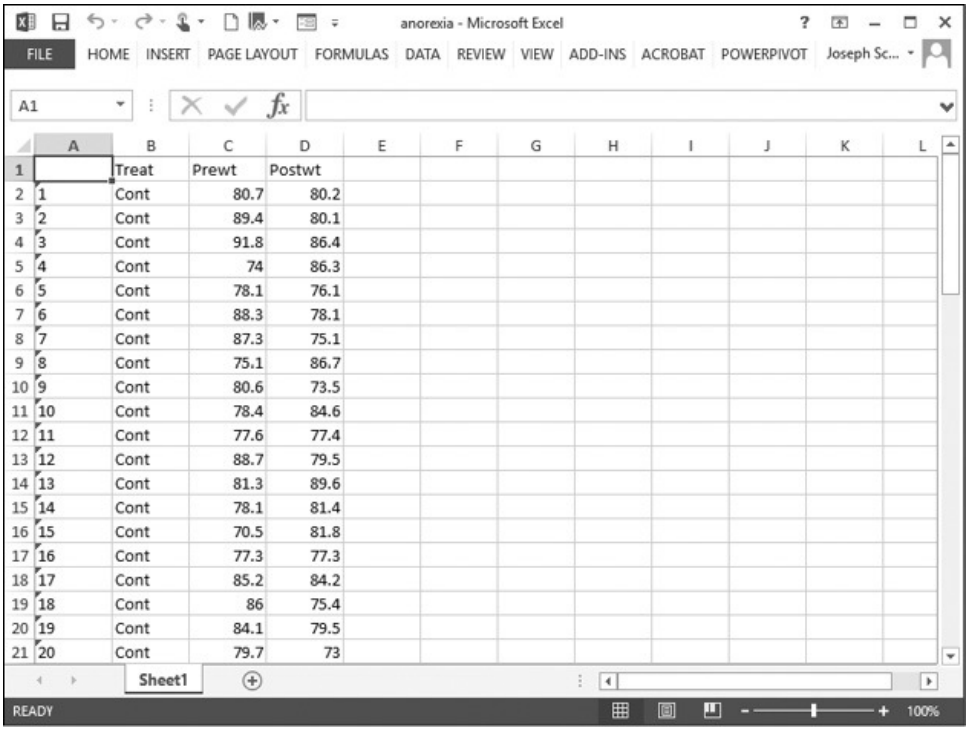
Como é o caso de qualquer data frame, se você quiser as notas de Matemática do quarto aluno, bastará fornecer:

```
> scores_frame$Math_Score[4]
[1] 88
```

O pacote xlsx também permite escrever em uma planilha. Então, se quiser que seus amigos Excelcêntricos vejam o data frame anorexia, faça o seguinte:

```
> write.xlsx(anorexia,"C:/Spreadsheets/anorexia.xlsx")
```

Essa linha coloca o data frame em uma planilha na pasta indicada no drive C. Caso não acredite em mim, a Figura 2-13 mostra como fica a planilha.



		Treat	Prewt	Postwt
1	1	Cont	80.7	80.2
2	2	Cont	89.4	80.1
3	3	Cont	91.8	86.4
4	4	Cont	74	86.3
5	5	Cont	78.1	76.1
6	6	Cont	88.3	78.1
7	7	Cont	87.3	75.1
8	8	Cont	75.1	86.7
9	9	Cont	80.6	73.5
10	10	Cont	78.4	84.6
11	11	Cont	77.6	77.4
12	12	Cont	88.7	79.5
13	13	Cont	81.3	89.6
14	14	Cont	78.1	81.4
15	15	Cont	70.5	81.8
16	16	Cont	77.3	77.3
17	17	Cont	85.2	84.2
18	18	Cont	86	75.4
19	19	Cont	84.1	79.5
20	20	Cont	79.7	73

FIGURA 2-13:
O data frame anorexia, exportado para uma planilha do Excel.

Arquivos CSV

As funções para ler e escrever arquivos CSV e de texto estão na instalação de R, portanto não são necessários outros pacotes adicionais. Um arquivo CSV é igual a uma planilha quando aberto no Excel. Na verdade, criei um arquivo CSV para a planilha Scores salvando-a como um arquivo CSV na pasta CSVFiles no drive C. (Para ver todas as vírgulas, é preciso abrir em um editor de texto, como o Notepad++.)

Veja como ler esse arquivo CSV no R:

```
> read.csv("C:/CSVFiles/Scores.csv")
Student Math_Score Science_Score
1 1 85 90
2 2 91 87
3 3 78 75
4 4 88 78
5 5 93 99
6 6 82 89
7 7 67 71
8 8 79 84
9 9 89 88
10 10 98 97
```

Para escrever o data frame anorexia em um arquivo CSV:

```
> write.csv(anorexia,"C:/CSVFiles/anorexia.csv")
```

Arquivos de texto

Se você tem dados armazenados em arquivos de texto, p R pode importá-los para os data frames. A função read.table() faz isso. Eu armazenei os dados Scores como um arquivo de texto em um diretório chamado TextFiles. Veja como o R os transforma em um data frame:

```
> read.table("C:/TextFiles/ScoresText.txt", header=TRUE)
Student Math_Score Science_Score
1 1 85 90
2 2 91 87
3 3 78 75
4 4 88 78
5 5 93 99
6 6 82 89
7 7 67 71
8 8 79 84
9 9 89 88
10 10 98 97
```

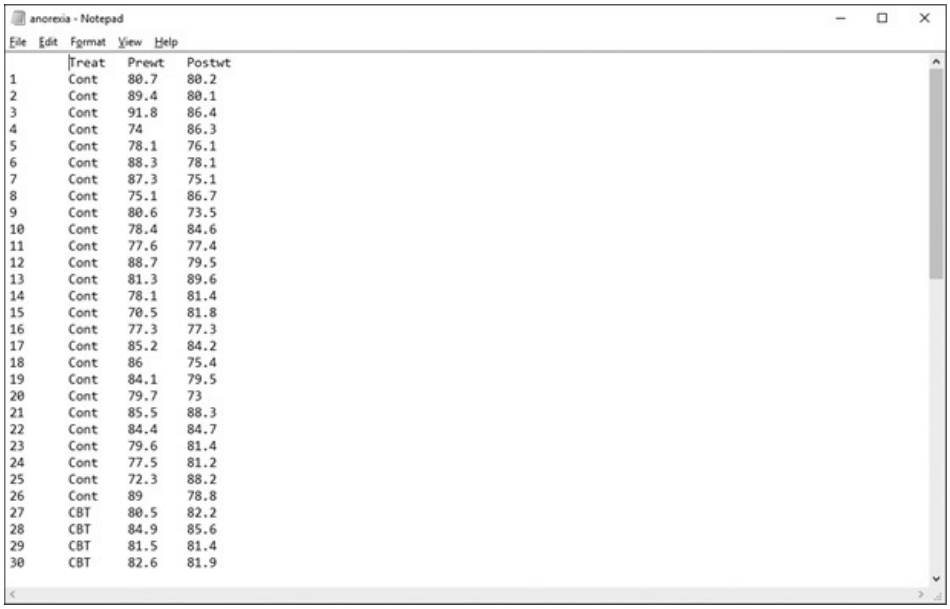
O segundo argumento (header=TRUE) faz com que R saiba que a primeira linha do arquivo contém os cabeçalhos das colunas.

Use write.table() para escrever o data frame anorexia em um arquivo de texto:

```
> write.table(anorexia, "C:/TextFiles/anorexia.txt", quote = FALSE,
  sep = "\t")
```

Isso coloca o arquivo anorexia.txt na pasta TextFiles no drive C. O segundo argumento (quote = FALSE) garante que as aspas não apareçam e o terceiro (sep = “\t”) delimita o arquivo com tabulações.

A Figura 2-14 mostra como o arquivo de texto fica no Notepad. Ou seja, na primeira linha do arquivo de texto, é preciso pressionar a tecla Tab uma vez para posicionar corretamente os cabeçalhos.



anorexia - Notepad

File Edit Format View Help

	Treat	Prewt	Postwt
1	Cont	80.7	80.2
2	Cont	89.4	80.1
3	Cont	91.8	86.4
4	Cont	74	86.3
5	Cont	78.1	76.1
6	Cont	88.3	78.1
7	Cont	87.3	75.1
8	Cont	75.1	86.7
9	Cont	80.6	73.5
10	Cont	78.4	84.6
11	Cont	77.6	77.4
12	Cont	88.7	79.5
13	Cont	81.3	89.6
14	Cont	78.1	81.4
15	Cont	70.5	81.8
16	Cont	77.3	77.3
17	Cont	85.2	84.2
18	Cont	86	75.4
19	Cont	84.1	79.5
20	Cont	79.7	73
21	Cont	85.5	88.3
22	Cont	84.4	84.7
23	Cont	79.6	81.4
24	Cont	77.5	81.2
25	Cont	72.3	88.2
26	Cont	89	78.8
27	CBT	80.5	82.2
28	CBT	84.9	85.6
29	CBT	81.5	81.4
30	CBT	82.6	81.9

FIGURA 2-14:
O data frame anorexia como arquivo de texto delimitado com tabulações.

LEMBRE-SE

Em cada um desses exemplos utiliza-se o caminho completo para cada arquivo. Isso não será necessário se os arquivos estiverem no diretório de trabalho. Se, por exemplo, colocarmos a planilha Scores no diretório de trabalho, veja o que é preciso fazer para lê-la no R:

```
> read.xlsx("Scores.xlsx","Sheet1")
```

Parte 2
Descrevendo Dados

NESTA PARTE. . .

Resuma e descreva dados.

Trabalhe com gráficos R.

Determine a tendência central e a variabilidade.

Trabalhe com pontuações padrão.

Entenda e visualize as distribuições normais.

Capítulo **3**

NESTE CAPÍTULO

- **Usando gráficos para encontrar padrões**
- **Aprendendo gráficos R básicos**
- **Evoluindo para ggplot2**

Gráficos

A visualização de dados é uma parte importante da Estatística. Um bom gráfico possibilita a localização de tendências e relacionamentos que podem passar despercebidos se observarmos apenas os números. Gráficos também são valiosos, pois o ajudam a apresentar suas ideias a grupos.

Isso é especialmente importante no campo da ciência de dados. As organizações dependem que os cientistas de dados deem sentido a quantidades enormes de dados para que os tomadores de decisão possam formular estratégias. Os gráficos possibilitam que os cientistas de dados expliquem padrões para os gerentes e pessoal não especializado.

Encontrando Padrões

Os dados geralmente residem em tabelas longas e complexas. Com frequência, é preciso visualizar apenas uma parte da tabela para descobrir um padrão ou tendência. Um bom exemplo é o data frame Cars93, que está no pacote MASS. (No Capítulo 2 eu mostro como colocar esse pacote em sua biblioteca R.) Esse data frame contém os dados de 27 variáveis para 93 modelos de carros que estavam disponíveis em 1993.

A Figura 3-1 mostra parte do data frame na janela Data Editor que se abre depois que você digita

```
> edit(Cars93)
```

Data Editor

File Edit Help

	Manufacturer	Model	Type	Min.Price	Price	Max.Price	MPG.city	MPG.highway	AirBags	DriveTrain
1	Acura	Integra	Small	12.9	15.9	18.8	25	31	None	Front
2	Acura	Legend	Midsize	29.2	33.9	38.7	18	25	Driver & Passenger	Front
3	Audi	90	Compact	25.9	29.1	32.3	20	26	Driver only	Front
4	Audi	100	Midsize	30.8	37.7	44.6	19	26	Driver & Passenger	Front
5	BMW	535i	Midsize	23.7	30	36.2	22	30	Driver only	Rear
6	Buick	Century	Midsize	14.2	15.7	17.3	22	31	Driver only	Front
7	Buick	LeSabre	Large	19.9	20.8	21.7	19	28	Driver only	Front
8	Buick	Roadmaster	Large	22.6	23.7	24.9	16	25	Driver only	Rear
9	Buick	Riviera	Midsize	26.3	26.3	26.3	19	27	Driver only	Front
10	Cadillac	DeVille	Large	33	34.7	36.3	16	25	Driver only	Front
11	Cadillac	Seville	Midsize	37.5	40.1	42.7	16	25	Driver & Passenger	Front
12	Chevrolet	Cavalier	Compact	8.5	13.4	18.3	25	36	None	Front
13	Chevrolet	Corsica	Compact	11.4	11.4	11.4	25	34	Driver only	Front
14	Chevrolet	Camaro	Sporty	13.4	15.1	16.8	19	28	Driver & Passenger	Rear
15	Chevrolet	Lumina	Midsize	13.4	15.9	18.4	21	29	None	Front
16	Chevrolet	Lumina_APV	Van	14.7	16.3	18	18	23	None	Front
17	Chevrolet	Astro	Van	14.7	16.6	18.6	15	20	None	4WD
18	Chevrolet	Caprice	Large	18	18.8	19.6	17	26	Driver only	Rear
19	Chevrolet	Corvette	Sporty	34.6	38	41.5	17	25	Driver only	Rear

FIGURA 3-1:
Parte do data frame Cars93.

Colocando uma distribuição no gráfico

Um padrão interessante é a distribuição de preços de todos os carros listados no data frame Cars93. Seria uma tarefa tediosa ter que examinar todo o data frame para determinar isso. Um gráfico, no entanto, fornece a informação imediatamente. O *histograma* da Figura 3-2 mostra o que quero dizer com isso.

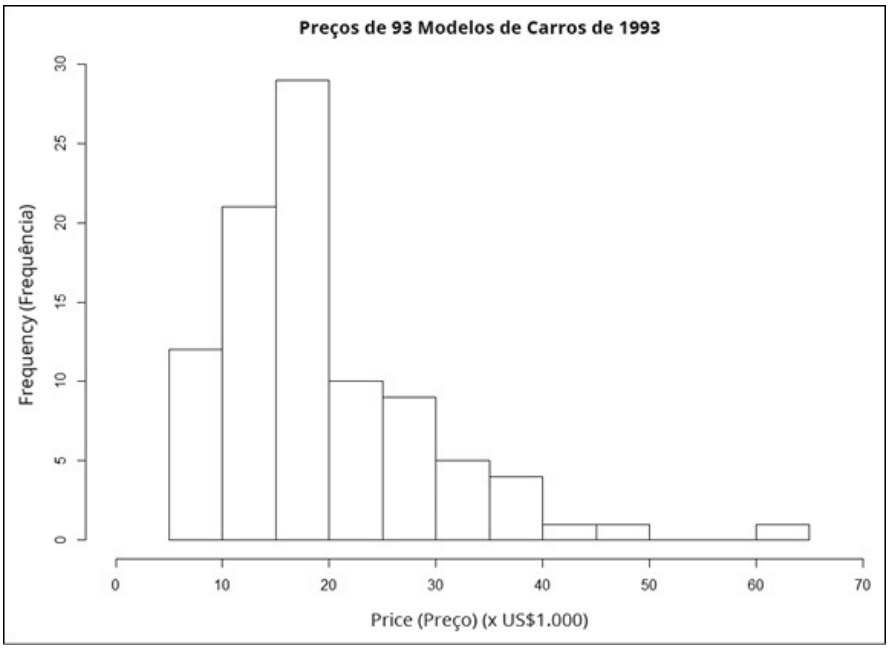


FIGURA 3-2:
Histograma dos preços de carros no data frame Cars93.

O histograma é adequado quando a variável no eixo x é uma variável intervalar ou de razão. (Veja o Capítulo 1.) Com esses tipos de variáveis, os números têm significado.

No Capítulo 1, faço a distinção entre variáveis independentes e dependentes. Aqui, Price (Preço) é a variável independente e Frequency (Frequência) é a

variável dependente. Na maioria dos gráficos (mas não em todos), a variável independente fica no eixo x, e a variável dependente, no eixo y.

Pulando as barras

Para as variáveis nominais (de novo, veja o Capítulo 1), os números são apenas rótulos. Na verdade, os níveis de uma variável nominal (também chamada de *fator*; veja o Capítulo 2) podem ser nomes. No caso em questão, outro possível ponto de interesse são as frequências dos diferentes tipos de carros (esportivo, médio, van e assim por diante) no data frame. Então, "Type" (Tipo) é uma variável nominal. Se observássemos todas as entradas do data frame e criássemos uma tabela com essas frequências, ela se pareceria com a Tabela 3-1.

TABELA 3-1 Tipos e Frequências de Carros no data frame Cars93

Type (Tipo)	Frequency (Frequência)
Compacto	16
Grande	11
Médio	22
Pequeno	21
Esportivo	14
Van	9

A tabela mostra algumas tendências — mais modelos de carros médios e pequenos do que grandes e vans. Os carros compactos e esportivos estão no meio.

A Figura 3-3 mostra essas informações em forma de gráfico, que é do tipo *barra*. Os espaços entre as barras enfatizam que Type, no eixo x, é uma variável nominal.

Embora a tabela seja bem direta, acho que concordamos que um público preferiria ver uma imagem. E eu gosto de dizer que os olhos que brilham quando veem números costumam brilhar mais quando veem imagens.

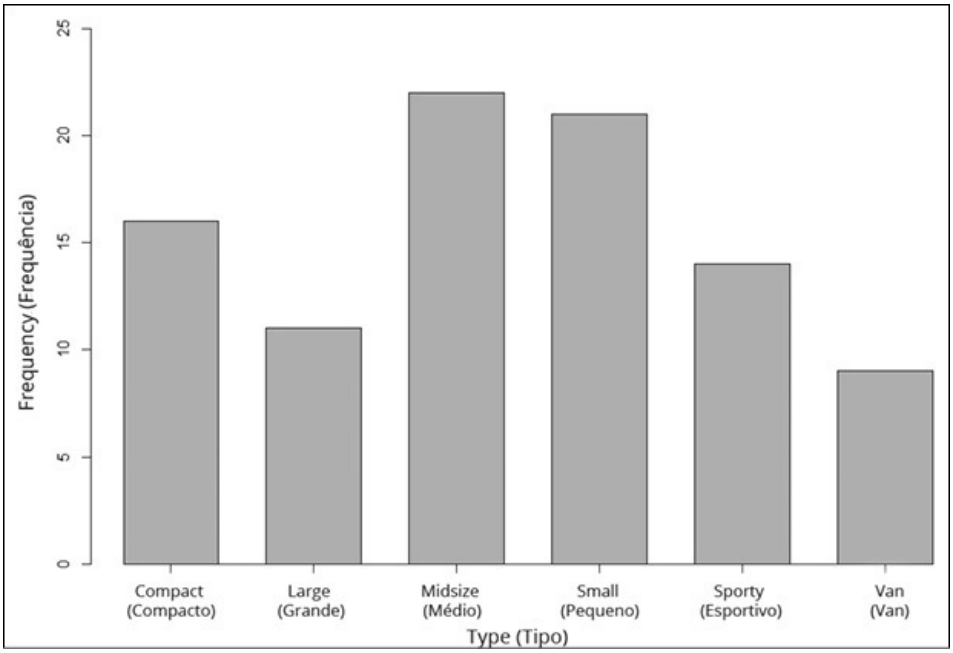


FIGURA 3-3:
Tabela 3-1 em forma de gráfico de barras.

Fatiando a pizza

O *gráfico de pizza* é outro tipo de imagem que mostra os mesmos dados de maneira um pouco diferente. Cada frequência aparece como uma fatia de pizza. A Figura 3-4 ilustra o que quero dizer. Em um gráfico de pizza, a área da fatia representa a frequência.

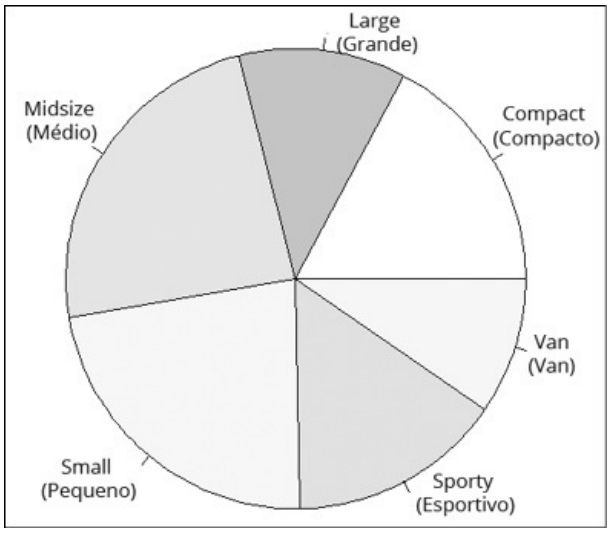


FIGURA 3-4:
Tabela 3-1em forma de gráfico de pizza.

REGRAS GERAIS DO GRÁFICO DE PIZZA

Desculpe-me se você já ouviu isso antes. É uma história bonitinha que serve

como regra prática para os gráficos de pizza.

O grande, e já falecido, Yogi Berra fazia, com frequência, declarações errôneas e adoráveis que se tornaram parte de nossa cultura. Uma vez ele supostamente entrou em uma pizzaria e pediu uma pizza.

"Devo cortá-la em quatro ou oito fatias?", perguntou a garçonete.

"Melhor cortar em quatro", disse Yogi. "Não estou com fome suficiente para comer oito."

Moral da história: Se um fator tem vários níveis que resultam em um gráfico de pizza com muitas fatias, isso provavelmente é uma sobrecarga de informações. A mensagem seria mais bem transmitida com um gráfico de barras.

(Esse incidente de Yogi realmente aconteceu? Não se sabe. Resumindo uma vida de citações atribuídas a ele, o Sr. Berra disse: "Metade das mentiras que contam sobre mim não são verdadeiras.")

Plano da dispersão

Outro padrão de interesse potencial é o relacionamento entre MPG (milhas por galão ou quilômetros por litro no Brasil) para dirigir na cidade e potência. Esse tipo de gráfico é um *diagrama de dispersão*. A Figura 3-5 mostra o diagrama de dispersão para essas duas variáveis.

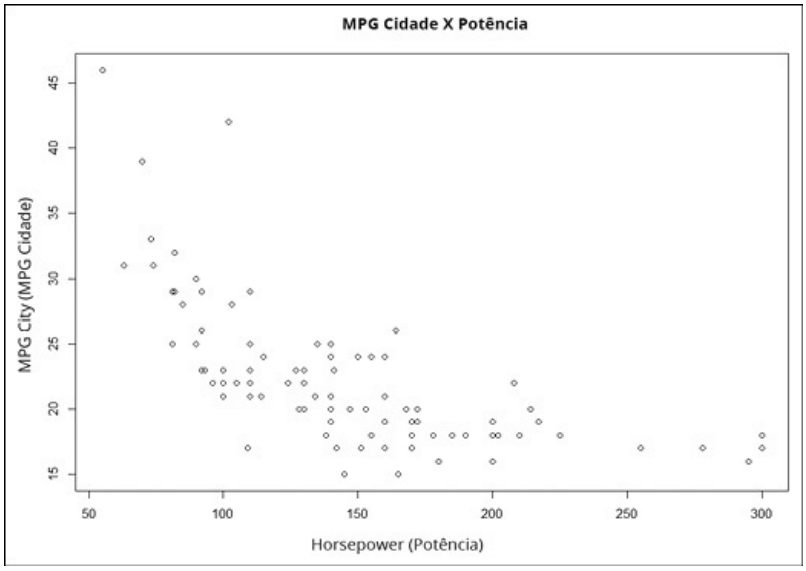


FIGURA 3-5:
MPG (km/l)ao dirigir na cidade e a potência para os dados em Cars93.

Cada pequeno círculo representa um dos 93 carros. A posição de um círculo no eixo x (sua *coordenada x*) é sua potência e sua posição no eixo y (sua *coordenada y*) é a MPG (km/l) para dirigir na cidade.

Uma olhada rápida na forma do diagrama de dispersão sugere um relacionamento: à medida que a potência aumenta, MPG-cidade parece diminuir. (Estatísticos diriam que "a MPG-cidade diminui com a potência".) É possível usar a estatística para analisar esse relacionamento e, talvez, fazer previsões? Com certeza! (Veja o Capítulo 14.)

De caixas e bigode

E o relacionamento entre potência e o número de cilindros no motor de um carro? É esperado que a potência aumente com os cilindros e a Figura 3-6 mostra que esse é realmente o caso. Inventado pelo famoso estatístico John Tukey, esse tipo de gráfico é chamado de *diagrama de caixa*, e é uma maneira boa e rápida de visualizar os dados.

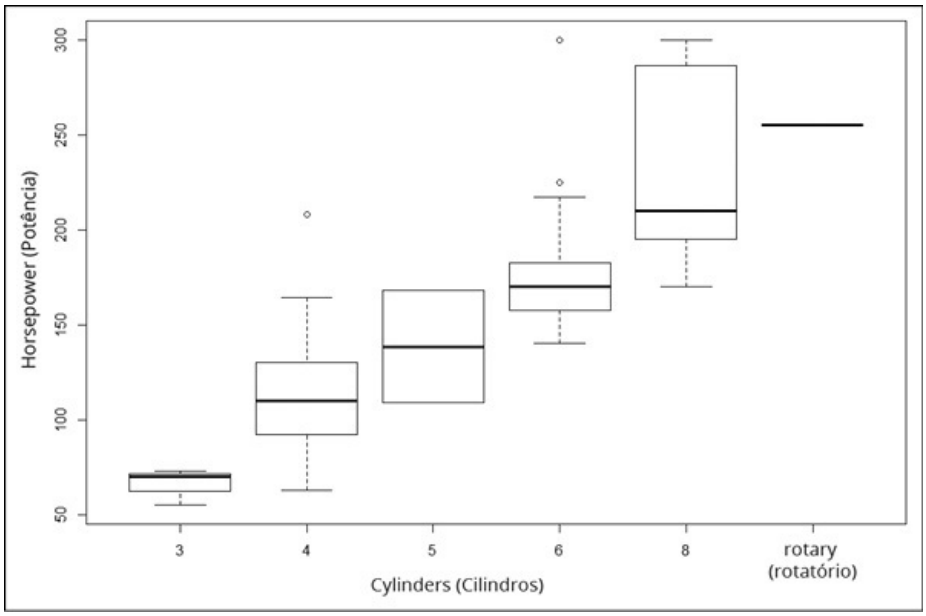


FIGURA 3-6:
Diagrama de caixa da potência X número de cilindros no data frame Cars93.

Cada caixa representa um grupo de números. A da extrema esquerda, por exemplo, representa a potência de carros com três cilindros. A linha preta sólida dentro da caixa é a *mediana* ou o valor da potência que fica entre os números inferiores e superiores. As extremidades inferior e superior da caixa são chamadas de *quartis*. O mais baixo é o *quartil inferior,* o número abaixo do qual ficam 25% dos números. O mais alto é o quartil *superior*, o número que excede 75% dos números. (Falo sobre medianas no Capítulo 4 e percentis no Capítulo 6.)

Os elementos que saem dos quartis são chamados de *bigodes* (é possível ver esses gráficos sendo chamados de diagrama de *caixa e bigode*). Os bigodes incluem valores de dados que estão fora dos quartis. O bigode do limite superior pode ser o valor máximo ou o quartil superior mais 1,5 vezes o comprimento da caixa, o *menor* entre eles. O bigode do limite inferior pode ser o valor mínimo ou o quartil inferior menos 1,5 vezes o comprimento da caixa, o *maior* entre eles. Os pontos de dados fora dos bigodes são *valores discrepantes*. O diagrama de caixa mostra que os dados para quatro e para seis cilindros têm valores discrepantes.

Note que o gráfico mostra apenas uma linha sólida para o "rotatório", um tipo de motor que ocorre apenas uma vez nos dados.

Gráficos de Base R

A capacidade de criar gráficos, como mostrada nas seções anteriores, vem com sua instalação de R, que torna esses gráficos parte dos gráficos de base R. Começarei por eles. Depois, na seção seguinte, mostrarei o utilíssimo pacote ggplot2.

Na base R, o formato geral para criar gráficos é

```
graphics_function(data, arg1, arg2, ...)
```

Depois de criar um gráfico no RStudio, clique em Zoom na aba Plots do RStudio para abrir o gráfico em uma janela maior. Ele é mais claro na janela Zoom do que na aba Plots.

Histogramas

Hora de dar outra olhada no data frame Cars93 que introduzi na seção "Encontrando Padrões", anteriormente neste capítulo. Para criar um histograma da distribuição de preços nesse data frame, digite

```
> hist(Cars93$Price)
```

que produz a Figura 3-7.

Você notará que ele não é tão elegante quanto o da Figura 3-2. Como é possível ajeitá-lo? Adicionando argumentos.

Um argumento muito usado nos gráficos de base R muda o rótulo do eixo x do padrão de R para algo mais significativo. Ele se chama xlab. Para o eixo x na Figura 3-2, eu adicionei

```
xlab= “Price (x $1,000)”
```

aos argumentos. É possível usar ylab para mudar o rótulo do eixo y, mas não fiz isso aqui.

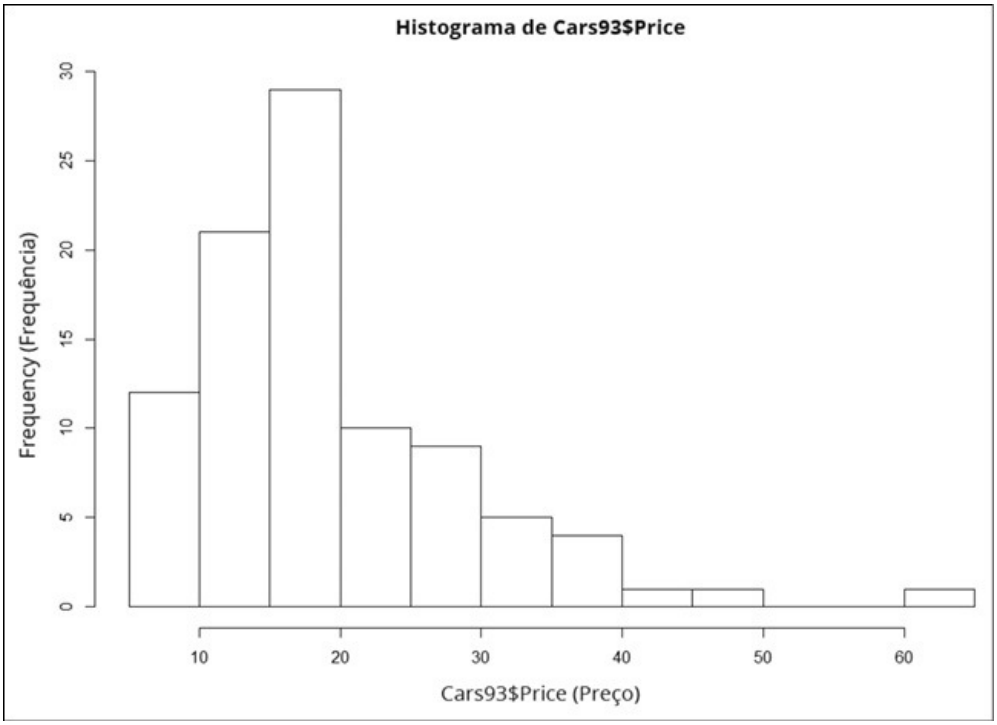


FIGURA 3-7:
Histograma inicial da distribuição de preços em Cars93.

Eu queria que o eixo x aumentasse de um limite inferior 0 para um limite superior 70, e essa é a área do argumento xlim. Como esse argumento trabalha com um vetor, adicionei

```
xlim = c(0,70)
```

Eu também quis um título diferente, e para isso usei main:

```
main = “Prices of 93 Models of 1993 Cars”
```

Para produzir o histograma da Figura 3-2, a história toda é

```
> hist(Cars93$Price, xlab="Price (x $1,000)", xlim = c(0,70), main =
  "Prices of 93 Models of 1993 Cars")
```

DICA

Ao criar um histograma, R calcula qual é o melhor número de colunas para uma boa aparência. Aqui, R decidiu que 12 é um bom número. É possível variar o número de colunas adicionando um argumento chamado breaks e declarando seu valor. R nem sempre lhe dá o valor a ser atribuído, mas fornece algo próximo e tenta manter uma boa aparência. Adicione esse argumento, declare seu valor (breaks =4, por exemplo) e entenderá o que quero dizer.

Adicionando recursos gráficos

Um aspecto importante dos gráficos de base R é a habilidade de adicionar recursos a um gráfico depois de criá-lo. Para mostrar o que quero dizer, preciso começar com um tipo de gráfico levemente diferente.

Outra maneira de mostrar as informações do histograma é pensar nos dados como *probabilidades*, em vez de frequências. Então, em vez da frequência de uma faixa de preços específica, colocamos no gráfico a probabilidade de que um carro daquela faixa seja selecionado nos dados. Para isso, adicione

```
probability = True
```

aos argumentos. Agora o código R se parece com o seguinte:

```
> hist(Cars93$Price, xlab="Price (x $1,000)", xlim = c(0,70), main =
  "Prices of 93 Models of 1993 Cars", probability= TRUE)
```

O resultado está na Figura 3-8. O eixo y mede a *Density* (Densidade), um conceito relacionado à probabilidade que analisamos no Capítulo 8. O gráfico é chamado de *gráfico de densidade*.

O propósito de tudo isso é o que vem a seguir. Depois de criar o gráfico, é possível usar uma função adicional chamada lines() para adicionar uma linha ao gráfico de densidade:

```
> lines(density(Cars93$Price))
```

O gráfico agora se parece com o da Figura 3-9.

Então, nos gráficos de base R, é possível criar um gráfico e adicionar coisas a ele depois de ver o gráfico inicial. É parecido com pintar um quadro de um lago e depois adicionar montanhas e árvores como achar melhor.

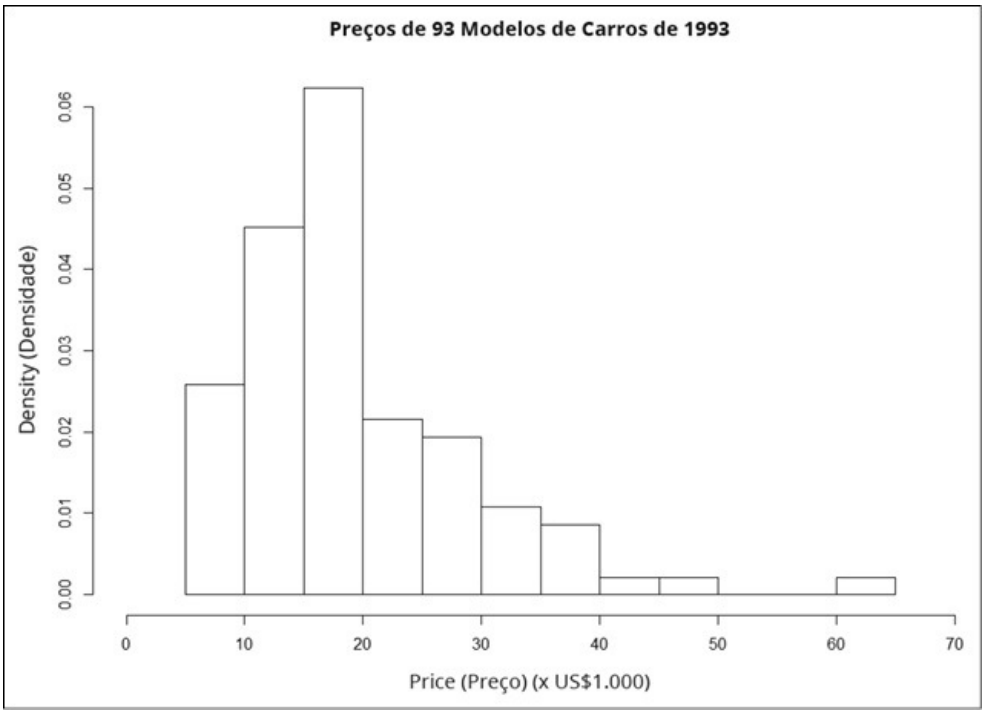


FIGURA 3-8:

Gráfico de densidade da distribuição de preços em Cars93.

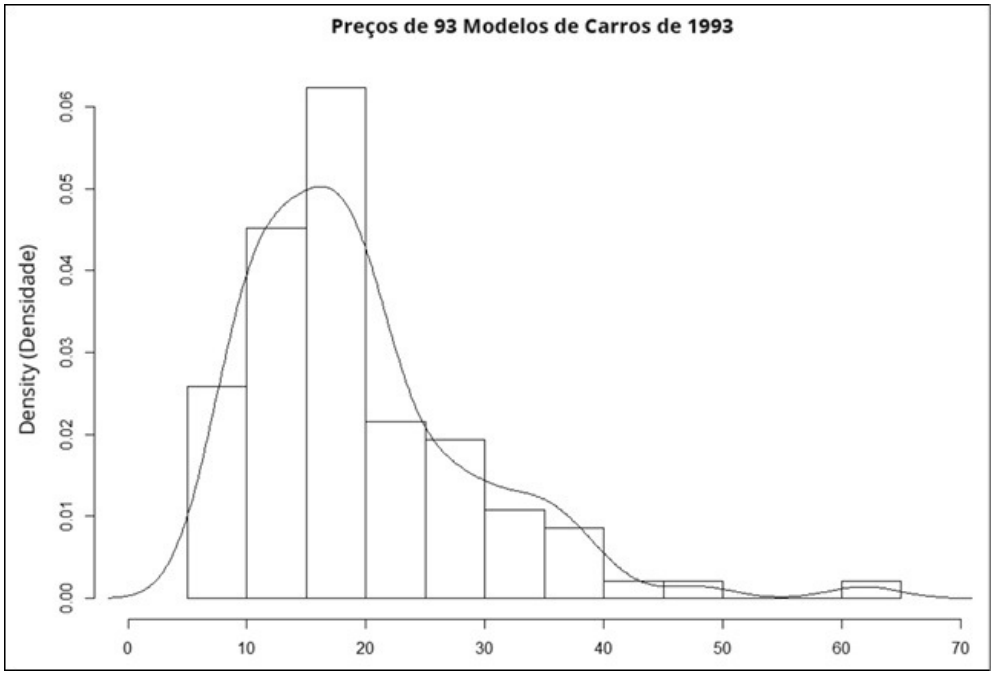


FIGURA 3-9:
Gráfico de densidade com uma linha adicionada.

Gráficos de barras

Anteriormente, na seção "Encontrando Padrões", mostrei um gráfico de barras ilustrando os tipos e frequências de carros, além da Tabela 3-1. Como se vê, é preciso fazer esse tipo de tabela antes de poder usar barplot() para criar um gráfico de barras.

Para criar a Tabela 3-1, o código R é (muito adequadamente)

```
> table(Cars93$Type)
Compact Large Midsize Small Sporty Van
16 11 22 21 14 9
```

Para o gráfico de barras, então, ele é

```
> barplot(table(Cars93$Type))
```

o que cria o gráfico na Figura 3-10.

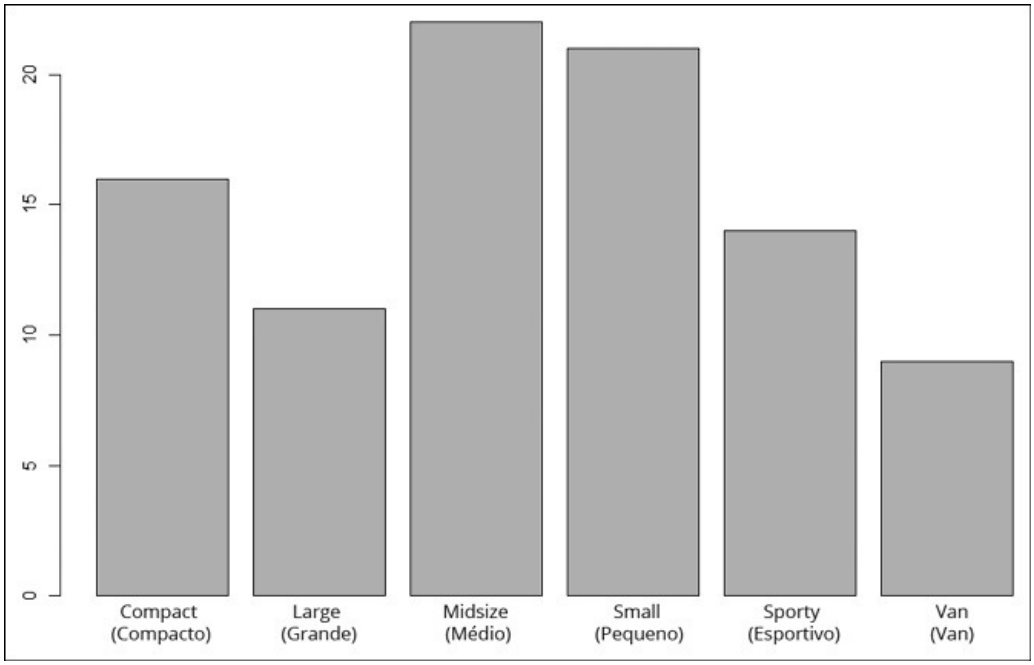


FIGURA 3-10:
Gráfico de barra inicial de table (Cars93 $Type).

Novamente, não é tão chamativo quanto o produto exibido na Figura 3-3. Os argumentos adicionais cuidam disso. Para colocar de 0 a 25 no eixo y, use ylim, que, como xlim, funciona com um vetor:

```
ylim = c(0,25)
```

Para os rótulos dos eixos x e y, use

```
xlab = “Type”
ylab = “Frequency”
```

Para desenhar um eixo sólido, trabalhe com axis.lty. Pense nisso como um "tipo de linha do eixo" que você define para solid escrevendo

```
axis.lty = “solid”
```

Os valores dashed e dotted de axis.lty resultam em aparências diferentes do eixo x.

Finalmente, use space para aumentar o espaçamento entre as barras:

```
space = .05
```

Veja a função completa para produzir o gráfico na Figura 3-3:

```
> barplot(table(Cars93$Type),ylim=c(0,25), xlab="Type",
  ylab="Frequency", axis.lty = "solid", space = .05)
```

Gráficos de pizza

Este tipo de gráfico não poderia ser mais fácil. A linha

```
> pie(table(Cars93$Type))
```

o leva diretamente para a Figura 3-4.

Gráficos de pontos

Espera. O quê? De onde veio isso? Essa é mais uma maneira de visualizar os dados da Tabela 3-1. O chefão da notação de gráficos William Cleveland acredita que as pessoas percebem valores ao lado de uma escala comum (como um gráfico de barras) melhor do que percebem áreas (como em um gráfico de pizza). Então ele inventou o *gráfico de pontos*, exibido na Figura 3-11.

Ele se parece um pouco com um ábaco deitado de lado, não parece? É um daqueles casos raros em que a variável independente está no eixo y e a dependente está no eixo x.

O formato para a função que cria o gráfico de pontos é

```
> dotchart(x, labels, arg1, arg2 ...)
```

Os dois primeiros argumentos são vetores e os outros são argumentos opcionais para modificar a aparência do gráfico de pontos. O primeiro vetor é o de valores (as frequências). O segundo é autoexplicativo; nesse caso, são rótulos para os tipos de veículos.

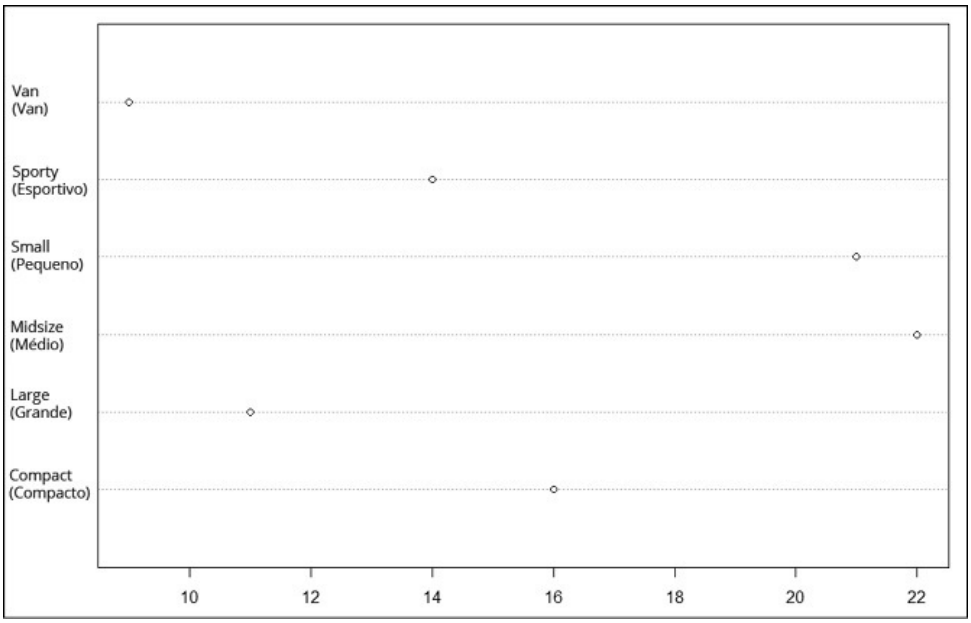


FIGURA 3-11:
Gráfico de pontos para os dados da Tabela 3-1.

Para criar os vetores necessários, transforme a tabela em um data frame:

```
> type.frame <- data.frame(table(Cars93$Type))
> type.frame
Var1 Freq
1 Compact 16
2 Large 11
3 Midsize 22
4 Small 21
```

```
5 Sporty 14
6 Van 9
```

Depois use esta linha para produzir o gráfico de pontos:

```
> dotchart(type.frame$Freq,type.frame$Var1)
```

O type.frame$Freq especifica que a coluna Frequency (Frequência) no data frame é o eixo x e type.frame$Var1 especifica que a coluna Var1 (que contém os tipos de carros) é o eixo y.

Esta linha também funciona:

```
> dotchart(type.frame[,2],type.frame[,1])
```

Lembre-se, do Capítulo 2, que [,2] significa "coluna 2" e [,1] significa "coluna 1".

Revendo gráficos de barra

Em todos os gráficos anteriores, a variável dependente foi a frequência. Contudo, muitas vezes, a variável dependente é um ponto de dados em vez de uma frequência. Veja o que quero dizer.

A Tabela 3-2 mostra os dados para as receitas do espaço comercial no início da década de 1990. (Aliás, os dados são do Departamento de Comércio dos EUA, via Statistical Abstract of the U.S. ["Resumo Estatístico dos Estados Unidos", em tradução livre].)

TABELA 3-2 Receitas do Espaço Comercial nos EUA 1990–1994 (Em Milhões de Dólares)

Setor	1990	1991	1992	1993	1994
Commercial Satellites Delivered (Satélites Comerciais Entregues)	1.000	1.300	1.300	1.100	1.400
Satellite Services (Serviços por Satélite)	800	1.200	1.500	1.850	2.330
Satellite Grund Equipment (Equipamento Terrestre por Satélite)	860	1.300	1.400	1.600	1.970
Commercial Launches (Lançamentos Comerciais)	570	380	450	465	580
Remote Sensing Data (Dados de Detecção Remota)	155	190	210	250	300

Os dados são os números nas células, que representam a receita em milhares de dólares. Um diagrama de base R dos dados nessa tabela aparece na Figura 3-12.

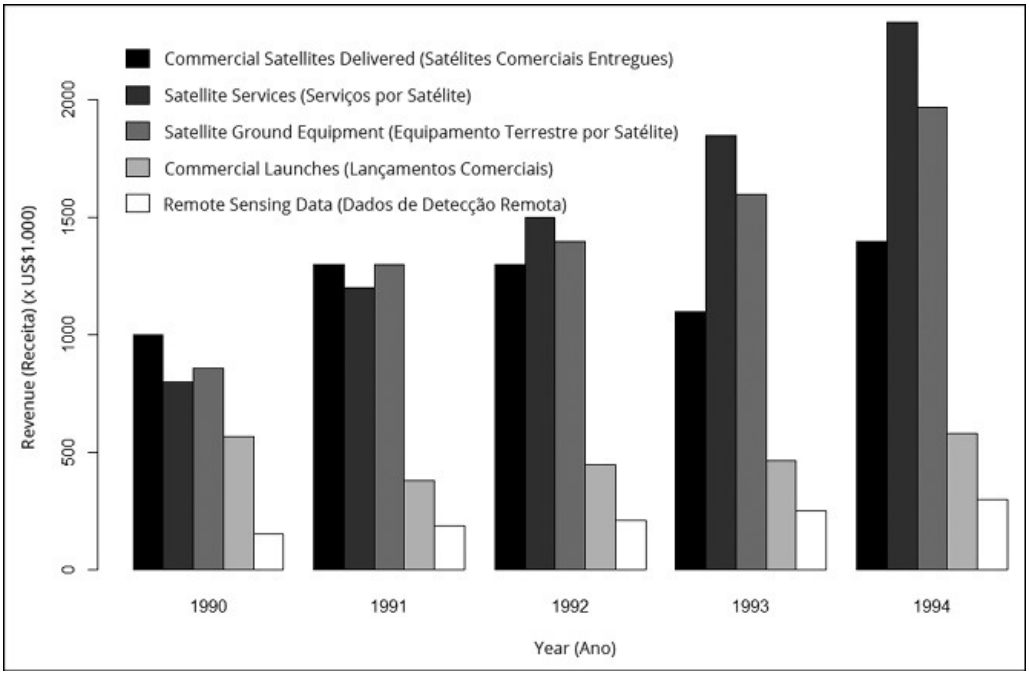


FIGURA 3-12:
Gráfico de barras dos dados da Tabela 3-2.

Se tivéssemos que fazer uma apresentação desses dados, acho que você concordaria que nosso público preferiria o gráfico à tabela. Embora a tabela seja informativa, ela não chama a atenção. É mais fácil ver tendências no gráfico — Satellite Services cresceu mais rápido, enquanto Commercial Launches permaneceu bem nivelado, por exemplo.

Esse gráfico é chamado de *gráfico de barras agrupadas*. Como um gráfico desse tipo é criado na base R?

A primeira coisa a fazer é criar um vetor de valores nas células:

```
rev.values <- c(1000,1300,1300,1100,1400,800,1200,1500,1850,
  2330,860,1300,1400,1600,1970,570,380,450,465,580,
  155,190,210,250,300)
```

CUIDADO

Embora os pontos apareçam nos valores da tabela (para os valores maiores que mil), não se pode ter pontos nos valores do vetor! (E por uma razão óbvia, também não se pode ter vírgulas, pois elas são usadas para separar os valores consecutivos no vetor.)

Em seguida, transforme esse vetor em uma matriz. É preciso informar a R quantas linhas (ou colunas) a matriz terá e quais valores carregar na matriz linha a linha:

```
space.rev <- matrix(rev.values,nrow=5,byrow = T)
```

Por fim, forneça os nomes das colunas e das linhas à matriz:

```
colnames(space.rev) <- c("1990","1991","1992","1993","1994")
rownames(space.rev) <- c("Commercial Satellites
  Delivered","Satellite Services","Satellite Ground
  Equipment","Commercial Launches","Remote Sensing Data")
```

Vejamos a matriz:

```
> space.rev
1990 1991 1992 1993 1994
Commercial Satellites Delivered 1000 1300 1300 1100 1400
Satellite Services 800 1200 1500 1850 2330
Satellite Ground Equipment 860 1300 1400 1600 1970
Commercial Launches 570 380 450 465 580
Remote Sensing Data 155 190 210 250 300
```

Perfeito. Está igualzinha à Tabela 3-2.

Com os dados em mãos, prossiga para o gráfico de barras. Crie um vetor de cores para as barras:

```
color.names = c("black","grey25","grey50","grey75","white")
```

DICA

Sobre os nomes das cores: é possível colocar qualquer número de 0 a 100 ao lado de "grey" e obter uma cor: "grey0" é equivalente a "black" e "grey100" é equivalente a "white". (Muito mais de 50 tons, se você entende o que eu quero dizer...)

E agora, o gráfico:

```
> barplot(space.rev, beside = T, xlab= "Year",ylab= "Revenue(X
  $1,000)", col=color.names)
```

beside = T significa que as barras ficarão lado a lado. (Você deve experimentar isso sem o argumento e ver o que acontece.) O argumento col = color.names fornece as cores especificadas no vetor.

O diagrama resultante é exibido na Figura 3-13.

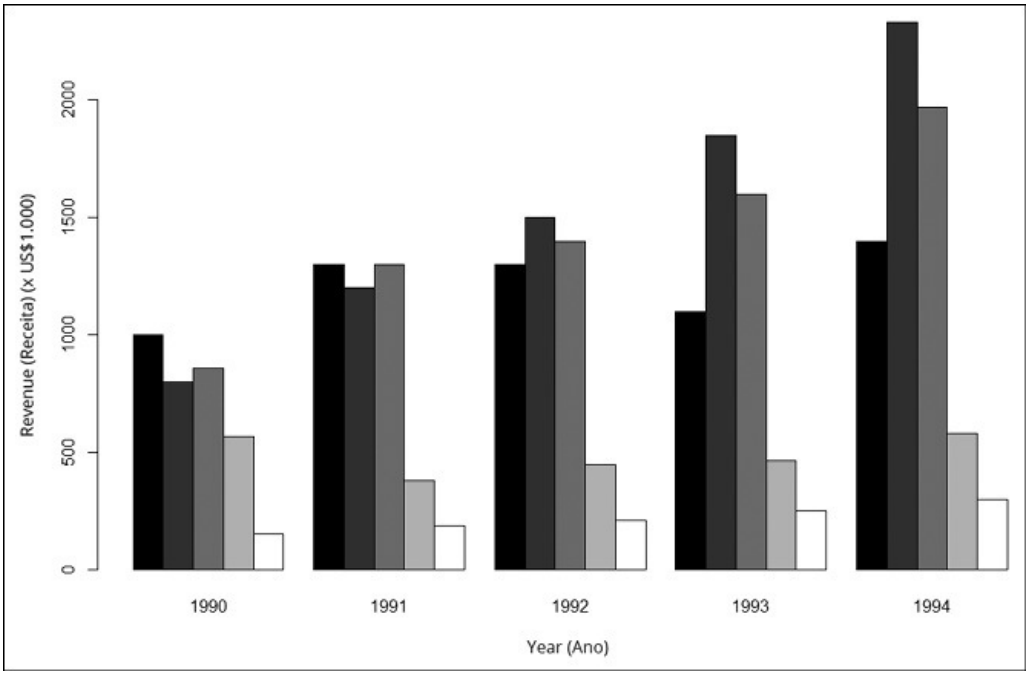


FIGURA 3-13:
Gráfico de barras inicial dos dados na Tabela 3-2.

O que ainda falta é a legenda, claro. Adicione-a com a função legend() para produzir a Figura 3-12:

```
> legend(1,2300,rownames(space.rev), cex=0.7, fill = color.names,
  bty = "n")
```

Os dois primeiros valores são as coordenadas x e y para localizar a legenda. (Isso exigiu *muitos* ajustes!). O argumento seguinte mostra o que entra na legenda (os nomes dos setores). O argumento cex especifica o tamanho dos caracteres da legenda. O valor, 0.7, indica que os caracteres têm 70% do tamanho que teriam normalmente. Essa é a única maneira de a legenda caber no gráfico. (Pense em "cex" como "expansão de caracteres", embora, nesse caso, seja "contração de caracteres".) fill = color.names coloca as amostras de cor na legenda, ao lado dos nomes das linhas. Definir bty (o "tipo de borda") para "n" ("none") é outro truque para que a legenda caiba no gráfico.

Diagramas de dispersão

Para visualizar o relacionamento entre potência e MPG (km/l) para dirigir na cidade (como na Figura 3-5), use a função plot():

```
> plot(Cars93$Horsepower, Cars93$MPG.city,
  xlab="Horsepower",ylab="MPG City", main ="MPG City vs
  Horsepower")
```

Como você pode ver, adicionei os argumentos para rotular os eixos e o título.

Outra maneira de fazer isso é usar a notação da fórmula que mostro no Capítulo 2. Então, se quiser que o código R mostre que MPG-city depende da potência, digite

```
> plot(Cars93$MPG.city ~ Cars93$Horsepower,
xlab="Horsepower",ylab="MPG City", main ="MPG City vs
Horsepower")
```

para produzir o mesmo diagrama de dispersão.

LEMBRE-SE

O operador til (~) significa "depende de".

Uma reviravolta

R possibilita que você mude o símbolo que representa os pontos no gráfico. A Figura 3-5 mostra que o símbolo padrão é um círculo vazio. Para mudar o símbolo, chamado *caractere gráfico*, configure o argumento pch. R tem um conjunto de valores numéricos (0–25) incorporados para pch que correspondem a um conjunto de símbolos. Os valores de 0 a 15 correspondem a formas ocas e 16 a 25 são formas preenchidas.

O valor padrão é 1. Para mudar o caractere gráfico para quadrados, defina pch para 0. Para triângulos, é 2, e para círculos preenchidos, é 16:

```
> plot(Cars93$Horsepower,Cars93$MPG.city, xlab= "Horsepower",
ylab="MPG City", main = "MPG City vs Horsepower",pch=16)
```

A Figura 3-14 mostra o gráfico com os círculos preenchidos.

Também é possível configurar o argumento col para mudar a cor de "black" (preto) para "blue" (azul) ou várias outras cores (que não ficariam bem em uma página em preto e branco como a que você está vendo agora).

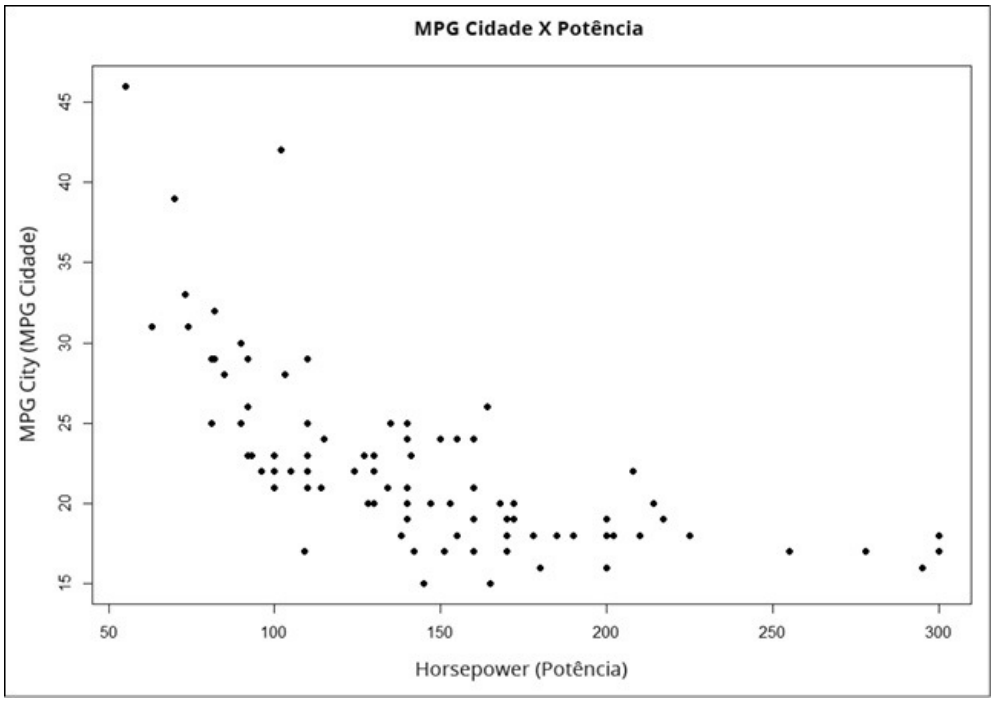


FIGURA 3-14:
MPG na Cidade X Potência com círculos preenchidos (pch = 16).

Não estamos limitados aos valores numéricos incorporados para pch. Aqui, por exemplo, há um toque interessante: para ajudar a encontrar padrões nos dados, é possível desenhar cada ponto no gráfico como o número de cilindros no carro correspondente, em vez de um símbolo.

Para isso, tome cuidado ao configurar pch. Você não pode simplesmente atribuir Cars93$.Cylinders como o valor. É preciso garantir que o que for passado a pch seja um caractere (como "3", "4" ou "8"), em vez de um número (como 3, 4 ou 8). Outra complicação é que os dados contêm "rotatório" como um valor de Cylinders. Para forçar que o valor dos cilindros seja um caractere, aplique as.character() a Cars93$Cylinders:

```
pch = as.character(Cars93$Cylinders)
```

E a função plot() é

```
> plot(Cars93$Horsepower,Cars93$MPG.city, xlab=
  "Horsepower",ylab="MPG City", main = "MPG City vs
  Horsepower", pch = as.character (Cars93$Cylinders))
```

O resultado é o diagrama de dispersão na Figura 3-15. Curiosamente, as.character() passa "rotatório" como "r".

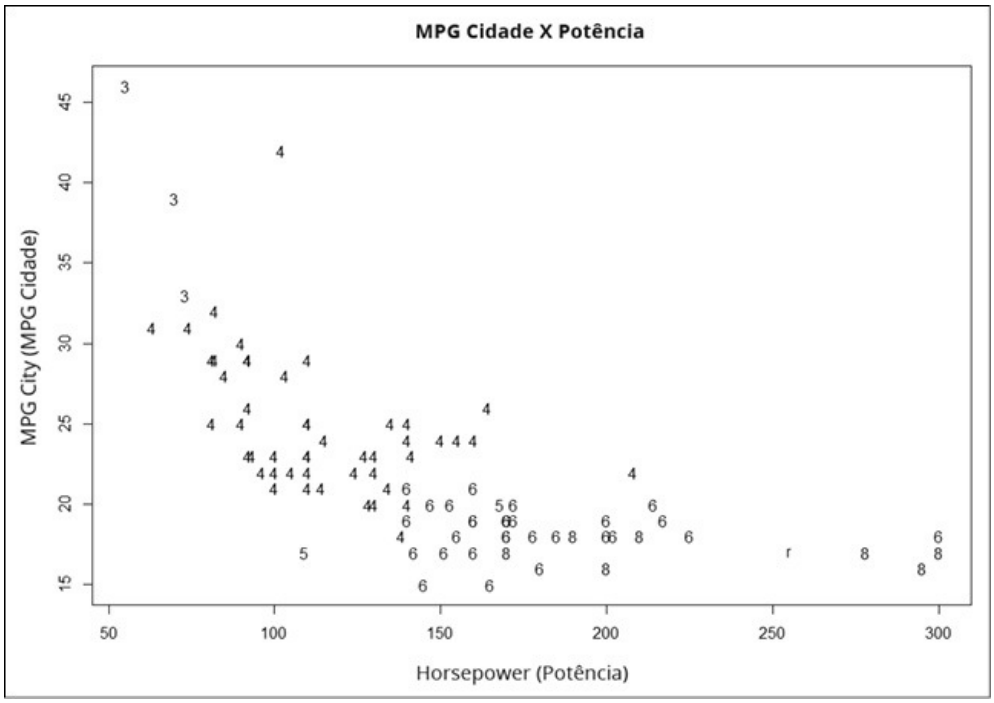


FIGURA 3-15:
MPG Cidade X Potência com pontos diagramados como número de cilindros.

Alinhado à nossa intuição sobre carros, esse diagrama mostra claramente que números baixos de cilindros se associam a potências mais baixas e quilometragens de combustível mais altas, e números mais altos de cilindros

se associam a potências mais altas e quilometragens de combustível mais baixas. Também é possível ver rapidamente onde o motor rotatório se encaixa nisso tudo (baixa quilometragem de combustível, potência alta).

Matriz do diagrama de dispersão

A base R fornece uma bela maneira de visualizar relacionamentos entre mais de duas variáveis. Se adicionarmos o preço na mistura e quisermos mostrar todos os relacionamentos emparelhados entre MPG-city, preço e potência, precisaremos de vários diagramas de dispersão. R pode reuni-los em uma matriz, como mostra a Figura 3-16.

Os nomes das variáveis estão nas células da diagonal principal. Cada célula fora da diagonal mostra o diagrama de dispersão para sua própria variável de linha (no eixo y) e sua variável de coluna (no eixo x). Por exemplo, o diagrama de dispersão na primeira linha, segunda coluna, mostra MPG-city no eixo y e o preço no eixo x. Na segunda linha, primeira coluna, os eixos estão trocados: MPG-city no eixo x e o preço no eixo y.

A função R para diagramar essa matriz é pairs(). Para calcular as coordenadas de todos os diagramas de dispersão, essa função trabalha com colunas numéricas e uma matriz ou data frame.

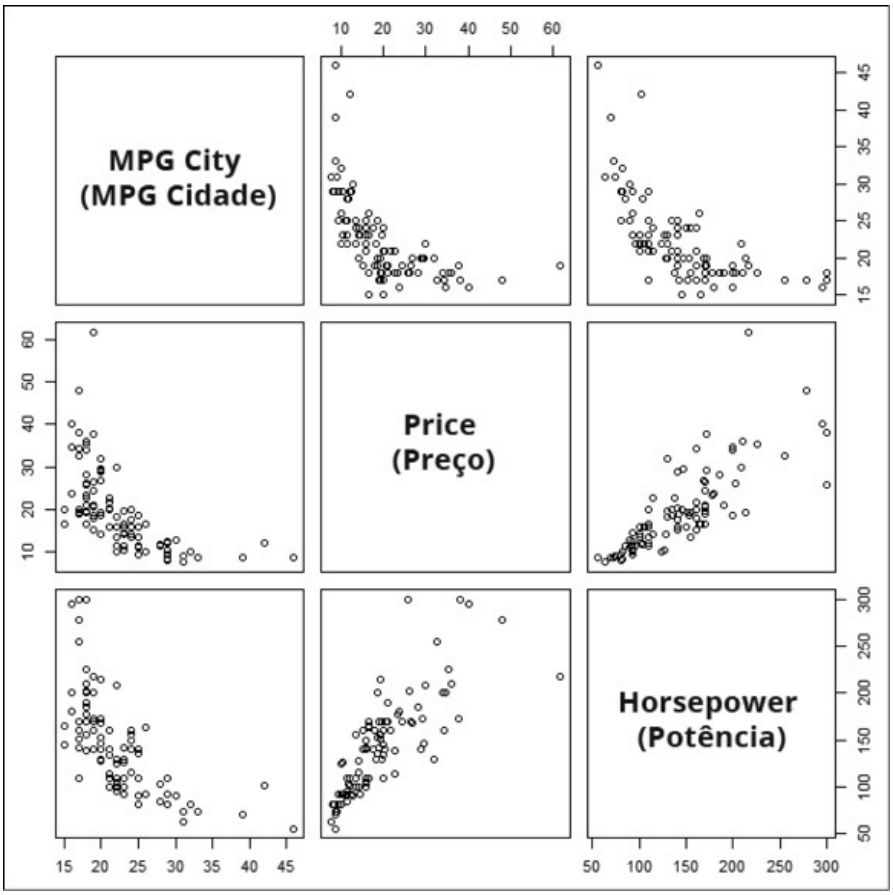


FIGURA 3-16:
Vários diagramas de dispersão para os relacionamentos entre MPG-city, preço e potência.

Por conveniência, crie um data frame que seja um subconjunto do data frame Cars93. Esse novo data frame consiste em apenas três variáveis para diagramar. A função subset() lida com isso facilmente:

```
> cars.subset <- subset(Cars93, select = c(MPG.
  city,Price,Horsepower))
```

O segundo argumento de subset cria um vetor do que exatamente é preciso retirar de Cars93. Apenas para garantir que o novo data frame esteja como queremos, use a função head() para dar uma olhada nas primeiras seis linhas:

```
> head(cars.subset)
MPG.city Price Horsepower
1 25 15.9 140
2 18 33.9 200
3 20 29.1 172
4 19 37.7 172
5 22 30.0 208
6 22 15.7 110
```

E agora,

```
> pairs(cars.subset)
```

cria o diagrama da Figura 3-16.

Essa capacidade não é limitada a três variáveis, nem a variáveis contínuas. Para ver o que acontece com um tipo diferente de variável, adicione Cylinders ao vetor de select e use a função pairs()em cars.subset.

Diagramas de caixa

Para desenhar um diagrama de caixa como o exibido anteriormente na Figura 3-6, use uma fórmula para mostrar que Horsepower é a variável dependente e Cylinders é a variável independente:

```
> boxplot(Cars93$Horsepower ~ Cars93$Cylinders,
  xlab="Cylinders", ylab="Horsepower")
```

Veja outra maneira de fazer isso se cansar de digitar $:

```
> boxplot(Horsepower ~ Cylinders, data = Cars93, xlab="Cylinders",
  ylab="Horsepower")
```

Com os argumentos definidos como em qualquer um dos dois exemplos anteriores de código, plot() funciona exatamente como boxplot().

Evoluindo para ggplot2

O conjunto de ferramentas gráficas de base R é o começo, mas se você quiser realmente brilhar nas visualizações, será uma boa ideia aprender o ggplot2. Criado pela megaestrela de R, Hadley Wickham, o "gg" no nome do pacote é relativo à "gramática dos gráficos" e é um bom indicador do que vem a seguir. Esse também é o título do livro (de Leland Wilkinson) que é a fonte dos conceitos desse pacote.

Em geral, uma *gramática* é um conjunto de regras para combinar coisas. Na gramática que estamos mais familiarizados, essas coisas são palavras, frases e orações: a gramática de nossa língua informa como combinar esses componentes para produzir frases válidas.

Então uma "gramática dos gráficos" é um conjunto de regras para combinar componentes gráficos e produzir gráficos. Wilkinson propôs que todos os gráficos têm componentes subjacentes em comum, como dados, um sistema de coordenadas (por exemplo, os eixos x e y que conhecemos bem), transformações estatísticas (como contagens de frequência) e objetos dentro do gráfico (por exemplo, pontos, barras, linhas ou fatias de pizza), para citar alguns.

Assim como combinar palavras e frases produz orações gramaticais, combinar componentes gráficos produz gráficos. E assim como algumas frases são gramaticais, mas não fazem sentido ("Ideias verdes sem cor dormem furiosamente."), algumas criações ggplot2 são lindos gráficos que nem sempre são úteis. Fica a cargo de quem fala/escreve passar o sentido ao público, e fica a cargo do desenvolvedor gráfico criar gráficos úteis para que as pessoas utilizem.

Histogramas

Em ggplot2, a implementação de Wickham da gramática de Wilkinson é uma estrutura fácil de aprender para o código de gráficos R. Para aprender essa estrutura, tenha o ggplot2 em sua biblioteca para que possa seguir o que vem agora. (Encontre ggplot2 na aba Packages e clique em sua caixa de verificação.)

Um gráfico começa com ggplot(), que recebe dois argumentos. O primeiro é a fonte dos dados. O segundo mapeia os componentes dos dados de interesse para os componentes do gráfico. A função que faz isso é aes().

Para começar um histograma para Price em Cars93, a função é

```
> ggplot(Cars93, aes(x=Price))
```

A função aes() associa Price ao eixo x. No mundo de ggplot, isso é chamado de *mapeamento estético*. Na verdade, cada argumento de aes() é chamado de *estética*.

Essa linha de código desenha a Figura 3-17, que é apenas uma grade com um fundo cinza e Price no eixo x.

Bem, e o eixo y? Alguma coisa nos dados é mapeada nele? Não. Isso porque ele é um histograma e nada explicitamente nos dados fornece um valor de y para cada x. Então não é possível dizer "y=" em aes(). Em vez disso, deixamos que R calcule as alturas das barras no histograma.

E o histograma? Como o colocamos nessa grade em branco? É preciso adicionar algo indicando que queremos diagramar um histograma e deixar que R cuide do resto. Adicionamos a função geom ("geom" é a abreviação de "objeto geométrico").

Existem vários tipos de funções geom. ggplot2 fornece uma para quase todas as necessidades gráficas e dá a flexibilidade de trabalhar com casos especiais. Para desenhar um histograma, a função geom a ser usada é chamada de geom_histogram().

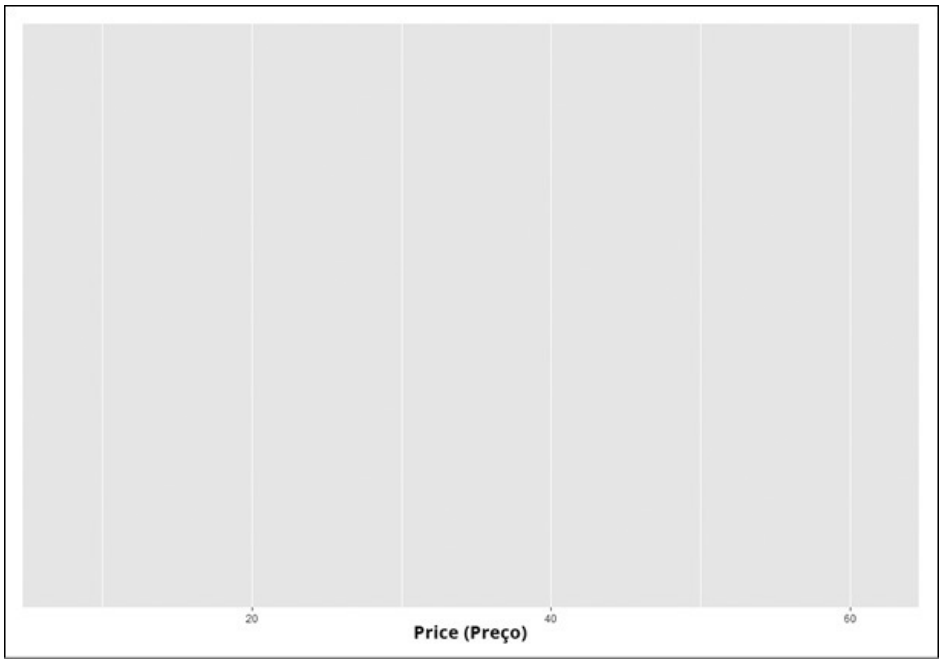


FIGURA 3-17:
Aplicando apenas ggplot().

Como adicionar geom_histogram() a ggplot()? Com um sinal de adição:

```
ggplot(Cars93, aes(x=Price)) +
geom_histogram()
```

Isso produz a Figura 3-18. As regras gramaticais dizem a ggplot2 que quando o objeto geométrico é um histograma, R faz os cálculos necessários nos dados e produz o diagrama adequado.

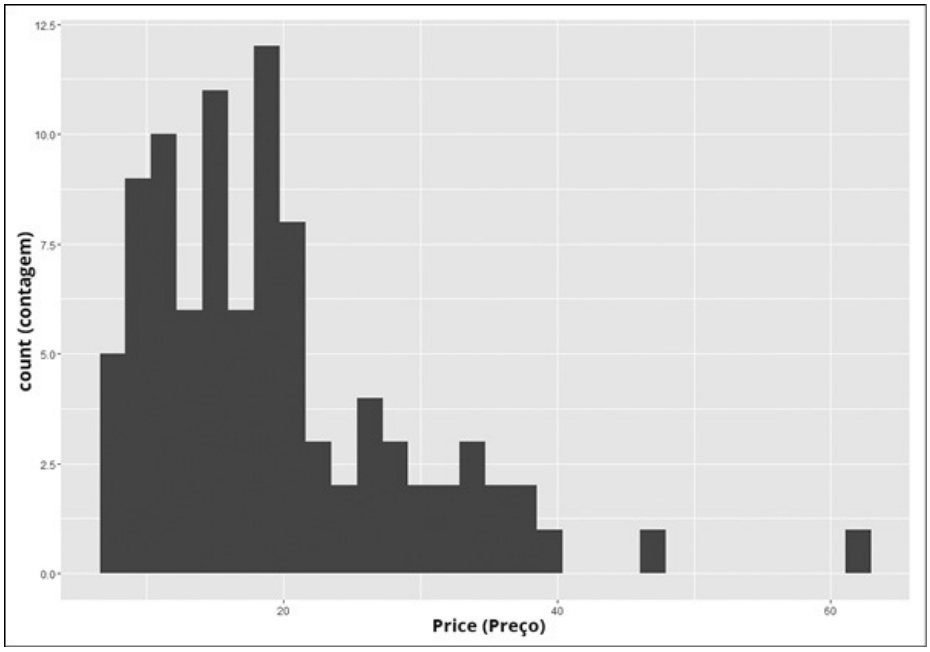


FIGURA 3-18:
Histograma inicial para Price em Cars93.

No mínimo, o código gráfico de ggplot2 precisa ter dados, mapeamentos estéticos e um objeto geométrico. É como responder a uma sequência lógica de perguntas: Qual é a fonte dos dados? Você tem interesse em qual parte dos dados? Quais partes dos dados correspondem a quais partes do gráfico? Que aparência o gráfico deve ter?

Além dessas exigências mínimas, é possível modificar o gráfico. Cada barra é chamada de *bin* e um ggplot() padrão usa 30 delas. Depois de diagramar o histograma, ggplot() exibe uma mensagem na tela que aconselha experimentar binwidth (que especifica a largura de cada bin) para mudar a aparência do gráfico. Portanto, use binwidth = 5 como um argumento em geom_histogram().

Argumentos adicionais modificam a aparência das barras:

```
geom_histogram(binwidth=5, color = “black”, fill = “white”)
```

Com outra função, labs(), é possível modificar os rótulos dos eixos e fornecer um título para o gráfico:

```
labs(x = "Price (x $1000)", y="Frequency",title="Prices of 93 Models
  of 1993 Cars")
```

Tudo junto agora:

```
ggplot(Cars93, aes(x=Price)) +
geom_histogram(binwidth=5,color="black",fill="white") +
labs(x = "Price (x $1000)", y="Frequency", title= "Prices of93
  Models of 1993 Cars")
```

O resultado é a Figura 3-19. (Note que ela é um pouco diferente da Figura 3-2. Eu tive que ajustar um pouco as duas para que ficassem iguais.)

Gráficos de barras

Desenhar um gráfico de barras no ggplot2 é um pouco mais fácil do que desenhar um na base R: não é necessário criar uma tabela como a Tabela 3-1 antes de desenhar o gráfico. Como no exemplo da seção anterior, não é preciso especificar um mapeamento estético para y. Dessa vez, a função geom é geom_bar() e as regras gramaticais informam a ggplot2 para fazer o trabalho necessário com os dados, então, desenhar o gráfico:

```
ggplot(Cars93, aes(x=Type))+
geom_bar() +
labs(y="Frequency", title="Car Type and Frequency in Cars93")
```

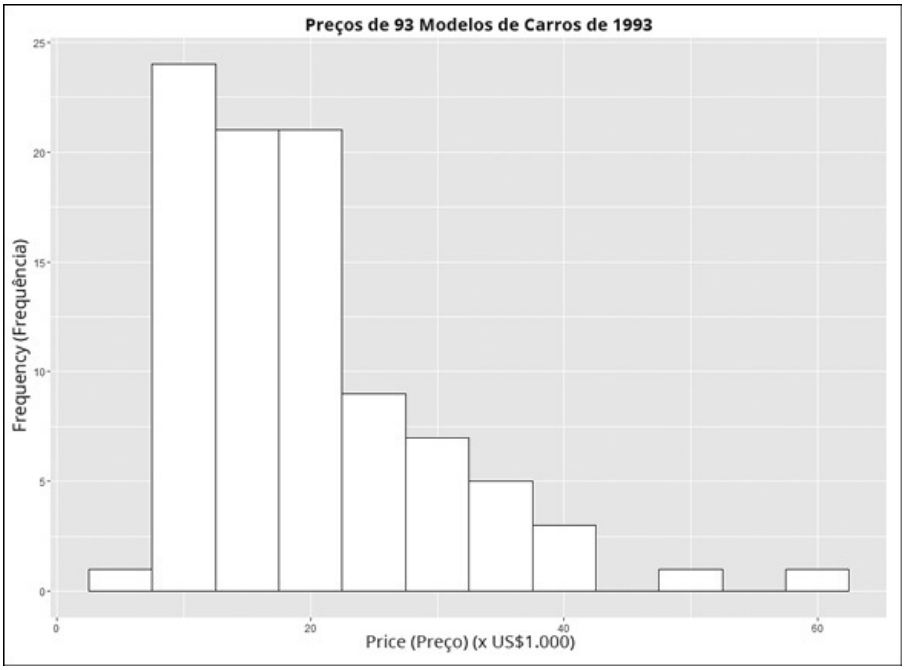


FIGURA 3-19:
Histograma Price finalizado.

A Figura 3-20 mostra o gráfico de barras resultante.

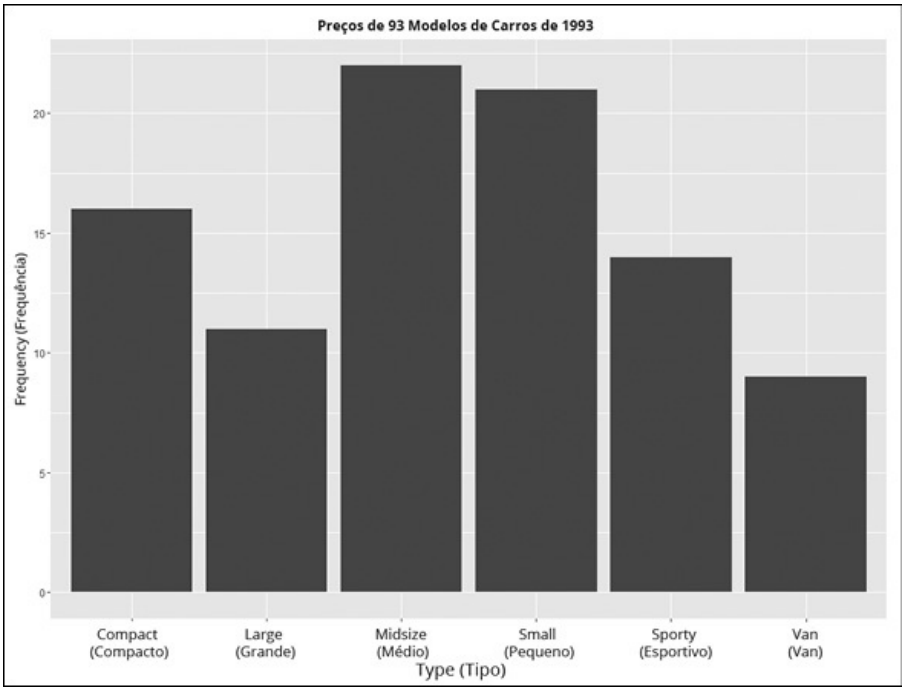


FIGURA 3-20:
Gráfico de barras para Car Type.

Gráficos de pontos

Anteriormente neste capítulo, mostrei o gráfico de pontos como uma alternativa ao gráfico de pizza. Nesta seção mostro como usar ggplot() para desenhá-lo.

DICA

Por que eu não comecei com o gráfico de pizza e mostrei como criar um com o pacote ggplot2? É muito trabalhoso, e não vale tanto a pena. Se quiser criar um, a função pie() de base R é muito mais fácil e usar.

O começo da criação de um gráfico de pontos é bem parecido com o de base R: crie uma tabela para Type (Tipo) e transforme-a em um data frame.

```
type.frame <- data.frame(table(Cars$93.Type))
```

Para garantir que os nomes de variáveis sejam significativos para o mapeamento estético, aplique a função colnames() ao nome das colunas nesse data frame. (Esse passo não existe em base R.)

```
colnames(type.frame)<- c("Type","Frequency")
```

Agora type.frame se parece exatamente como a Tabela 3-1:

```
> type.frame
Type Frequency
1 Compact 16
```

```
2 Large 11
3 Midsize 22
4 Small 21
5 Sporty 14
6 Van 9
```

E agora o gráfico. Para orientar o gráfico de pontos como na Figura 3-11, mapeie Frequency para o eixo x e Type para o eixo y:

```
ggplot(type.frame, aes(x=Frequency,y= Type))
```

Novamente, a variável independente geralmente fica no eixo x e a dependente no eixo y, mas esse não é o caso neste gráfico.

Em seguida, adicione uma função geom.

CUIDADO

Há uma função geom disponível chamada geom_dotplot(), mas, surpreendentemente, ela não é adequada aqui. Ela desenha algo diferente. No mundo ggplot, um *diagrama* de pontos é diferente de um *gráfico* de pontos. Vai entender!

A função geom para o gráfico de pontos é geom_point(). Então este código

```
ggplot(type.frame, aes(x=Frequency,y=Type)) +
geom_point()
```

resulta na Figura 3-21.

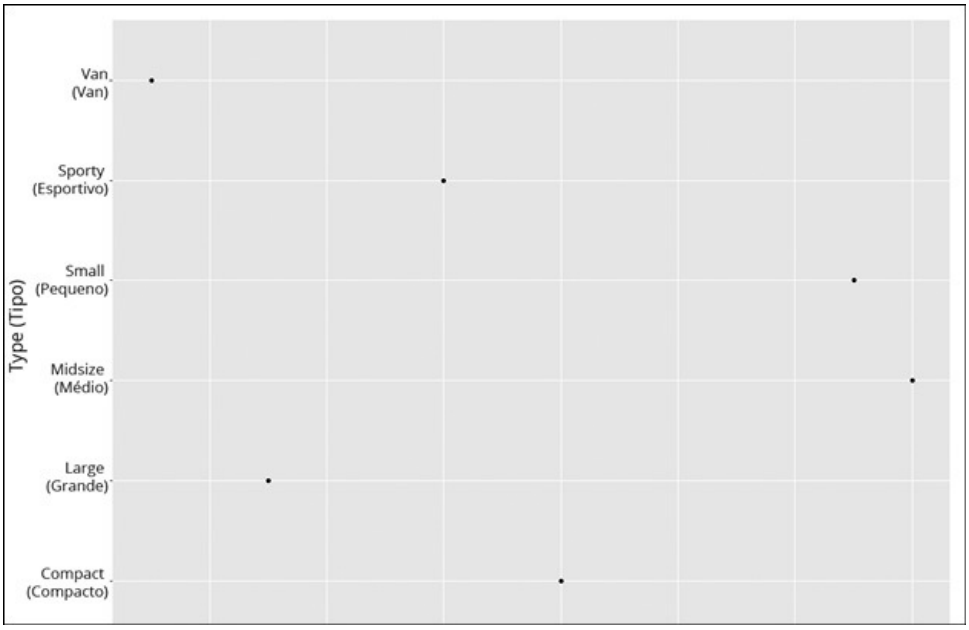


FIGURA 3-21:

Gráfico de pontos inicial de Type.

Há algumas modificações a fazer. Primeiro, com um gráfico como este, é bom organizar as categorias no eixo y respeitando a ordem em que são medidas no eixo x. Isso pede uma pequena mudança no mapeamento estético do eixo y:

```
ggplot(type.frame, aes(x=Frequency,y=reorder(Type,Frequency))
```

Pontos maiores deixam a aparência do gráfico um pouco melhor:

```
geom_point(size =4)
```

Funções adicionais modificam a aparência geral do gráfico. Uma família dessas funções é chamada de *temas*. Um membro dessa família, theme_bw(), remove o fundo cinza. Adicionar theme()com os argumentos adequados a) remove as linhas verticais na grade e b) enegrece as linhas horizontais e as torna pontilhadas:

```
theme_bw() +
theme(panel.grid.major.x=element_blank(),
panel.grid.major.y=element_line(color = "black", linetype = "dotted"))
```

Por fim, labs()muda o rótulo do eixo y:

```
labs(y= “Type”)
```

Sem essa mudança, o rótulo do eixo y seria "reorder(Type,Frequency)". Embora seja pitoresco, o rótulo dá um leve sentido ao público comum.

Veja o código do início ao fim:

```
ggplot(type.frame, aes(x=Frequency,y=reorder(Type,Frequency))) +
geom_point(size = 4) +
theme_bw() +
theme(panel.grid.major.x=element_blank(),
panel.grid.major.y=element_line(color = "black", linetype=
  "dotted"))+
labs(y="Type")
```

A Figura 3-22 mostra o gráfico de pontos.

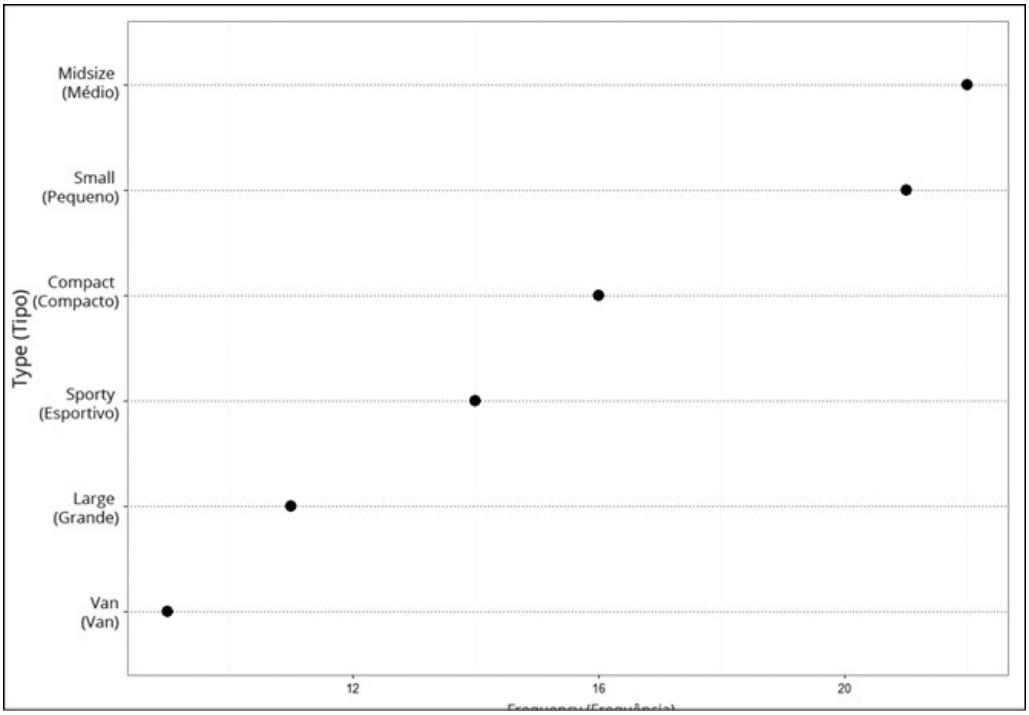


FIGURA 3-22:
Gráfico de pontos modificado de Type.

Revendo de novo os gráficos de barra

Como no caso dos primeiros gráficos em base R, os mostrados até agora nesta seção têm frequências (ou "contagens") como a variável dependente. E claro, como mostra a Tabela 3-2, esse nem sempre é o caso.

Na seção sobre base R, mostro como criar um gráfico de barras agrupadas. Aqui eu mostro como usar ggplot() para criar um com space.rev, o conjunto de dados que criei com os dados da Tabela 3-2. O produto final será igual ao da Figura 3-23.

Primeiro, prepare os dados. Eles não estão no formato que ggplot() usa. Este formato

```
> space.rev
1990 1991 1992 1993 1994
Commercial Satellites Delivered 1000 1300 1300 1100 1400
Satellite Services 800 1200 1500 1850 2330
Satellite Ground Equipment 860 1300 1400 1600 1970
Commercial Launches 570 380 450 465 580
Remote Sensing Data 155 190 210 250 300
```

é chamado de formato *largo*. No entanto, ggplot()trabalha com o formato *longo*, que se parece com isto:

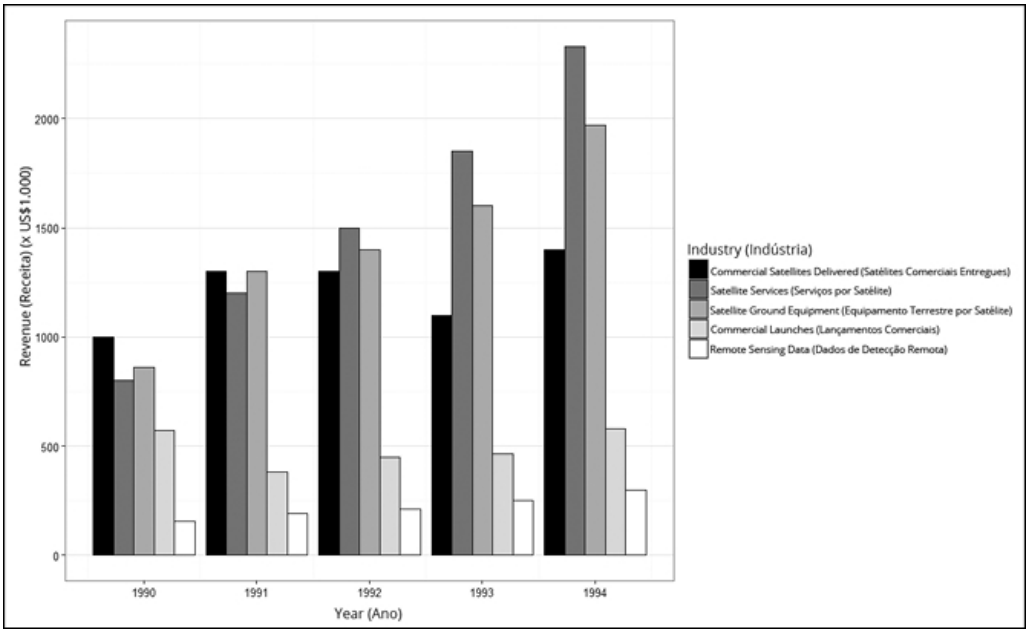


FIGURA 3-23:
Gráfico de barras para os dados da Tabela 3-2 com ggplot().

```
Industry Year Revenue
1 Commercial Satellites Delivered 1990 1000
2 Satellite Services 1990 800
3 Satellite Ground Equipment 1990 860
4 Commercial Launches 1990 570
5 Remote Sensing Data 1990 155
6 Commercial Satellites Delivered 1991 1300
```

Essas são as seis primeiras linhas do conjunto de dados. O número total de linhas é 25 (porque 5 linhas e 5 colunas estão no formato largo).

Hadley Wickham (olha o nome de novo!) criou um pacote chamado reshape2, que fornece tudo para uma transformação suave. A função melt() transforma o formato largo em longo. Outra função, cast(), faz o contrário. Elas são de grande ajuda, pois eliminam a necessidade de ficar procurando em planilhas para reformatar um conjunto de dados.

Então, com reshape2 na biblioteca (clique em sua caixa de verificação na aba Packages), o código é

```
> space.melt <- melt(space.rev)
```

Sim, é só isso. Veja, eu provo para você:

```
> head(space.melt)
Var1 Var2 value
1 Commercial Satellites Delivered 1990 1000
2 Satellite Services 1990 800
3 Satellite Ground Equipment 1990 860
```

```
4 Commercial Launches 1990 570
5 Remote Sensing Data 1990 155
6 Commercial Satellites Delivered 1991 1300
```

Em seguida, dê nomes significativos para as colunas:

```
> colnames(space.melt) <- c("Industry","Year","Revenue")
> head(space.melt)
Industry Year Revenue
1 Commercial Satellites Delivered 1990 1000
2 Satellite Services 1990 800
3 Satellite Ground Equipment 1990 860
4 Commercial Launches 1990 570
5 Remote Sensing Data 1990 155
6 Commercial Satellites Delivered 1991 1300
```

E agora estamos prontos. Comece com ggplot(). Os mapeamentos estéticos são fáceis:

```
ggplot(space.melt, aes(x=Year,y=Revenue,fill=Industry))
```

Adicione a função geom para a barra e especifique três argumentos:

```
geom_bar(stat = "identity", position = "dodge", color ="black")
```

O primeiro argumento é absolutamente necessário para um gráfico desse tipo. Se deixado por conta própria, geom_bar terá como padrão o gráfico de barras mostrado anteriormente, um gráfico baseado em frequências. Como definimos um mapeamento estético para y e esse tipo de gráfico é incompatível com uma estética para y, não configurar esse argumento resulta em uma mensagem de erro.

Assim, informe a ggplot() que esse é um gráfico baseado em valores de dados explícitos. Então stat="identity" significa "use os números apresentados como dados".

O valor do próximo argumento, position, é um nome bonitinho que significa que as barras "desviam" uma das outras e se alinham lado a lado. (Omita esse argumento e veja o que acontece.) Ele é análogo a "beside =T" em base R.

O terceiro argumento estabelece a cor das bordas de cada barra. O esquema de preenchimento de cor para as barras é a área da função a seguir:

```
scale_fill_grey(start = 0,end = 1)
```

Como o nome sugere, a função preenche as barras com tons de cinza. O valor start, 0, é preto e o valor end, 1, é branco. (Lembra "grey0" = "black" e "grey100" = "white".) O efeito é o preenchimento das cinco barras com cinco tons de preto a branco.

Renomeie o eixo y, assim

```
labs(y="Revenue (X $1,000)")
```

então remova o fundo cinza

```
theme_bw()
```

e, finalmente, remova as linhas verticais da grade

```
theme(panel.grid.major.x = element_blank())
```

Tudo que é necessário para produzir a Figura 3-23 é

```
ggplot(space.melt, aes(x=Year,y=Revenue,fill=Industry)) +
geom_bar(stat = "identity", position = "dodge", color="black") +
scale_fill_grey(start = 0,end = 1)+
labs(y="Revenue (X $1,000)")+
theme_bw()+
theme(panel.grid.major.x = element_blank())
```

Diagramas de dispersão

Como descrevi anteriormente, um diagrama de dispersão é uma ótima maneira de mostrar o relacionamento entre duas variáveis, como potência e milhas por galão (km/l) para dirigir na cidade. E ggplot() é uma ótima maneira de desenhar um diagrama de dispersão. Se você chegou até aqui, a gramática para fazer isso será fácil:

```
ggplot(Cars93,aes(x=Horsepower,y=MPG.city))+
geom_point()
```

A Figura 3-24 mostra o diagrama de dispersão. Deixo para que você mude o rótulo do eixo y para "Milhas por Galão (Cidade)" e adicione um título descritivo.

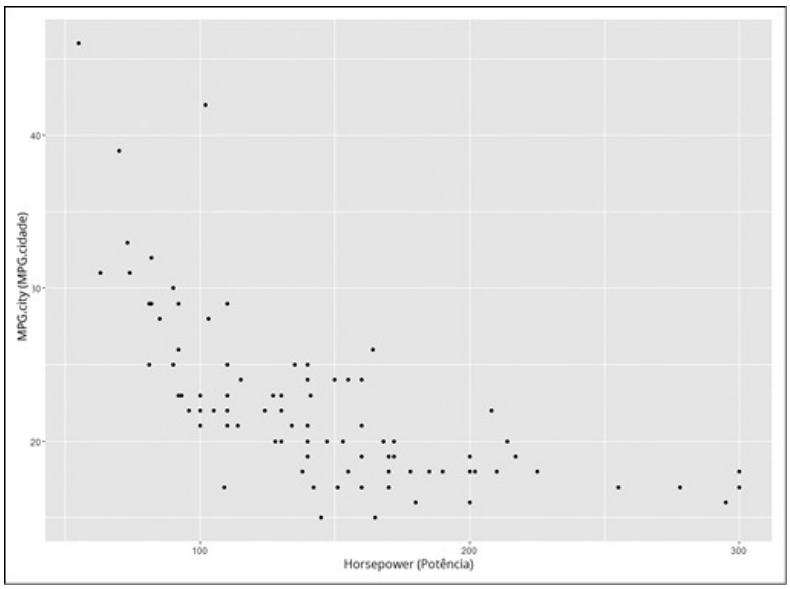


FIGURA 3-24:

MPG.city X Horsepower em Cars93.

E aquela reviravolta...

Dê outra olhada na Figura 3-15, o relacionamento entre MPG.city e Horsepower. Nele, as marcas do diagrama não são pontos. Em vez disso, cada marca de dado é o número de cilindros, que é um rótulo que aparece como um caractere de texto.

Como fazer isso no mundo do ggplot? Primeiro é preciso um mapeamento estético adicional em aes(). Esse mapeamento é label e está definido para Cylinders:

```
ggplot(Cars93, aes(x=Horsepower, y=MPG.city, label = Cylinders))
```

Adicione um objeto geométrico para o texto e voilà:

```
ggplot(Cars93, aes(x = Horsepower,y = MPG.city,label = Cylinders))
  +
geom_text()
```

A Figura 3-25 mostra o gráfico que esse código produz. Uma diferença da base R é o "rotatório", em vez de "r", como rótulo da marca de dados.

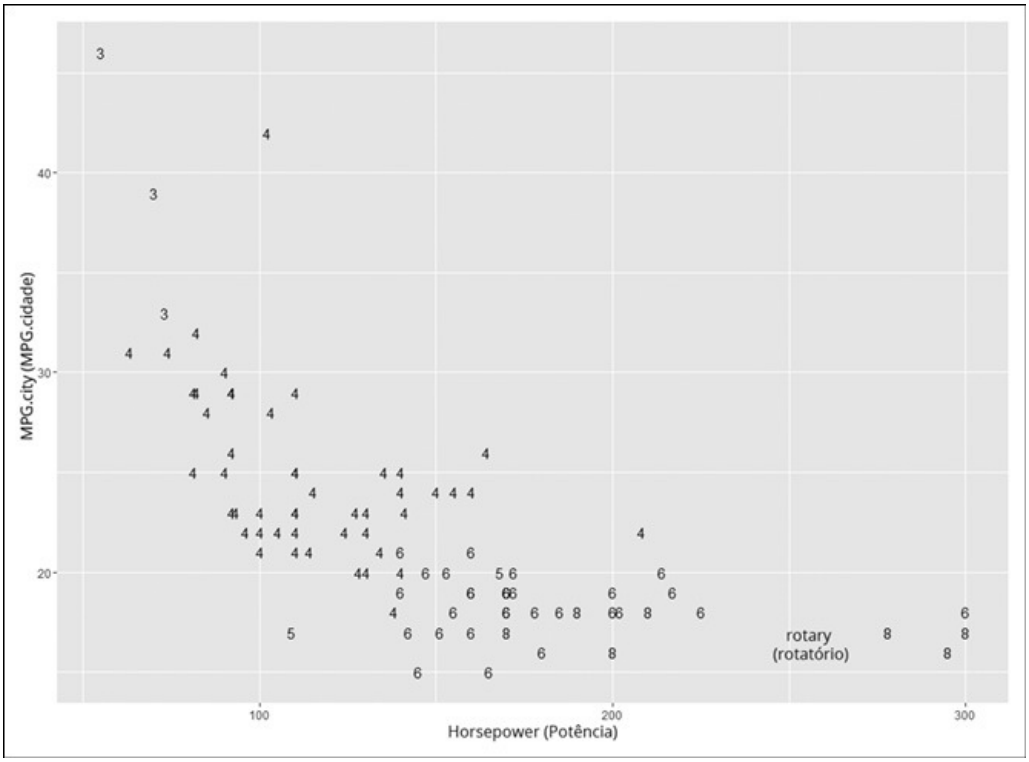


FIGURA 3-25:
Diagrama de dispersão inicial para MPG.city X Horsepower com Cylinders como o rótulo da marca de dados.

Só por diversão, usei funções de tema (veja a seção anterior, "Gráficos de pontos") para deixar a aparência do gráfico como a apresentada na Figura 3-

15. Como no exemplo do gráfico de pontos, theme_bw() elimina o fundo cinza. A função theme() (com um argumento específico) elimina a grade:

```
theme(panel.grid=element_blank())
```

element_blank() é uma função que desenha um elemento em branco. Juntando tudo,

```
ggplot(Cars93, aes(x=Horsepower, y=MPG.city, label=Cylinders)) +
geom_text() +
theme_bw() +
theme(panel.grid=element_blank())
```

produz a Figura 3-26. Mais uma vez, deixo para que você use labs() para mudar o rótulo do eixo y e adicione um título descritivo.

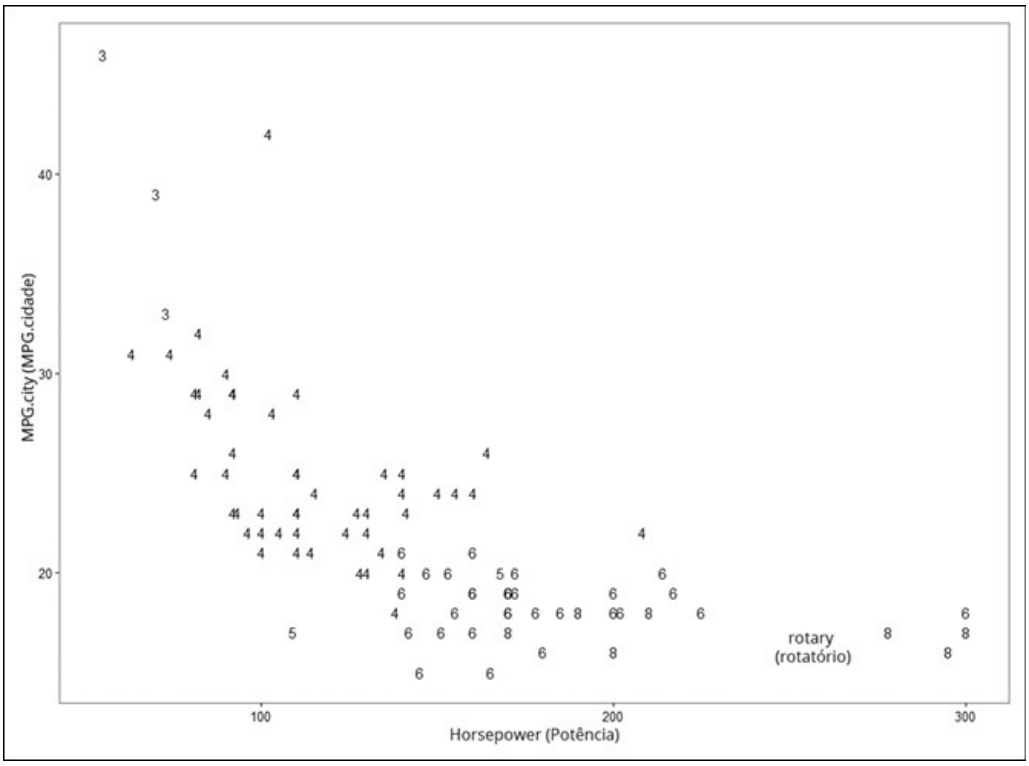


FIGURA 3-26:
Diagrama de dispersão modificado para MPG.city X Horsepower com Cylinders como rótulo da marca de dados.

Matriz do diagrama de dispersão

Uma matriz do diagrama de dispersão mostra os relacionamentos emparelhados entre duas variáveis. A Figura 3-16 mostra como a função pairs() de base R desenha esse tipo de matriz.

O pacote ggplot2 tinha uma função chamada plotpairs(), que fazia algo parecido, mas não tem mais. GGally, um pacote baseado no ggplot2, fornece ggpairs() para desenhar matrizes de diagramas de dispersão, e faz isso de forma extravagante.

DICA

O pacote GGally não está na aba Packages. É preciso selecionar Install e digitar **GGally** na caixa de diálogo Install Packages. Quando aparecer na aba Packages, clique na caixa de verificação ao lado dele.

Anteriormente, criei um subconjunto de Cars93 que inclui MPG.city, Price e Horsepower:

```
> cars.subset <- subset(Cars93, select =
  c(MPG.city,Price,Horsepower))
```

Com o pacote GGally em sua biblioteca, este código cria a matriz do diagrama de dispersão na Figura 3-27:

```
> ggpairs(cars.subset)
```

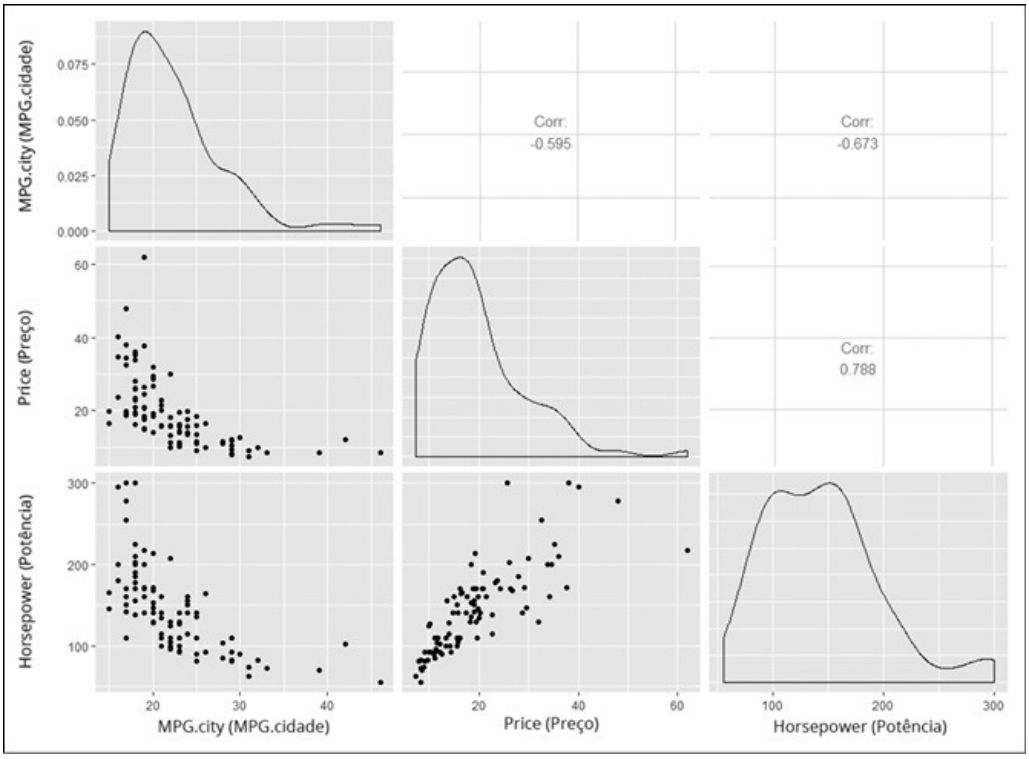


FIGURA 3-27:
Matriz do diagrama de dispersão para MPG.city, Price e Horsepower.

Como mostra a Figura 3-27, ficou lindo. As células na diagonal principal apresentam diagramas de densidade das variáveis. (Veja a subseção anterior "Adicionando recursos gráficos" e também o Capítulo 8.) Uma desvantagem é que o eixo y fica visível para a variável MPG.city apenas na primeira linha e primeira coluna.

Os três diagramas de dispersão estão nas células abaixo da diagonal principal. Em vez de mostrar os mesmos diagramas de dispersão com os eixos trocados nas células acima da diagonal principal (como pairs() faz), cada célula acima

mostra um *coeficiente de correlação* que resume o relacionamento entre as variáveis de linha e coluna da célula. (Coeficientes de correlação? Não, não vou explicá-los agora. Veja o Capítulo 15.)

Para ter um visual mais agradável, adicione Cylinders a cars.subset e aplique ggpairs():

```
> cars.subset <- subset(Cars93, select = c(MPG. city,Price,
  Horsepower,Cylinders))
> ggpairs(cars.subset)
```

A Figura 3-28 mostra a nova matriz do diagrama de dispersão com toda sua elegância.

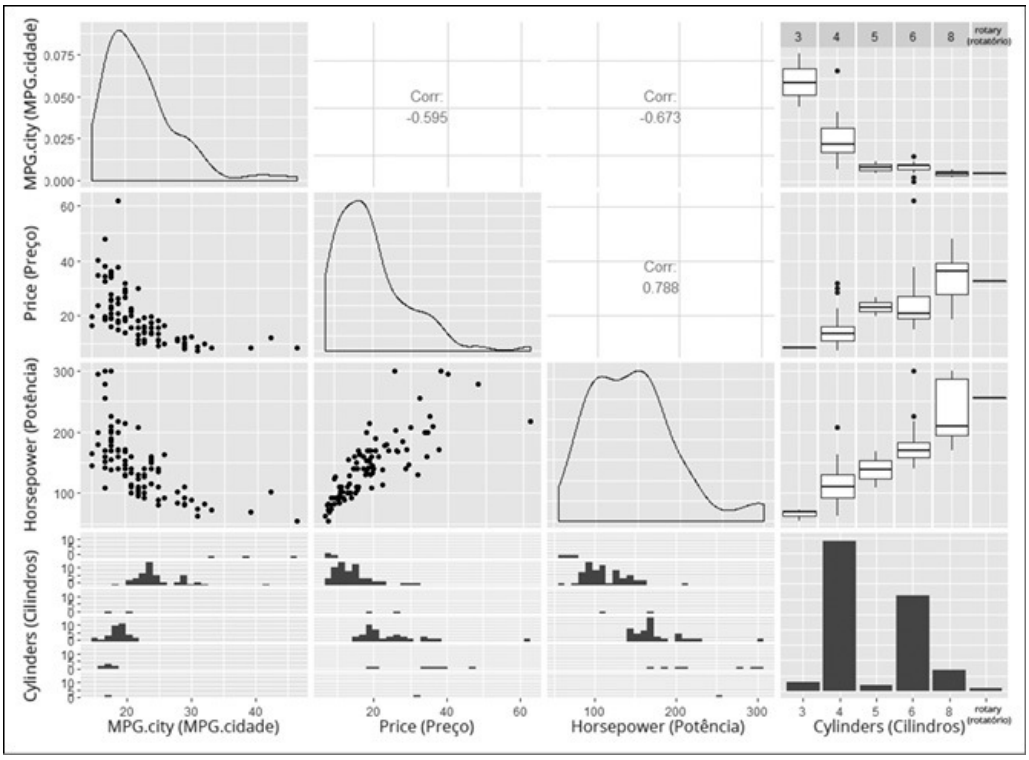


FIGURA 3-28:
Adicionar Cylinders produz esta matriz do diagrama de dispersão.

Cylinders não é uma variável que serve para os diagramas de dispersão ou coeficientes de relação. (Pergunta: Por que não?) Assim, a célula na quarta coluna, quarta linha, tem um gráfico de barras, em vez de um diagrama da densidade. Os gráficos de barras relacionando Cylinders (em cada eixo y) às outras três variáveis (nos eixos x) ficam nas três células restantes da linha 4. Os diagramas de caixa relacionando Cylinders (em cada eixo x) às outras três variáveis (nos eixos y) ficam nas três células restantes da coluna 4.

O que nos leva ao próximo tipo de gráfico...

Diagramas de caixa

Os estatísticos usam diagramas de caixa para mostrar rapidamente como os grupos diferem uns dos outros. Como no exemplo de base R, mostro o

diagrama de caixa para Cylinders (Cilindros) e Horsepower (Potência). É uma replicação do gráfico na linha 3, coluna 4 da Figura 3-28.

A essa altura você provavelmente pode adivinhar a função ggplot():

```
ggplot(Cars93, aes(x=Cylinders, y= Horsepower))
```

Qual é a função geom? Se você chutou geom_boxplot(), acertou!

Então o código é

```
ggplot(Cars93, aes(x=Cylinders,y=Horsepower)) +
geom_boxplot()
```

E isso nos dá a Figura 3-29.

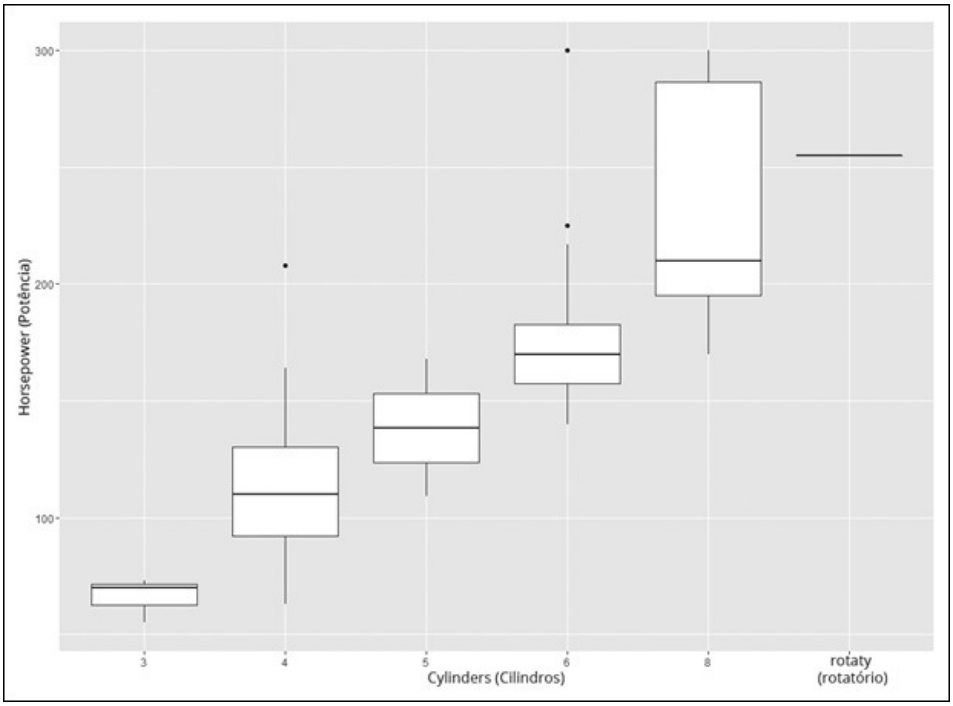


FIGURA 3-29:
Diagrama de caixa para Horsepower X Cylinders.

Quer mostrar todas as marcas de dados além das caixas? Adicione a função geom para as marcas

```
ggplot(Cars93, aes(x=Cylinders,y=Horsepower)) +
geom_boxplot()+
geom_point()
```

para produzir o gráfico na Figura 3-30.

Lembre-se de que são dados de 93 carros. Você vê 93 marcas de dados? Nem eu. Isso, claro, acontece porque muitas se sobrepõem. Os gurus de gráficos chamam isso de *overplotting*.

Uma maneira de lidar com o overplotting é reposicionar aleatoriamente as marcas para que apareçam, mas não mudem o que representam. Isso é

chamado de *jittering*. E o ggplot2 tem uma função geom para isso: geom_jitter(). Adicionar essa função ao código

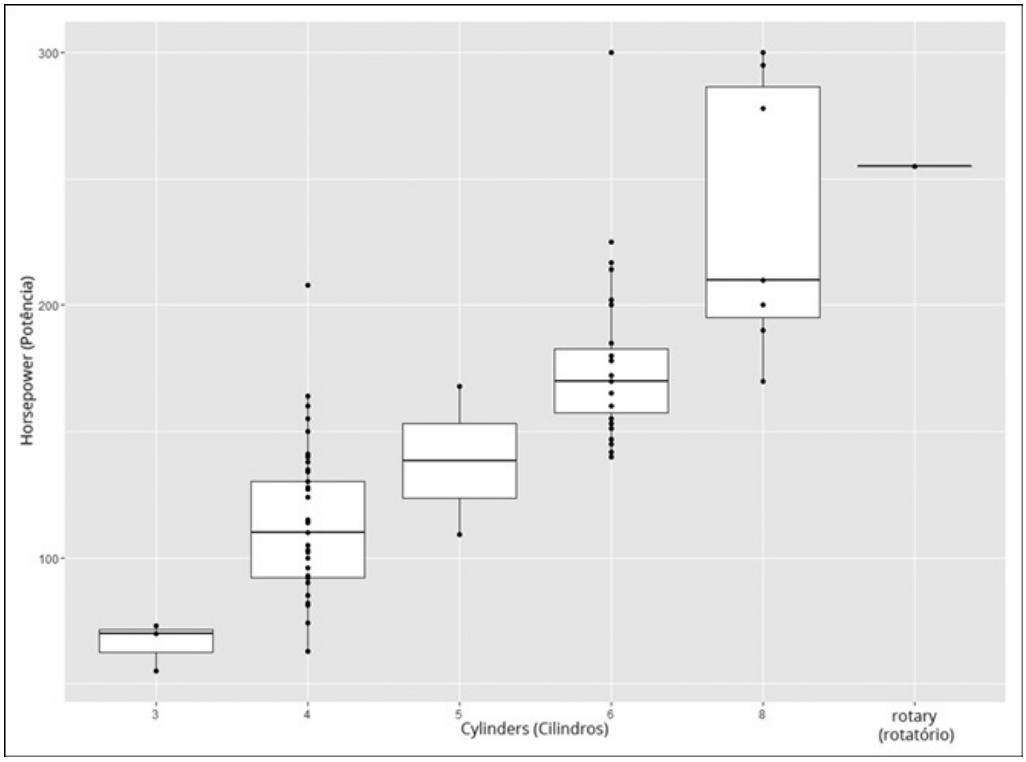


FIGURA 3-30:
Diagrama de caixa com marcas de dados.

```
gplot(Cars93, aes(x=Cylinders,y=Horsepower)) +
geom_boxplot()+
geom_point()+
geom_jitter()
```

desenha a Figura 3-31.

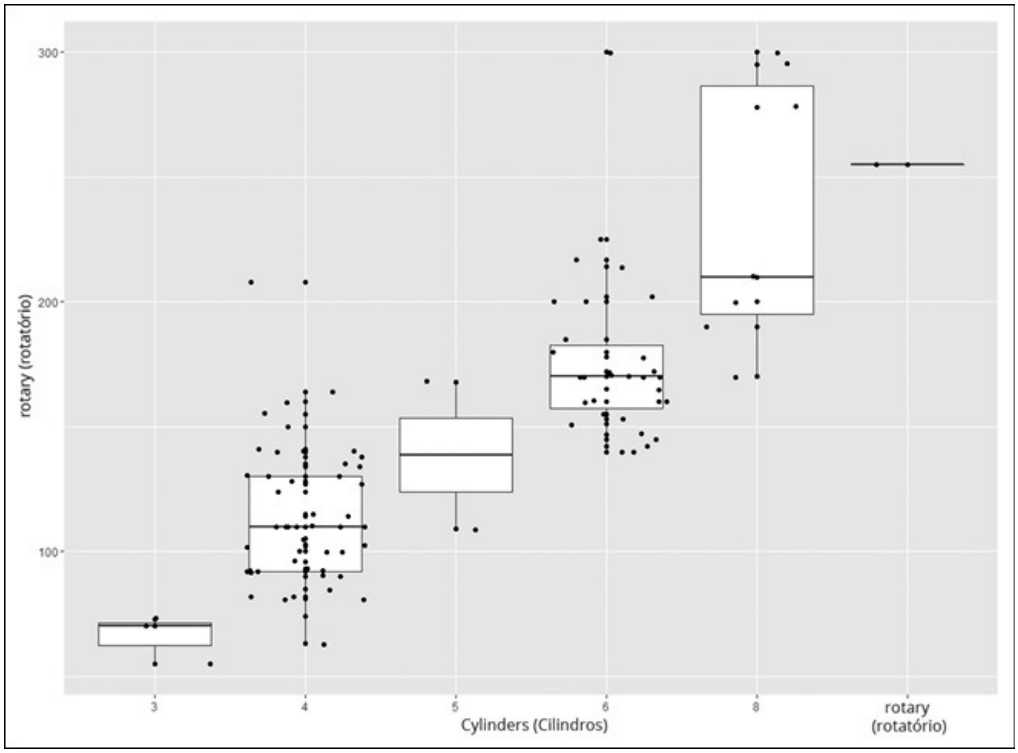


FIGURA 3-31:
Diagrama de caixa com marcas de dados com jittering.

Encerrando

No que diz respeito aos gráficos, apenas dei uma passada superficial. O R tem um conjunto avançado de ferramentas e pacotes de gráficos, muito mais do que eu poderia mostrar neste capítulo. Nos capítulos que estão por vir, sempre que eu mostrar uma técnica analítica, também mostrarei como visualizar seus resultados. Usarei o que você leu neste capítulo junto com novas ferramentas e pacotes quando for necessário.

Capítulo **4**

NESTE CAPÍTULO

- **Trabalhando dentro das médias**
- **Satisfazendo condições**
- **Entendendo que a mediana é a mensagem**
- **Entrando na moda**

Encontrando Seu Centro

Se você já trabalhou com um conjunto de números e teve que descobrir como resumi-los em um único número, enfrentou uma situação que os estatísticos encaram o tempo todo. De onde vem esse "número único" ideal?

Uma boa ideia pode ser selecionar um número em algum lugar do meio do conjunto. Esse número poderia, então, representar o conjunto inteiro de números. Quando procuramos no meio do conjunto, observamos a *tendência central*. É possível abordar a tendência central de várias maneiras.

Médias: Atração

Todos já usamos *médias*. A média é uma maneira fácil de resumir seus gastos, suas notas escolares, sua performance esportiva ao longo do tempo.

No decorrer do trabalho, os cientistas calculam médias. Quando um pesquisador faz um estudo, ele aplica algum tipo de tratamento ou procedimento a uma pequena amostra de pessoas ou coisas. Então mede os resultados e estima os efeitos do procedimento na população que produziu a amostra. Os estatísticos demostram que a média amostral é a estimativa da média da população.

Acho que você sabe como calcular a média, mas a veremos de qualquer forma. Depois mostrarei a fórmula estatística. Meu objetivo é que você entenda as fórmulas estatísticas em geral, então mostrarei como R calcula as médias.

Uma *média* é apenas a soma de um conjunto de números dividido pela quantidade de números somados. Suponha que precisemos medir as alturas (em polegadas) de seis crianças de 5 anos de idade e descobrimos que suas alturas são

36, 42, 43, 37, 40, 45

A altura média dessas seis crianças é

$$\frac{36+42+43+37+40+45}{6}=40{,}5$$

A média desse exemplo é, então, 40,5 polegadas (1,03m).

A primeira tentativa de uma fórmula para a média pode ser

$$\text{Média} = \frac{\text{Soma dos Números}}{\text{Quantidade de Números Somados}}$$

No entanto, as fórmulas normalmente envolvem abreviações. Uma abreviação comum para "Número" é *X*. Os estatísticos geralmente abreviam "Quantidade de Números Somados" como *N*. Então a fórmula fica

$$\text{Média} = \frac{\text{Soma de } X}{N}$$

Os estatísticos também usam uma abreviação para *Soma de* — a letra grega S maiúscula. Chamada de "sigma", ela tem essa aparência: Σ. Então a fórmula com sigma é

$$\text{Média} = \frac{\sum X}{N}$$

Ainda não terminamos. Os estatísticos também abreviam a "média". Você pode achar que a abreviação seria um *M*, e alguns estatísticos concordam com você, mas a maioria prefere um símbolo relacionado a *X*. Por isso, a abreviação mais popular para a média é $\bar{x}$, que lemos como "X barra". Veja a fórmula:

$$\bar{X} = \frac{\sum X}{N}$$

Preciso resolver mais uma pendência. No Capítulo 1 eu falo sobre amostras e populações. Os símbolos nas fórmulas devem refletir a distinção entre as duas. A convenção é que as letras do alfabeto latino, como $\bar{x}$, representam as características das amostras e as letras gregas representam as características das populações. Para a média da população, o símbolo é o equivalente grego de *M*, que é μ. Pronuncia-se "mi". A fórmula para a média da população é

$$\mu = \frac{\sum X}{N}$$

Média em R: mean()

R fornece uma maneira extremamente simples de calcular a média de um conjunto de números: mean(). Vamos aplicá-la ao exemplo das alturas das seis crianças.

Primeiro, crie um vetor das alturas:

```
> heights <- c(36, 42, 43, 37, 40, 45)
```

Depois aplique a função:

```
> mean(heights)
[1] 40.5
```

E é isso.

Qual é sua condição?

Quando trabalhamos com um data frame, às vezes queremos calcular a média apenas dos casos (linhas) que satisfazem certas condições, em vez da média de todos eles. Isso é fácil de fazer em R.

Para a análise que vem a seguir, usei o mesmo data frame Cars93 do Capítulo 3. É aquele que tem dados de uma amostra de 93 carros de 1993. Está no pacote MASS. Então verifique se o pacote MASS está em sua biblioteca. (Encontre MASS na aba Packages e clique em sua caixa de verificação.)

Suponha que estejamos interessados na potência média dos carros feitos nos EUA. Primeiro selecione esses carros e coloque suas potências em um vetor:

```
Horsepower.USA <- Cars93$Horsepower[Cars93$Origin == “USA”]
```

(Se a parte do lado direito da linha parece estranha, releia o Capítulo 2.)

A potência média é, então,

```
> mean(Horsepower.USA)
[1] 147.5208
```

Hmm, imagino qual será a média para os carros não fabricados nos EUA:

```
Horsepower.NonUSA <- Cars93$Horsepower[Cars93$Origin ==
  "non-USA"]
> mean(Horsepower.NonUSA)
[1] 139.8889
```

As médias são um pouco diferentes. (Podemos examinar essa diferença mais de perto? Sim, podemos, e é o que faremos no Capítulo 11.)

Elimine os cifrões $ com with()

No código R anterior, os cifrões $ denotam variáveis no data frame Cars93. O R fornece uma maneira de não usar o nome do data frame (daí o cifrão) cada vez que nos referimos a uma de suas variáveis.

No Capítulo 3, mostrei que as funções gráficas recebem, como primeiro argumento, a fonte de dados. Depois, na lista de argumentos, não é necessário repetir a fonte junto do cifrão para indicar uma variável a ser diagramada.

A função with()faz isso para outras funções R. O primeiro argumento é a fonte de dados e o segundo é a função aplicada a uma variável na fonte de dados.

Para encontrar a potência média de carros dos EUA em Cars93:

```
> with(Cars93, mean(Horsepower[Origin == "USA"]))
[1] 147.5208
```

Isso também pula o passo de criar o vetor Horsepower.USA.

E que tal várias condições, como a potência média dos carros norte-americanos de quatro cilindros?

```
> with(Cars93, mean(Horsepower[Origin == "USA" & Cylinders ==4]))
```

```
[1] 104.0909
```

O R também fornece a função attach() como uma maneira de eliminar os cifrões e as teclas pressionada. Anexe o data frame (attach(Cars93), por exemplo) e não será necessário referir-se a ele novamente ao usar suas variáveis. Contudo, várias autoridades em R não recomendam isso, pois pode levar a erros.

Explorando os dados

Agora que examinamos as médias da potência dos carros norte-americanos e estrangeiros, que tal as distribuições gerais?

Isso pede um pouco de exploração de dados. Usamos o pacote ggplot2 (veja o Capítulo 3) para criar histogramas lado a lado a partir do data frame Cars93 para que possamos compará-los. (Verifique se ggplot2 está na biblioteca.) A Figura 4-1 exemplifica o que quero dizer.

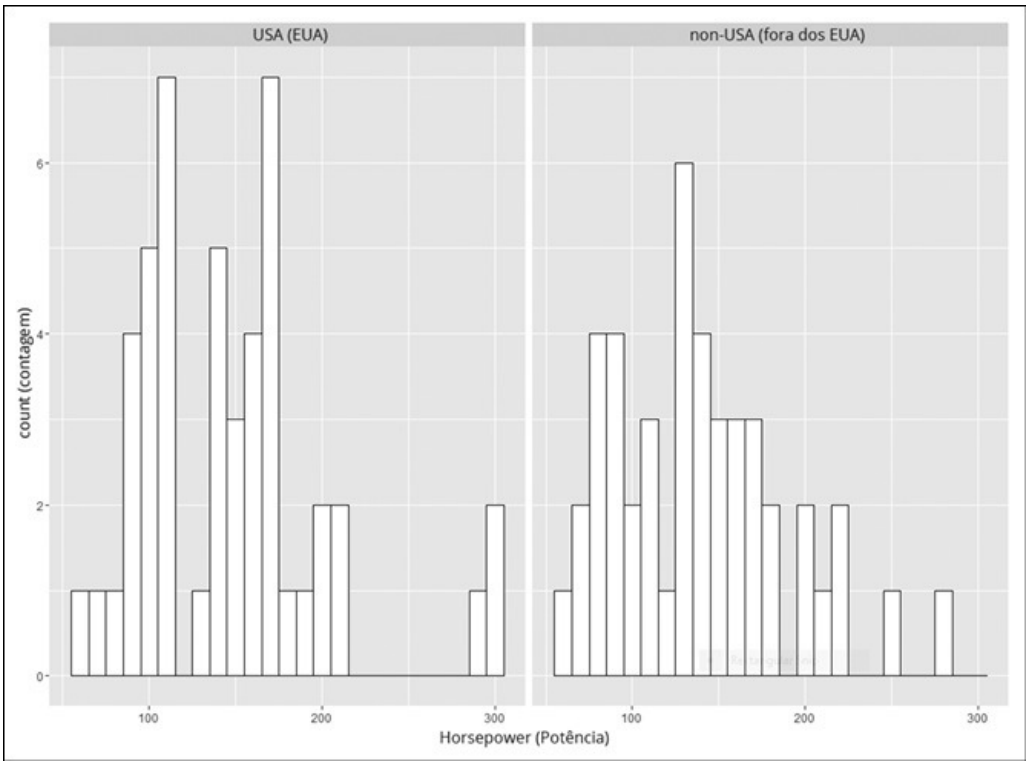


FIGURA 4-1:
Histogramas de potência para carros norte-americanos e de outros países em Cars93.

Para criar os histogramas da figura, comece da maneira usual:

```
ggplot(Cars93, aes(x=Horsepower))
```

E depois adicione uma função geom:

```
geom_histogram(color="black", fill="white",binwidth = 10)
```

Eu ajustei um pouco para chegar ao valor binwidth.

Até agora, o código criou um histograma normal com Horsepower no eixo x. Como podemos criar a Figura 4-1? Para isso, adicione um recurso do ggplot chamado *facetagem*. Em termos simples, a facetagem divide os dados de acordo com uma variável nominal, como Origin, que pode ser "EUA" ou "fora dos EUA". Há algumas funções de facetagem disponíveis. A que usei aqui é chamada facet_wrap(). Para dividir os dados de acordo com Origin, insira

```
facet_wrap(~Origin)
```

Só um lembrete: o operador til (~) significa "depende de", então pense em Origin como uma variável independente. O código inteiro da Figura 4-1 é

```
ggplot(Cars93, aes(x=Horsepower)) +
geom_histogram(color="black", fill="white",binwidth = 10)+
facet_wrap(~Origin)
```

Como podemos ver, as distribuições têm formas gerais diferentes. Os carros norte-americanos parecem ter uma lacuna entre os 200 inferiores e os próximos valores mais altos; os estrangeiros nem tanto. Também podemos ver os valores máximos mais altos para os carros norte-americanos. Que outras diferenças você vê? (Eu abordo essas diferenças no Capítulo 7.)

Discrepâncias: A falha das médias

Uma *discrepância* é um valor extremo no conjunto de dados. Se o conjunto de dados for uma amostra e você estiver tentando estimar a média da população, a discrepância poderá enviesar a estimativa.

Os estatísticos lidam com as discrepâncias *aparando* a média, ou seja, eliminando os valores extremos inferior e superior antes de calcular a média amostral. A quantidade de cortes é uma porcentagem, como 5% das pontuações superiores e inferiores.

Por exemplo, o histograma à esquerda da Figura 4-1 mostra alguns valores extremos. Para cortar os 5% superiores e inferiores, adicione o argumento trim a mean():

```
> mean(Horsepower.USA, trim =.05)
[1] 144.1818
```

O resultado é um pouco menor do que a média não aparada.

LEMBRE-SE

Qual é a porcentagem apropriada para trim? Isso é você quem decide. Depende do que você está medindo, quão extremas as pontuações podem ser

e o quanto você conhece a área que está estudando. Ao reportar uma média aparada, informe ao seu público o que você fez e a porcentagem cortada.

Na próxima seção sobre a mediana, mostrarei outra maneira de lidar com as pontuações extremas.

Outros meios para um fim

Nesta seção, eu explico as duas médias adicionais diferentes daquelas com que estamos acostumados a trabalhar.

LEMBRE-SE

A média comum é chamada de *média aritmética*.

Quantos tipos diferentes de média são possíveis? Os matemáticos da Grécia antiga chegaram a 11!

Média geométrica

Suponha que você tenha um investimento de 5 anos que produz as seguintes porcentagens: 10%, 15%, 10%, 20% e 5%. (Sim, sim, eu sei. Isso é ficção.) Qual é a taxa de retorno médio anual?

Seu primeiro palpite deve ser a média dessas porcentagens. Essa média é 12%. E isso está incorreto.

Por quê? Ela deixa passar um ponto importante. No final do primeiro ano, você *multiplica* seu investimento por 0,10 e não adiciona 1,10 a ele. No final do segundo ano, multiplica o resultado do primeiro ano por 1,15, e assim por diante.

A média aritmética não fornece a taxa de retorno médio. Para isso, calcule a média da seguinte forma:

$$\text{Taxa de Retorno Médio} = \sqrt[5]{1{,}10 \times 1{,}15 \times 1{,}10 \times 1{,}20 \times 1{,}05} = 1{,}118847$$

A taxa de retorno médio é um pouco menor que 12%. Esse tipo de média é chamada de *média geométrica*.

Neste exemplo, a média geométrica é a raiz quinta do produto de cinco números. Ela é sempre a *enésima* raiz do produto de *n* números? Sim.

A base R não fornece uma função para calcular a média geométrica, mas ela é fácil de calcular.

Comece criando um vetor dos números:

```
invest <- c(1.10,1.15,1.10,1.20,1.05)
```

Eu usei a função prod() para calcular o produto dos números no vetor e a função length() para calcular quantos números existem no vetor. Então o cálculo é

```
> gm.invest <- prod(invest)^(1/(length(invest)))
> gm.invest
```

```
[1] 1.118847
```

Média harmônica

Veja uma situação com a qual às vezes nos deparamos na vida, mas com mais frequência em livros de Álgebra.

Suponha que você não esteja com pressa de chegar ao trabalho de manhã e dirige de sua casa até o trabalho a uma velocidade de 30 milhas por hora (50km/h). Mas, no fim do dia, você gostaria de chegar em casa mais rápido. Então, na viagem de volta (com exatamente a mesma distância), dirige do trabalho até sua casa a 50 milhas por hora (80km/h). Qual é a taxa média de seu tempo total na estrada?

Não é 40 milhas por hora (65km/h), porque você tem uma quantidade de tempo diferente na estrada para cada viagem. Sem entrar em detalhes, a fórmula para descobrir isso é

$$\frac{1}{\text{Média}} = \frac{1}{2}\left[\frac{1}{30} + \frac{1}{50}\right] = \frac{1}{37{,}5}$$

A média é 37,5. Esse tipo de média é chamada de *média harmônica*. Este exemplo consiste em dois números, mas é possível calculá-la para qualquer quantidade de números. É só colocar cada número no denominador de uma fração com 1 como o numerador. Os matemáticos chamam isso de o *recíproco* de um número. (Então $\frac{1}{30}$ é o recíproco de 30.) Some todos os recíprocos e calcule a média. O resultado é o recíproco da média harmônica.

A base R não tem uma função para a média harmônica, mas (novamente) ela é fácil de calcular. Comece criando um vetor das duas velocidades:

```
speeds <- c(30,50)
```

Calcular o recíproco do vetor resulta em um vetor de recíprocos:

```
> 1/speeds
[1] 0.03333333 0.02000000
```

Então a média harmônica é

```
> hm.speeds <- 1/mean(1/speeds)
> hm.speeds
[1] 37.5
```

Medianas: Preso no Meio

A média é uma maneira útil de resumir um grupo de números. Uma desvantagem ("a falha das médias") é que ela é sensível a valores extremos. Se houver um número maluco, a média também ficará doida. Quando isso acontece, ela pode não ser uma boa representante do grupo.

Aqui, por exemplo, estão as velocidades de leitura (em palavras por minuto) de um grupo de crianças:

56, 78, 45, 49, 55, 62

A média é

```
> reading.speeds <- c(56, 78, 45, 49, 55, 62)
> mean(reading.speeds)
[1] 57.5
```

Suponha que a criança que lê 78 palavras por minuto deixe o grupo e um leitor excepcionalmente rápido a substitui. Sua velocidade de leitura será de fenomenais 180 palavras por minuto:

```
> reading.speeds.new <- replace(reading.speeds,reading.speeds
  == 78,180)
> reading.speeds.new
[1] 56 180 45 49 55 62
```

Agora a média é

```
> mean(reading.speeds.new)
[1] 74.5
```

A nova média engana. Exceto pela criança nova, mais ninguém no grupo lê tão rápido. Em casos como esse, é uma boa ideia usar uma medida diferente de tendência central: a mediana.

Mediana é um nome chique para um conceito simples: é o valor do meio de um grupo de números. Organize os números em ordem e a mediana será os valores abaixo e acima dos quais metade das pontuações fica:

```
> sort(reading.speeds)
[1] 45 49 55 56 62 78
> sort(reading.speeds.new)
[1] 45 49 55 56 62 180
```

Em cada caso, a mediana está no meio do caminho entre 55 e 56, ou seja, 55,5.

Mediana em R: median()

Então não é um grande mistério usar R para encontrar a mediana:

```
> median(reading.speeds)
[1] 55.5
> median(reading.speeds.new)
[1] 55.5
```

Com conjuntos de dados maiores, pode-se encontrar replicações de pontuações. Em qualquer caso, a mediana ainda é o valor do meio. Por exemplo, aqui estão as potências para os carros de quatro cilindros em Cars93:

```
> with(Cars93, Horsepower.Four <- Horsepower[Cylinders == 4])
> sort(Horsepower.Four)
```

```
[1] 63 74 81 81 82 82 85 90 90 92 92 92 92 92
[15] 93 96 100 100 100 102 103 105 110 110 110 110 110 110
[29] 110 114 115 124 127 128 130 130 130 134 135 138 140 140
[43] 140 141 150 155 160 164 208
```

Podemos ver vários números duplicados aqui, particularmente perto do meio. Conte os valores e você verá que 24 pontuações são iguais ou menores que 110, e 24 pontuações são iguais ou maiores que 110, o que torna a mediana

```
> median(Horsepower.Four)
[1] 110
```

Estatística à Moda da Casa

Mais uma medida de tendência central, a *moda*, é importante. Ela é a pontuação que ocorre com mais frequência em grupos de pontuações.

Às vezes a moda é a melhor medida de tendência central para se usar. Imagine uma pequena empresa que consista em 30 consultores e dois executivos de alto escalão. Cada consultor tem um salário anual de US$40.000. Cada executivo tem um salário anual de US$250.000. O salário médio nessa empresa é de US$53.125.

A média lhe dá uma imagem clara da estrutura salarial da empresa? Se você estiver procurando um emprego nela, a média influenciaria suas expectativas? Você teria uma ideia melhor se considerasse a moda, que nesse caso é US$40.000 (a não ser que você seja um executivo valoroso e de talento!).

Não há nada complicado em encontrar a moda. Observe as pontuações e encontre a que ocorre com mais frequência e você terá encontrado a moda. Há duas pontuações empatadas? Nesse caso, seu conjunto de pontuações tem duas modas. (O termo técnico para isso é *bimodal*.)

É possível ter mais de duas modas? Com certeza.

Se cada pontuação ocorrer com a mesma frequência, não haverá moda.

Moda em R

A base R não fornece uma função para encontrar a moda. Ela tem uma função mode(), mas é para algo *muito* diferente. Então você precisa de um pacote chamado *modeest* em sua biblioteca. (Na aba Packages, selecione Install e na caixa de diálogo Install, digite **modeest** na caixa Packages e clique em Install. Depois marque sua caixa de verificação quando aparecer na aba Packages.)

Uma função no pacote modeest é a mfv() ("most frequent value" ou valor mais frequente, em português) e é dela que você precisa. Veja um vetor com duas modas (2 e 4):

```
> scores <- c(1,2,2,2,3,4,4,4,5,6)
> mfv(scores)
[1] 2 4
```

Capítulo **5**

NESTE CAPÍTULO

- **Descobrindo do que trata a variação**
- **Trabalhando com variância e desvio padrão**
- **Explorando funções R que calculam a variação**

Desviando da Média

Veja uma piada de estatístico bem conhecida: Três estatísticos saem para caçar cervos com arco e flecha. Eles veem um cervo e miram. Um atira e sua flecha voa três metros para a esquerda. O segundo atira e sua flecha voa três metros para a direita. O terceiro estatístico grita feliz: "Pegamos ele!"

Moral da história: calcular a média é uma ótima maneira de resumir um conjunto de números, mas a média pode enganá-lo. Como? Não dando todas as informações necessárias normalmente. Se depender da média, você pode deixar algo importante passar despercebido sobre o conjunto de números.

Para evitar perder informações importantes, é necessário outro tipo de estatística, aquela que mede a *variação*. Pense na variação como um tipo de média do quanto cada número em um grupo difere da média do grupo. Há várias estatísticas disponíveis para medir a variação. Todas elas funcionam do mesmo jeito: quanto maior o valor da estatística, mais os números diferem de sua média. Quanto menor o valor, menos eles diferem.

Medindo a Variação

Suponha que precisemos medir as alturas de um grupo de crianças e descobrimos que suas alturas (em polegadas) são

48, 48, 48, 48 e 48

Depois medimos outro grupo e descobrimos que suas alturas são

50, 47, 52, 46 e 45

Se calcularmos a média de cada grupo, descobriremos que são iguais: 48 polegadas (1,22m). Apenas olhando os números podemos ver que as alturas são diferentes. As alturas no primeiro grupo são todas iguais, enquanto as alturas do segundo grupo variam bastante.

Médias de desvios quadrados: Variância e como calculá-la

Uma maneira de mostrar a diferença entre os dois grupos é examinar os desvios de cada um. Pense em um "desvio" como a diferença entre uma pontuação e a média de todas as pontuações em um grupo.

Veja o que quero dizer. A Tabela 5-1 mostra o primeiro grupo de alturas e seus desvios.

TABELA 5-1 Primeiro Grupo de Alturas e Seus Desvios

Altura	Altura Média	Desvio
48	48-48	0
48	48-48	0
48	48-48	0
48	48-48	0
48	48-48	0

Uma maneira de proceder é calcular a média dos desvios. Claramente, a média dos números na coluna Desvio é zero.

A Tabela 5-2 mostra o segundo grupo de alturas e seus desvios.

TABELA 5-2 Segundo Grupo de Alturas e Seus Desvios

Altura	Altura Média	Desvio
50	50-48	2
47	47-48	–1
52	52-48	4
46	46-48	–2
45	45-48	–3

E a média dos desvios na Tabela 5-2? Ela é... zero!

E agora?

As médias dos desvios não nos ajudam a ver a diferença entre os dois grupos, porque a média dos desvios da média em qualquer grupo de números é *sempre* zero. Na verdade, os estatísticos experientes dirão que essa é uma propriedade definidora da média.

O coringa aqui são os números negativos. Como os estatísticos lidam com eles?

O truque é usar algo que você deve se lembrar da Álgebra: menos vezes menos é mais. Isso soa familiar?

Então... isso quer dizer que devemos multiplicar cada desvio por si mesmo e tirar a média dos resultados? Exatamente. Multiplicar um desvio por si mesmo é chamado de *elevar o desvio ao quadrado.* A média dos desvios quadrados é tão importante que tem um nome especial: *variância.*

A Tabela 5-3 mostra o grupo de alturas da Tabela 5-2, junto com seus desvios e desvios quadrados.

TABELA 5-3 Segundo Grupo de Alturas e Seus Desvios Quadrados

Altura	Altura Média	Desvio	Desvio Quadrado
50	50-48	2	4
47	47-48	–1	1
52	52-48	4	16
46	46-48	–2	4
45	45-48	–3	9

A variância, ou seja, a média dos desvios quadrados desse grupo, é (4 + 1 + 16 + 4 + 9) / 5 = 34 / 5 = 6,8. Isso, claro, é bem diferente do primeiro grupo, cuja variância é zero.

Para desenvolver a fórmula da variância e mostrar como ela funciona, utilizo símbolos. X representa o cabeçalho das Alturas na primeira coluna da tabela e $\bar{X}$ representa a média.

Um desvio é o resultado da subtração da média de cada número, então

$$\left(X - \bar{X}\right)$$

simboliza um desvio. E que tal multiplicar um desvio por ele mesmo? Isso seria

$$\left(X - \bar{X}\right)^2$$

Para calcular a variância, eleve cada desvio ao quadrado, some-os e encontre a média dos desvios quadrados. Se N representa a quantidade de desvios quadrados (neste exemplo, cinco), a fórmula para calcular a variância é

$$\frac{\sum\left(X - \bar{X}\right)^2}{N}$$

Σ é a letra grega sigma maiúscula e significa "a soma de".

Qual é o símbolo da variância? Como mencionei no Capítulo 1, as letras gregas representam os parâmetros da população e as letras latinas representam as estatísticas amostrais. Imagine que nosso pequeno grupo de cinco números seja uma população inteira. O alfabeto grego tem uma letra

que corresponde a *V*, do mesmo jeito que μ (o símbolo para a média da população) corresponde a *M*?

Não. Em vez disso, usamos o sigma *minúsculo*! Ele é assim: σ. E além disso, como estamos falando sobre quantidades quadradas, o símbolo para a variância da população é σ^2.

Resumindo: A fórmula para calcular a variância da população é

$$\sigma^2 = \frac{\sum\left(X - \bar{X}\right)^2}{N}$$

LEMBRE-SE

Um valor grande para a variância informa que os números em um grupo variam muito de sua média. Um valor pequeno para a variância, que os números são bem similares à sua média.

Variância amostral

A fórmula da variância que acabei de mostrar será adequada se o grupo de cinco medidas for uma população. Isso significa que a variância de uma amostra é diferente? Sim, e aqui está o porquê.

Se seu conjunto de números for uma amostra retirada de uma população grande, seu objetivo provavelmente será usar a variância da amostra para estimar a variância da população.

A fórmula na seção anterior não funciona como uma estimativa para a variância da população. Embora a média calculada da maneira usual seja uma estimativa precisa da média populacional, esse não é o caso da variância, por razões que vão muito além do escopo deste livro.

LEMBRE-SE

É bem fácil calcular uma estimativa precisa da variância populacional. Só precisamos usar *N-1* no denominador, em vez de *N*. (Repito, por razões que vão muito além do escopo deste livro.)

E como estamos trabalhando com uma característica de uma amostra (em vez da população), usamos o equivalente em português da letra grega — *s*, em vez de σ. Isso significa que a fórmula para a variância amostral (como uma estimativa da variância populacional) é

$$s^2 = \frac{\sum\left(X - \bar{X}\right)^2}{N - 1}$$

O valor de s^2, dados os desvios quadrados no conjunto de cinco números, é

$$(4 + 1 + 16 + 4 + 9) / 4 = 34 / 4 = 8{,}5$$

Então, se esses números

50, 47, 52, 46 e 45

forem uma população inteira, sua variância será 6,8. Se forem uma amostra retirada de uma população maior, a melhor estimativa para a variância populacional será 8,5.

Variância em R

Calcular a variância em R é bem simples. Use a função var(). Mas qual variância ela nos dá? Com *N* ou *N*-1 no denominador? Vamos descobrir:

```
> heights <- c(50, 47, 52, 46, 45)
> var(heights)
[1] 8.5
```

Ela calcula a variância estimada (com *N*–1 no denominador). Para calcular a primeira variância que mostrei (com *N* no denominador), é preciso multiplicar esse número por (*N*–1)/*N*. Usando length() para calcular *N*, isso seria

```
> var(heights)*(length(heights)-1)/length(heights)
[1] 6.8
```

Se eu trabalhasse com esse tipo de variância com frequência, definiria uma função var.p():

```
var.p = function(x){var(x)*(length(x)-1)/length(x)}
```

Veja como usá-la:

```
> var.p(heights)
[1] 6.8
```

LEMBRE-SE

Por razões que ficarão claras mais tarde, eu gostaria que você pensasse no denominador de uma estimativa de variância (como *N*–1) como um *grau de liberdade*. Por quê? Fique ligado. (O Capítulo 12 revelará tudo!)

De Volta às Raízes: Desvio-padrão

Depois de calcular a variância de um conjunto de números, temos um valor cujas unidades são diferentes de suas medidas originais. Por exemplo, se suas medidas originais são em polegadas, sua variância será em polegadas *quadradas*. Isso porque elevamos os desvios ao quadrado antes de calcular a média. Então a variância na população de cinco pontuações no exemplo anterior é 6,8 polegadas quadradas (17,27cm^2).

Pode ser difícil compreender o que isso significa. Muitas vezes será mais claro se a variação estatística estiver nas mesmas unidades das medidas originais. É fácil transformar a variância nesse tipo de estatística. É só calcular sua raiz quadrada.

Como a variância, essa raiz quadrada é tão importante que tem um nome especial: desvio-padrão.

Desvio-padrão populacional

O *desvio-padrão* de uma população é a raiz quadrada da variância populacional. O símbolo para o desvio-padrão populacional é σ (sigma). Sua fórmula é

$$\sigma = \sqrt{\sigma^2} = \sqrt{\frac{\sum\left(X - \bar{X}\right)^2}{N}}$$

Para essa população com cinco pontuações de medidas (em polegadas):

50, 47, 52, 46 e 45

a variância populacional é 6,8 polegadas quadradas (17,27cm^2) e o desvio-padrão populacional é 2,61 polegadas (6,63cm) (arredondado).

Desvio-padrão amostral

O desvio-padrão de uma amostra, uma estimativa do desvio-padrão de uma população, é a raiz quadrada da variância amostral. Seu símbolo é *s* e sua fórmula é

$$s = \sqrt{s^2} = \sqrt{\frac{\sum\left(X - \bar{X}\right)^2}{N - 1}}$$

Para esta amostra de medidas (em polegadas):

50, 47, 52, 46 e 45

a variância populacional estimada é 8,4 polegadas quadradas (21,34cm^2) e o desvio-padrão populacional estimado é 2,92 polegadas (7,42cm) (arredondad0).

Desvio-padrão em R

Como no caso da variância, é fácil usar R para calcular o desvio-padrão: use a função sd(). E como o equivalente da variância, sd() calcula *s*, não σ:

```
> sd(heights)
[1] 2.915476
```

Para σ, em outras palavras, tratando os cinco números como uma população autônoma, é preciso multiplicar o resultado de sd() pela raiz quadrada de (*N*-1)/*N*:

```
> sd(heights)*(sqrt((length(heights)-1)/length(heights)))
[1] 2.607681
```

Novamente, se formos usar isso com frequência, será uma boa ideia definir uma função:

```
sd.p=function(x){sd(x)*sqrt((length(x)-1)/length(x))}
```

Veja como usar essa função:

```
> sd.p(heights)
[1] 2.607681
```

Condições, Condições, Condições...

No Capítulo 4, eu indico que, com data frames maiores, às vezes queremos calcular estatísticas em casos (linhas) que satisfazem certas condições, em vez de em todos os casos.

Como nos Capítulos 3 e 4, uso o data frame Cars93 para a análise que vem a seguir. Esse data frame tem dados de uma amostra de 93 carros de 1993. Você o encontra no pacote MASS, então verifique se o pacote existe em sua biblioteca. (Encontre MASS na aba Packages e clique em sua caixa de verificação.)

Eu calculo a variância das potências dos carros originados nos EUA. Usando a função with() que mostrei no Capítulo 4, isso seria

```
> with(Cars93, var(Horsepower[Origin == "USA"]))
[1] 2965.319
```

Quantos desses carros estão nesse grupo?

```
> with(Cars93, length(Horsepower[Origin == "USA"]))
[1] 48
```

E os carros que não são dos EUA?

```
> with(Cars93, var(Horsepower[Origin == "non-USA"]))
[1] 2537.283
> with(Cars93, length(Horsepower[Origin == "non-USA"]))
[1] 45
```

É possível comparar essas variâncias? Claro, mas só no Capítulo 11.

Deixarei isso como um exercício para que você calcule os desvios-padrão para os carros norte-americanos e de outros países.