

Rafael A. Irizarry

Introdução à Ciência de Dados

Análise de Dados e Algoritmos de Previsão com R

Contents

List of Tables	9
List of Figures	11
Prefácio	13
Agradecimentos	15
Introdução	17
1 Introdução ao R e RStudio	21
1.1 Por que R?	21
1.2 O console do R	22
1.3 <i>Scripts</i>	22
1.4 RStudio	23
1.4.1 Painéis	23
1.4.2 Combinações de teclas (atalhos)	25
1.4.3 Como executar comandos enquanto edita <i>scripts</i>	26
1.4.4 Como alterar opções globais	28
1.5 Instalando pacotes R	29
I R	31
2 R básico	33
2.1 Estudo de caso: assassinatos por armas nos EUA	33
2.2 Os princípios básicos	34
2.2.1 Objetos	34
2.2.2 A área de trabalho (<i>workspace</i>)	35
2.2.3 Funções	36
2.2.4 Outros objetos pré-construídos	37
2.2.5 Nomes de variáveis	38
2.2.6 Como salvar sua área de trabalho	38
2.2.7 Fundamentando o uso de <i>scripts</i>	39
2.2.8 Como comentar seu código	39
2.3 Exercícios	39
2.4 Tipos de dados	40
2.4.1 <i>Data frames</i>	40
2.4.2 Examinando um objeto	41
2.4.3 O operador de acesso: \$	41
2.4.4 Vetores: numéricos, caracteres e lógicos	42
2.4.5 Fatores (<i>factors</i>)	43
2.4.6 Listas	43
2.4.7 Matrizes	44

2.5	Exercícios	46
2.6	Vetores	47
2.6.1	Criando vetores	47
2.6.2	Nomes	47
2.6.3	Sequências	48
2.6.4	Criando subconjuntos	49
2.7	Conversão forçada	49
2.7.1	Valores NA	50
2.8	Exercícios	50
2.9	Ordenação	51
2.9.1	<code>sort</code>	51
2.9.2	<code>order</code>	52
2.9.3	<code>max</code> e <code>which.max</code>	53
2.9.4	<code>rank</code>	53
2.9.5	Cuidado com a reciclagem	53
2.10	Exercícios	54
2.11	Aritmética vetorial	55
2.11.1	Reescalonando um vetor	55
2.11.2	Dois vetores	55
2.12	Exercícios	56
2.13	Indexação	56
2.13.1	Criando subconjuntos com objetos do tipo lógico	57
2.13.2	Operadores lógicos	57
2.13.3	<code>which</code>	58
2.13.4	<code>match</code>	58
2.13.5	<code>%in%</code>	58
2.14	Exercícios	59
2.15	Gráficos básicos	59
2.15.1	<code>plot</code>	59
2.15.2	<code>hist</code>	60
2.15.3	<code>boxplot</code>	61
2.15.4	<code>image</code>	61
2.16	Exercícios	62
3	Conceitos básicos de programação	63
3.1	Expressões condicionais	63
3.2	Como definir funções	65
3.3	<i>Namespaces</i>	66
3.4	Laços do tipo <code>for</code>	67
3.5	Vectorização e funcionais	68
3.6	Exercícios	69
4	<i>tidyverse</i>	71
4.1	Data <code>tidy</code>	71
4.2	Exercícios	72
4.3	Como manipular <i>data frames</i>	73
4.3.1	Como adicionar uma coluna com <code>mutate</code>	73
4.3.2	Como criar subconjuntos com <code>filter</code>	74
4.3.3	Como selecionar colunas com <code>select</code>	74
4.4	Exercícios	74
4.5	O cano <code>_>%</code>	76

<i>0.0 Contents</i>	5
4.6 Exercícios	77
4.7 Como resumir dados	77
4.7.1 <code>summarize</code>	78
4.7.2 <code>pull</code>	79
4.7.3 Como agrupar e resumir com <code>group_by</code>	80
4.8 Como encomendar os <i>data frames</i>	81
4.8.1 Como encomendar aninhado	82
4.8.2 Os primeiros <i>n</i>	82
4.9 Exercícios	83
4.10 <i>Tibbles</i>	84
4.10.1 <i>Tibbles</i> parece melhor	84
4.10.2 <i>Tibbles_subconjuntos são tibbles</i>	85
4.10.3 <i>Tibbles</i> pode ter entradas complexas	85
4.10.4 <i>Tibbles</i> podem ser agrupados	85
4.10.5 Como criar um <i>tibble</i> usando <code>tibble</code> ao invés de <code>data.frame</code>	86
4.11 O operador de ponto	86
4.12 <code>do</code>	87
4.13 O pacote <code>purrr</code>	88
4.14 Os condicionais <i>tidyverse</i>	89
4.14.1 <code>case_when</code>	90
4.14.2 <code>between</code>	90
4.15 Exercícios	90
5 Importando dados	93
5.1 Caminhos e diretório de trabalho	94
5.1.1 Sistema de arquivos	95
5.1.2 Caminhos relativos e absolutos	95
5.1.3 O diretório de trabalho	95
5.1.4 Gerando nomes de caminhos	96
5.1.5 Copiando arquivos usando caminhos	96
5.2 Os pacotes <code>readr</code> e <code>readxl</code>	97
5.2.1 <code>readr</code>	97
5.2.2 <code>readxl</code>	98
5.3 Exercícios	98
5.4 Como fazer <i>download</i> de arquivos	99
5.5 Funções básicas de importação do R	99
5.5.1 <code>scan</code>	100
5.6 Arquivos de texto versus arquivos binários	100
5.7 Unicode versus ASCII	101
5.8 Como organizar dados com planilhas	102
5.9 Exercícios	103
II Exibição de dados	105
6 Introdução à visualização de dados	107
7 <code>ggplot2</code>	111
7.1 Os componentes de um gráfico	112
7.2 Objetos <code>ggplot</code>	113
7.3 Geometrias	114
7.4 Mapeamentos estéticos	115
7.5 Camadas	116

7.5.1	Trabalhando com argumentos	116
7.6	Mapeamento estético global versus local	117
7.7	Escalas	118
7.8	Rótulos e títulos	119
7.9	Usando categorias como cores	120
7.10	Anotações, formas e ajustes	122
7.11	Pacotes complementares	123
7.12	Como combinar tudo	123
7.13	Gráficos rápidos com <code>qplot</code>	124
7.14	Grade de gráficos	125
7.15	Exercícios	125
8	Visualizando distribuições de dados	129
8.1	Tipos de variáveis	129
8.2	Estudo de caso: descrevendo a altura dos alunos	130
8.3	Função de distribuição	130
8.4	Funções de distribuição acumulada	131
8.5	Histogramas	132
8.6	Curvas de densidade	133
8.6.1	Interpretando o eixo y	137
8.6.2	Densidades permitem a estratificação	138
8.7	Exercícios	138
8.8	Distribuição normal	142
8.9	Unidades padrão	144
8.10	Gráficos QQ	145
8.11	Percentis	147
8.12	Diagramas de caixas (<i>boxplots</i>)	147
8.13	Estratificação	148
8.14	Estudo de caso: descrevendo as alturas dos alunos (continuação)	149
8.15	Exercícios	150
8.16	Geometrias <code>ggplot2</code>	152
8.16.1	Gráficos de barra	152
8.16.2	Histogramas	154
8.16.3	Gráficos de densidade	154
8.16.4	<i>Boxplots</i>	155
8.16.5	Gráficos QQ	156
8.16.6	Imagens	157
8.16.7	Gráficos rápidos	158
8.17	Exercícios	159
9	Visualização de dados na prática	161
9.1	Estudo de caso: novas perspectivas sobre pobreza	161
9.1.1	Teste de Hans Rosling	162
9.2	Diagramas de dispersão (<i>scatterplots</i>)	163
9.3	Separe em facetas	164
9.3.1	<code>facet_wrap</code>	166
9.3.2	Escalas fixas para melhores comparações	167
9.4	Gráficos de séries temporais	168
9.4.1	Rótulos em vez de legendas	171
9.5	Transformações de dados	172
9.5.1	Transformação logarítmica	172

9.5.2 Qual base?	174
9.5.3 Transformar os valores ou a escala?	175
9.6 Como visualizar distribuições multimodais	176
9.7 Como comparar múltiplas distribuições com <i>boxplots</i> e gráficos <i>ridge</i>	176
9.7.1 <i>Boxplots</i> (diagramas de caixa)	177
9.7.2 Gráficos <i>ridge</i>	178
9.7.3 Exemplo: distribuições de renda 1970 versus 2010	180
9.7.4 Como acessar variáveis calculadas	185
9.7.5 Densidades ponderadas	188
9.8 A falácia ecológica e a importância de mostrar os dados	188
9.8.1 Transformação logística	189
9.8.2 Mostre os dados	189
10 Princípios de visualização de dados	191
10.1 Codificando dados usando dicas visuais	191
10.2 Saiba quando incluir 0	194
10.3 Não distorça quantidades	197
10.4 Ordenando categorias por um valor significativo	198
10.5 Mostre os dados	200
10.6 Facilite comparações	205
10.6.1 Use eixos em comum	205
10.6.2 Alinhe os gráficos verticalmente para ver alterações horizontais e horizontalmente para ver alterações verticais	205
10.6.3 Considere transformações	207
10.6.4 As indicações visuais comparadas devem ser adjacentes	209
10.6.5 Use cores	209
10.7 Pense no daltônico	210
10.8 Plotagens para duas variáveis	210
10.8.1 Gráficos de inclinação	211
10.8.2 Gráfico de Bland-Altman	212
10.9 Codificando uma terceira variável	213
10.10 Evite gráficos pseudo-tridimensionais	215
10.11 Evite muitos dígitos significativos	216
10.12 Conheça seu público	217
10.13 Exercícios	217
10.14 Estudo de caso: vacinas e doenças infecciosas	221
10.15 Exercícios	225
11 Resumos robustos	227
11.1 Valores atípicos	227
11.2 Mediana	228
11.3 O intervalo interquartil (IQR)	228
11.4 Definição de Tukey de um outlier	229
11.5 Desvio absoluto mediano	230
11.6 Exercícios	230
11.7 Estudo de caso: altura do aluno autorreferida	231
III Estatísticas com R	235
12 Introdução à estatística com R	237
13 Probabilidade	239

13.1 Probabilidade discreta	239
13.1.1 Frequência relativa	239
13.1.2 Notação	240
13.1.3 Distribuições de probabilidade	240
13.1.4 Defina a semente aleatória	241
13.1.5 Com e sem substituição	242
13.2 Independência	243
13.3 Probabilidades condicionais	243
13.4 Regras de adição e multiplicação	244
13.5 Combinações e permutações	245
13.6 Exemplos	249
13.6.1 Problema de aniversário	250
13.7 Exercícios	253
13.8 Probabilidade contínua	255
13.9 Distribuições teóricas contínuas	256
13.9.1 Distribuições teóricas como aproximações	256
13.9.2 A densidade de probabilidade	257
13.10 Distribuições contínuas	260
13.11 Exercícios	260
14 Variáveis aleatórias	263
14.1 Variáveis aleatórias	263
14.2 Modelos de amostragem	264
14.3 A distribuição de probabilidade de uma variável aleatória	265
14.4 Distribuições versus distribuições de probabilidade	267
14.5 Notação para variáveis aleatórias	267
14.6 O valor esperado e o erro padrão	268
14.6.1 Desvio padrão da população versus desvio padrão da amostra	270
14.7 Teorema do Limite Central	271
14.8 Propriedades estatísticas das médias	272
14.9 Lei dos grandes números	274
14.9.1 Interpretação incorreta da lei das médias	274
14.10 Exercícios	274
14.11 Estudo de caso: <i>The Big Short</i>	275
14.11.1 Taxas de juros explicadas com modelo de probabilidade	275
14.11.2 <i>The Big Short</i>	278

List of Tables

List of Figures

Prefácio

v1

Este livro começou como as notas de aulas empregadas para ministrar o curso na HarvardX Data Science Series¹.

Os códigos em Rmarkdown usados para gerar o livro estão disponíveis no [GitHub²](#). Observe que o tema usado para plotagem de gráficos em todo o livro pode ser recriado usando a função `ds_theme_set()` do pacote `dslabs`.

Um PDF gratuito, da versão de 24 de outubro de 2019, em inglês, deste livro está disponível no [Leanpub³](#).

Uma cópia impressa da versão em inglês deste livro está disponível em [CRC Press⁴](#).

Este trabalho foi publicado sob a licença Creative Commons Atribuição-NãoComercial-CompartilhaIgual 4.0 Internacional [CC BY-NC-SA 4.0](#).

Você pode obter mais informações relacionadas ao livro no Twitter. Para atualizações, siga [@rafalab](#).

¹<https://www.edx.org/professional-certificate/harvardx-data-science>

²<https://github.com/rafalab/dsbook>

³<https://leanpub.com/datasciencobook>

⁴<https://www.crcpress.com/Introduction-to-Data-Science-Data-Analysis-and-Prediction-Algorithms-with/Irizarry/p/book/9780367357986>

Agradecimentos

Este livro é dedicado a todos os envolvidos no desenvolvimento e manutenção do R e dos pacotes que utilizamos no texto. Um agradecimento especial aos desenvolvedores e mantenedores dos pacotes *base*, *tidyverse* e *caret*.

Um agradecimento especial ao meu guru em *tidyverse* David Robinson e, também, a Amy Gill por dezenas de comentários, alterações e sugestões. Além disso, muito obrigado a Stephanie Hicks, que serviu duas vezes como co-instrutora em minhas aulas de ciência de dados, e a Yihui Xie, que, pacientemente, tolerou minhas múltiplas perguntas sobre *bookdown*. Agradeço também a Karl Broman, de quem peguei idéias emprestadas para as seções de Visualização de Dados e Ferramentas de Produtividade, e a Hector Corrada-Bravo, por seus conselhos sobre como melhor ensinar aprendizado de máquina. Agradeço a Peter Aldhous, de quem peguei idéias emprestadas para a seção sobre princípios de visualização de dados, e Jenny Bryan por escrever *Happy Git and GitHub for the useR*, que influenciou nossos capítulos do Git. Agradeço a Alyssa Frazee por ajudar a criar o problema para lição de casa que se tornou o capítulo de Sistemas de Recomendação e a Amanda Cox por fornecer os dados dos exames padronizados de Nova Iorque (*New York Regents exams*). Além disso, muito obrigado a Jeff Leek, Roger Peng e Brian Caffo, que ministraram as aulas que inspiraram a forma como este livro é dividido, e Garrett Grolemund e Hadley Wickham por compartilharem o código utilizado no livro *R for Data Science*. Finalmente, obrigado a Alex Nones por corrigir o texto durante suas várias etapas.

Este livro foi concebido durante o ensino de vários cursos de estatística aplicada, iniciados há mais de quinze anos. Os monitores que trabalharam comigo ao longo dos anos fizeram importantes contribuições indiretas para este livro. A versão mais recente deste curso é uma série HarvardX coordenada por Heather Sternshein e Zofia Gajdos, a quem agradecemos por suas contribuições. Também agradecemos a todos os alunos cujas perguntas e comentários nos ajudaram a melhorar o livro. Os cursos foram parcialmente financiados pelo subsídio NIH R25GM114818. Agradecemos aos Institutos Nacionais de Saúde (*National Institutes of Health*) por seu apoio.

Um agradecimento especial a todos aqueles que editaram o livro através de *pull requests* no GitHub ou fizeram sugestões criando um *issue* ou enviando um email: **nickyfoto** (Huang Qiang) **desautm** (Marc-André Désautels), **michaschwab** (Michail Schwab) **alvarolarreategui** (Alvaro Larreategui), **jakevc** (Jake VanCampen), **omerta** (Guillermo Lengemann), **espinielli** (Enrico Spinielli), **asimumba** (Aaron Simumba) **braunschweig** (Maldewar), **gwierzchowski** (Grzegorz Wierzchowski), **technocrat** (Richard Careaga) **atzakas**, **defeit** (David Emerson Feit), **shiraamitchell** (Shira Mitchell) **Nathalie-S**, **andreashandel** (Andreas Handel) **berkowitz** (Elias Berkowitz) **Dean-Webb** (Dean Webber), **mohayusuf**, **jimrothstein**, **mploenzke** (Matthew Ploenzke), **NicholasDowand** (Nicholas Dow) **kant** (Darío Hereñú), **debbieyuster** (Debbie Yuster), **tuanchauict** (Tuan Chau), **phzeller**, David D. Kane, El Mustapha El Abbassi e Vadim Zipunnikov.

Este livro foi traduzido para o português por Benilton S Carvalho, Diego Mariano, Guilherme

Ludwig, Larissa Ávila, Pedro Baldoni, Samara F Kiihl e Tatiana Benaglia. Agradecemos a todos que contribuíram para esta tradução.

Ilia Ushkin e Dustin Tingley geraram um primeiro rascunho usando um programa de tradução automática.

Introdução

A demanda por profissionais qualificados em ciência de dados na indústria, academia e governo está crescendo rapidamente. Este livro apresenta conceitos e habilidades que podem ajudá-lo a enfrentar os desafios da análise de dados em situações reais. O texto aborda os conceitos de probabilidade, inferência estatística, regressão linear e aprendizado de máquina. Isso também ajudará no desenvolvimento de habilidades como programação em R, preparação de dados (*data wrangling*), **dplyr**, visualização de dados com **ggplot2**, criação de algoritmos com **caret**, organização de arquivos em ambiente UNIX/Linux, controle de versão com Git e GitHub e preparação de documentos reprodutíveis com **knitr** e **Rmarkdown**. O livro está dividido em seis partes: **R**, **Visualização de dados**, **Wrangling_of data**, **Estatísticas com R**, **Aprendizado de Máquina** e **Ferramentas de Produtividade**. Cada parte possui vários capítulos que devem ser apresentados em uma única aula e inclui dezenas de exercícios distribuídos pelos capítulos.

Estudos de Caso

Ao longo do livro, utilizamos estudos de caso motivadores. Em cada estudo de caso, tentamos imitar realisticamente a experiência dos cientistas de dados. Para cada um dos conceitos que discutimos, começamos fazendo perguntas específicas, que, depois, respondemos através da análise de dados. Aprendemos conceitos como um meio de responder perguntas. Exemplos dos estudos de caso que incluímos neste livro são:

Estudo de caso	Conceito
Taxas estaduais de homicídio nos Estados Unidos	Conceitos básicos de R
Alturas dos estudantes	Resumos estatísticos
Tendências em saúde e economia global	Visualização de dados
O impacto das vacinas nas taxas de doenças infecciosas	Visualização de dados
A crise financeira de 2007-2008	Probabilidade
Previsão eleitoral	Inferência estatística
Alturas autorreferidas de estudantes	<i>Arranjo</i> de dados
<i>Money Ball</i> : Construindo um time de beisebol	Regressão linear
MNIST: Processamento de imagem com dígitos manuscritos	<i>Aprendizado de Máquina</i>

Estudo de caso	Conceito
Sistemas de recomendação de filmes	<i>Aprendizado de Máquina</i>

Quem achará este livro útil?

O objetivo deste livro é servir como texto para um primeiro curso de ciência de dados. Nenhum conhecimento prévio de R é necessário, embora alguma experiência em programação possa ser útil. Os conceitos estatísticos usados para responder às perguntas dos estudos de caso são apresentados apenas brevemente e, portanto, recomendamos um livro de probabilidade e estatística para quem deseja entender completamente esses conceitos. Ao ler e entender todos os capítulos e concluir todos os exercícios, os alunos estarão bem posicionados para concluir tarefas básicas de análise de dados e aprender os conceitos e habilidades mais avançados necessários para se tornarem especialistas.

O que este livro cobre?

Começamos com o **R básico** e o **tidyverse**. Você aprenderá R ao longo do livro, mas na primeira parte cobriremos os componentes básicos necessários para que você continue aprendendo.

A crescente disponibilidade de conjuntos de dados informativos e ferramentas de software levaram a uma maior dependência da **visualização de dados** em muitos campos. Na segunda parte, demonstramos como usar o **ggplot2** para gerar gráficos e descrever princípios importantes da visualização de dados.

Na terceira parte, demonstramos a importância da estatística na análise dos dados, respondendo a perguntas de estudos de caso usando **probabilidade**, **inferência** e **regressão** com R.

A quarta parte usa vários exemplos para familiarizar o leitor com a **preparação de dados** (*data wrangling*). As habilidades específicas que estudamos incluem raspagem da Web, usando expressões regulares e mesclando e remodelando tabelas de dados. Fazemos isso usando as ferramentas **tidyverse**.

Na quinta parte, apresentamos vários desafios que nos levam a introduzir o **aprendizado de máquina**. Aprenderemos a usar o pacote **caret** para criar algoritmos de previsão que incluem *K-vizinhos mais próximos* e *florestas aleatórias*.

Na parte final, oferecemos uma breve introdução às **ferramentas de produtividade** que usamos diariamente em projetos de ciência de dados. Estes são RStudio, ambiente UNIX/Linux, Git e GitHub e **knitr** e RMarkdown.

O que este livro não cobre?

Este livro é focado nos aspectos da análise de dados que compõem a Ciência de Dados. Portanto, não discutimos aspectos relacionados ao gerenciamento ou engenharia de dados. Embora a programação em R seja uma parte essencial do livro, não ensinamos tópicos de computação mais avançados, como estruturas de dados, otimização e teoria de algoritmos. Da mesma forma, não discutimos tópicos como serviços da web, gráficos interativos, computação paralela e processamento de *streaming* de dados. Os conceitos estatísticos são apresentados principalmente como ferramentas para a solução de problemas e nenhuma descrição teórica detalhada é incluída neste livro.

1

Introdução ao R e RStudio

1.1 Por que R?

R não é uma linguagem de programação como C ou Java. O R não foi criado por engenheiros de *software* para desenvolvimento de *software*. Ao invés disso, foi desenvolvido por estatísticos como um ambiente interativo para análise de dados. Você pode ler a história completa no artigo *A Brief History of S* (Uma breve história de S)¹. A interatividade é um recurso indispensável na ciência de dados porque, como você aprenderá em breve, a capacidade de explorar rapidamente os dados é necessária para o sucesso neste campo. No entanto, assim como em outras linguagens de programação, no R, você pode gravar o seu trabalho como *scripts*, que podem ser facilmente executados a qualquer momento. Esses *scripts* servem como um registro da análise que você realizou, uma peça-chave que facilita o trabalho reproduzível. Se você for um programador profissional, não espere que o R siga as convenções às quais está acostumado, pois você ficará desapontado. Se você for paciente, apreciará a grande vantagem do R quando se trata de análise e, principalmente, visualização de dados.

Outras características atraentes do R são:

1. R é gratuito e de código aberto².
2. Ele roda em todas as principais plataformas: Windows, Mac Os, UNIX/Linux.
3. *Scripts* e objetos de dados podem ser compartilhados diretamente entre plataformas.
4. Existe uma grande, crescente e ativa comunidade de usuários de R e, consequentemente, inúmeros recursos para aprender e buscar informações^{3 4 5}.
5. É fácil para que outros contribuam com plug-ins que permitem que desenvolvedores compartilhem implementações de *software* com novas metodologias para ciência de dados. Isso dá aos usuários do R acesso antecipado aos métodos e ferramentas mais recentes desenvolvidos para uma ampla variedade de disciplinas, incluindo ecologia, biologia molecular, ciências sociais e geografia, entre outros campos.

¹<https://pdfs.semanticscholar.org/9b48/46f192aa37ca122cfabb1ed1b59866d8bfda.pdf>

²<https://opensource.org/history>

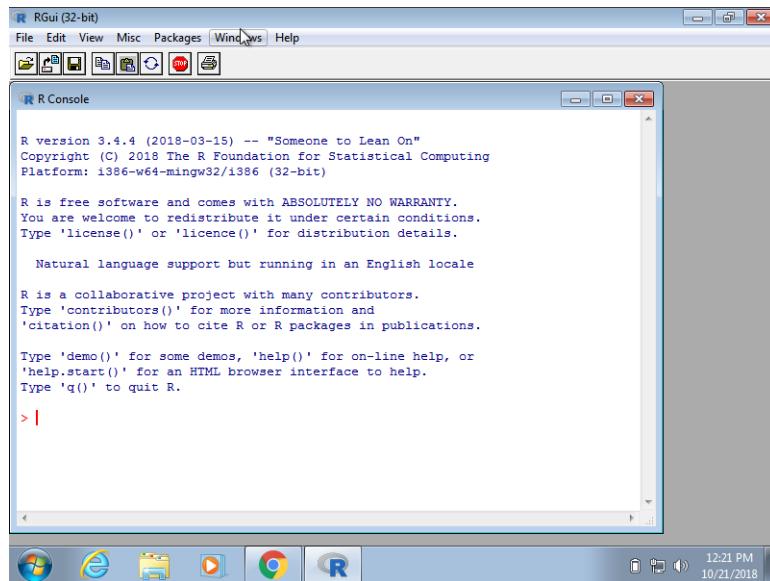
³<https://stats.stackexchange.com/questions/138/free-resources-for-learning-r>

⁴<https://www.r-project.org/help.html>

⁵<https://stackoverflow.com/documentation/r/topics>

1.2 O console do R

A análise de dados interativa geralmente ocorre no console do R que executa comandos à medida que são digitados. Existem várias maneiras de obter acesso a um console R. Uma forma é simplesmente iniciar o R no seu computador. O console fica assim:



Como um exemplo rápido, tente usar o console para calcular uma gorjeta de 15% em uma refeição que custa \$19,71:

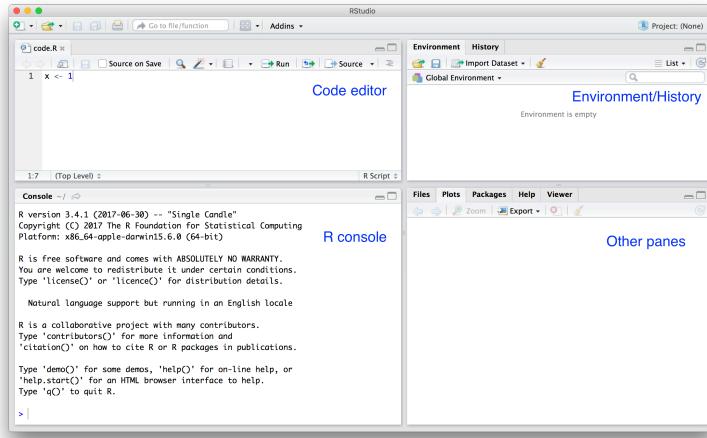
```
0.15 * 19.71
#> [1] 2.96
```

Nota: neste livro, as caixas cinza são usadas para mostrar o código R escrito no console R. O símbolo #> é usado para denotar a saída do console R.

1.3 Scripts

Uma das grandes vantagens do R, quando comparado a *softwares* de interface gráfica do tipo *point and click*, é que você pode salvar seu trabalho como *scripts*, que podem ser editados e salvos com um editor de texto. O material deste livro foi desenvolvido usando o *Integrated Development Environment* (IDE) do RStudio⁶. O RStudio inclui um editor com muitos recursos específicos de R, um console para executar seu código e outros painéis úteis, incluindo um para exibir figuras.

⁶<https://www.rstudio.com/>



A maioria dos consoles R disponíveis na Web também inclui um painel para editar *scripts*, mas nem todos permitem salvar *scripts* para uso posterior.

Todos os *scripts* R usados para gerar este livro podem ser encontrados no GitHub⁷.

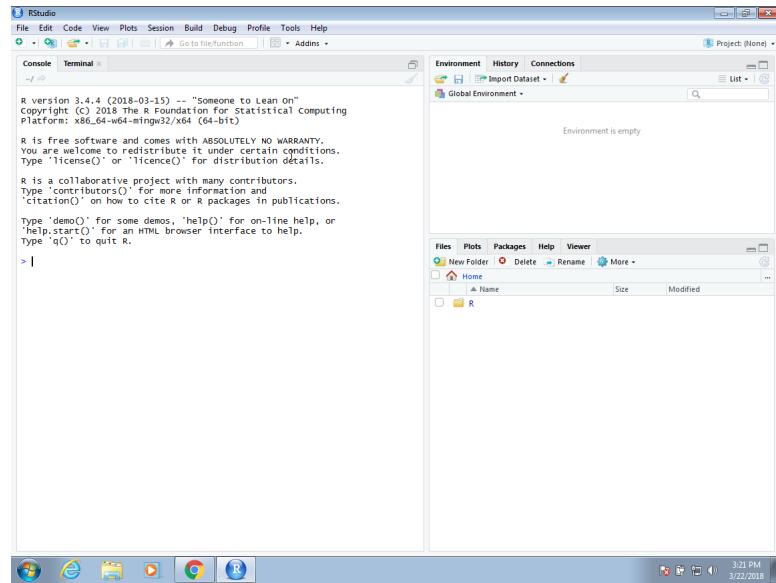
1.4 RStudio

O RStudio será nossa plataforma de desenvolvimento para projetos de ciência de dados. Ele não apenas nos fornece um editor para criar e editar nossos *scripts*, como também oferece muitas outras ferramentas úteis. Nesta seção, abordaremos alguns dos princípios básicos do RStudio.

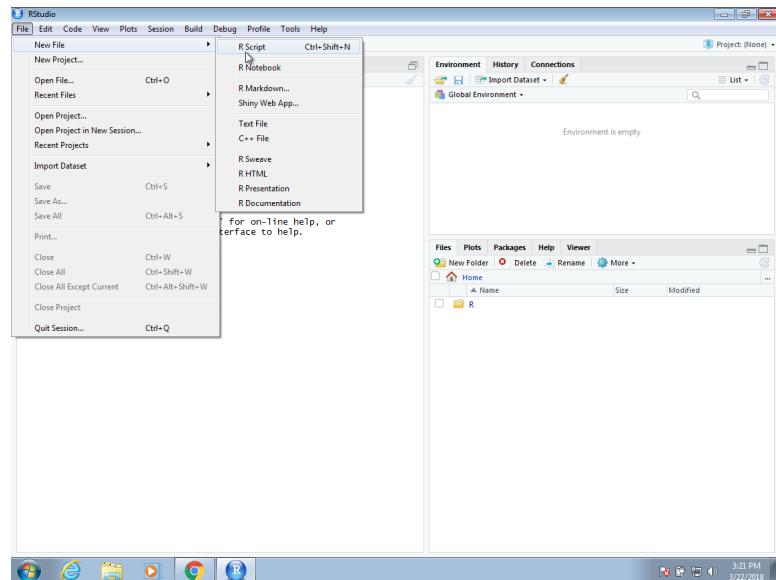
1.4.1 Painéis

Quando iniciar o RStudio pela primeira vez, você verá três painéis. O painel esquerdo mostra o console do R. À direita, o painel superior inclui guias como *Environment* (ambiente) e *History* (histórico), enquanto o painel inferior mostra cinco guias: *Files* (arquivos), *Plots* (gráficos), *Packages* (pacotes), *Help* (ajuda) e *Viewer* (visualizador). Essas guias podem ser diferentes nas novas versões do RStudio. Você pode clicar em cada guia para percorrer as diferentes opções.

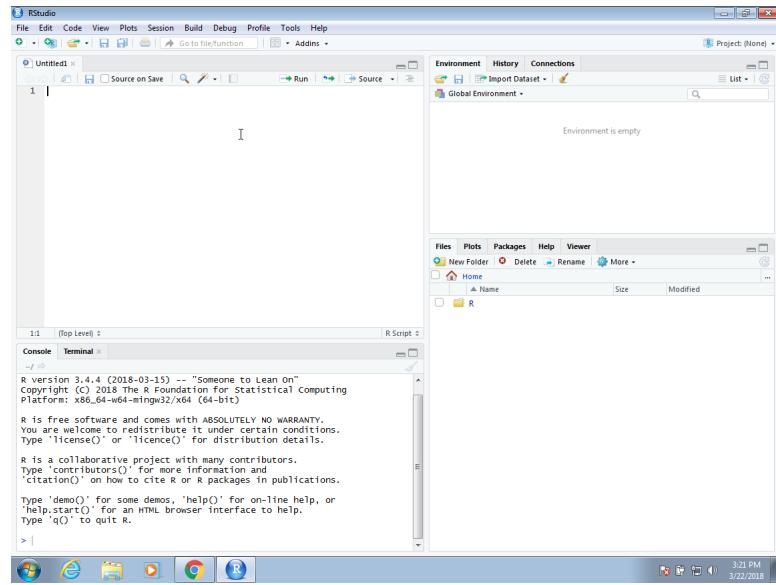
⁷<https://github.com/rafaelab/dslivro>



Para iniciar um novo *script*, clique em *File*, depois em *New File* e, em seguida, em *R Script*.



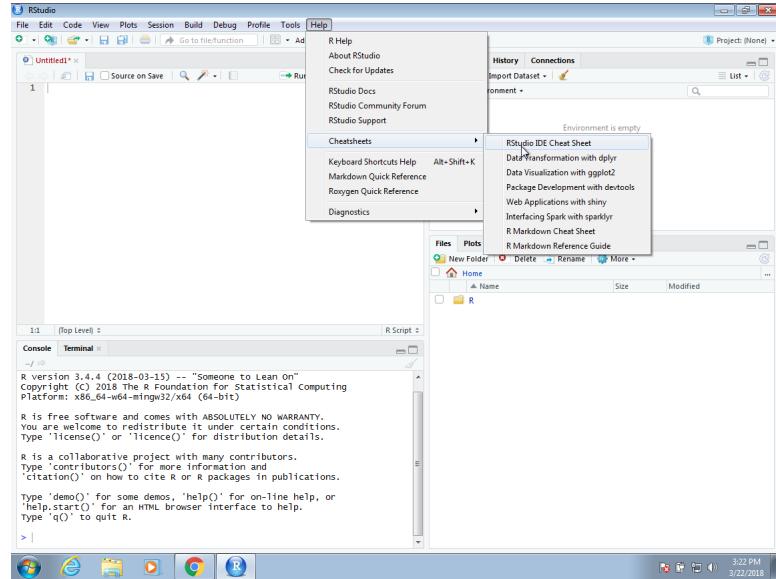
Isso inicia um novo painel à esquerda e é aqui que você pode começar a digitar seu *script*.



1.4.2 Combinações de teclas (atalhos)

Muitas das tarefas que realizamos com o *mouse* podem ser realizadas com uma combinação de teclas (*key bindings* em inglês). Por exemplo, acabamos de mostrar como usar o *mouse* para iniciar um novo *script*, mas você também pode usar a seguinte combinação de teclas: Ctrl + Shift + N no Windows e Command + Shift + N no Mac.

Embora muitas vezes demonstremos como usar o mouse neste tutorial, **recomendamos fortemente que você memorize combinações de teclas para as operações que use com mais frequência**. O RStudio inclui uma folha de dicas (*cheatsheets*, em inglês) com os comandos mais usados. Você pode obtê-lo diretamente do RStudio assim:



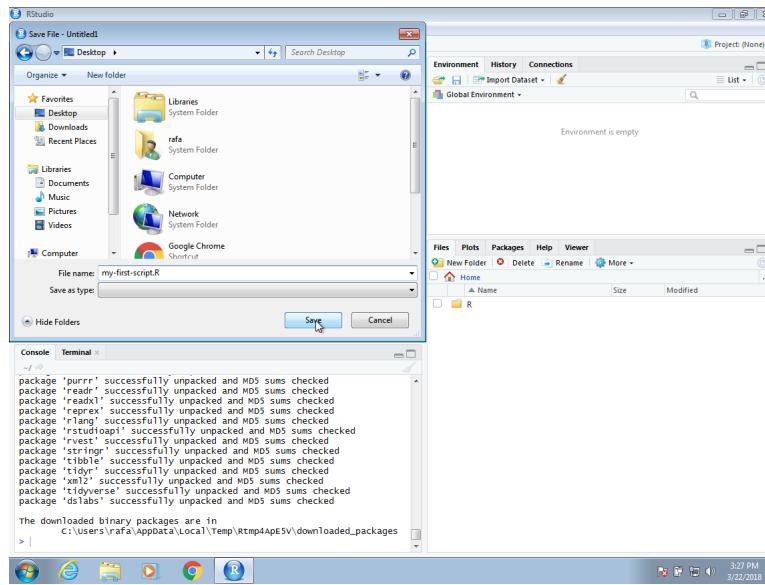
Recomendamos manter isso à mão, para que você possa procurar por combinações de teclas quando perceber que está repetindo tarefas na interface gráfica.

1.4.3 Como executar comandos enquanto edita *scripts*

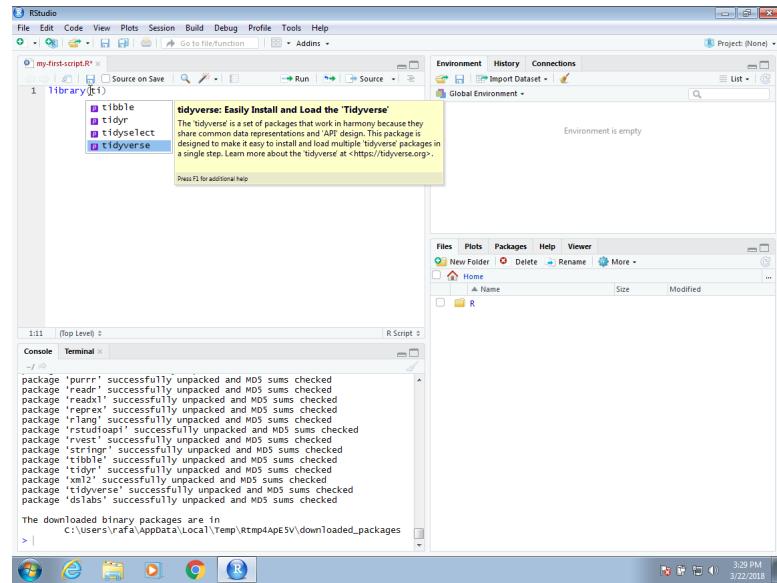
Existem muitos editores projetados especificamente para programar. Isso é útil porque a cor e o recuo são adicionados automaticamente para tornar o código mais legível. O RStudio é um desses editores e foi desenvolvido especificamente para R. Uma das principais vantagens do RStudio sobre outros editores é que podemos testar facilmente nosso código enquanto editamos nossos *scripts*. Aqui está um exemplo.

Vamos começar abrindo um novo *script* como fizemos antes. Então, vamos nomear o *script*. Podemos fazer isso através do editor salvando o novo *script* atual ainda sem nome. Para começar, clique no ícone Salvar ou use a combinação de teclas Ctrl + S no Windows e Command + S no Mac.

Quando você tenta salvar o documento pela primeira vez, o RStudio solicita um nome. Uma boa convenção é usar um nome descritivo, com letras minúsculas, sem espaços, apenas hífens para separar as palavras e, em seguida, seguido pelo sufixo *.R*. Vamos chamar esse *script*: *my-first-script.R*.



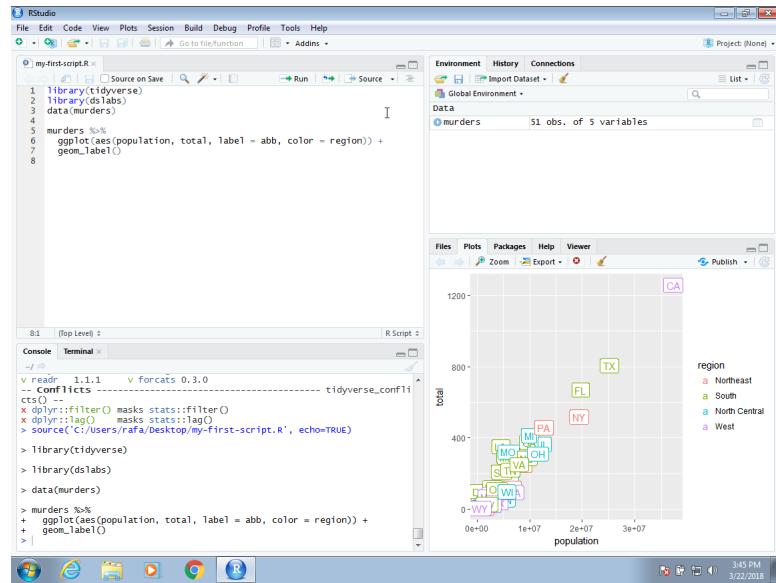
Agora estamos prontos para começar a editar nosso primeiro *script*. As primeiras linhas de código em um *script* R são dedicadas ao carregamento dos pacotes que iremos usar. Outro recurso útil do RStudio é que, uma vez que digitamos `library()`, o RStudio começa a autocompletar o que estamos escrevendo com as bibliotecas que instalamos. Veja o que acontece quando escrevemos `library(ti)`:



Outra característica que você deve ter notado é que, quando digita `library()`, o segundo parêntese é adicionado automaticamente. Isso ajudará a evitar um dos erros de programação mais comuns: esquecer de fechar um parêntese.

Agora podemos continuar escrevendo código. Como exemplo, criaremos um gráfico mostrando totais de assassinatos *versus* tamanho da população por estado nos EUA. Uma vez que você termine de escrever o código necessário para construir esse gráfico, você pode testá-lo executando-o. Para fazer isso, clique no botão *Run* no canto superior direito do painel de edição. Você também pode usar a combinação de teclas: Ctrl + Shift + Enter no Windows ou Command + Shift + Return no Mac.

Assim que executar o código, você verá que ele aparece no console do R e, neste caso, o gráfico resultante aparecerá no console de gráficos. Note que o console gráfico possui uma interface útil que permite que você navegue por diferentes gráficos gerados (clicando nas setas para ir frente ou para trás), amplie regiões do gráfico ou salve os gráficos como arquivos.



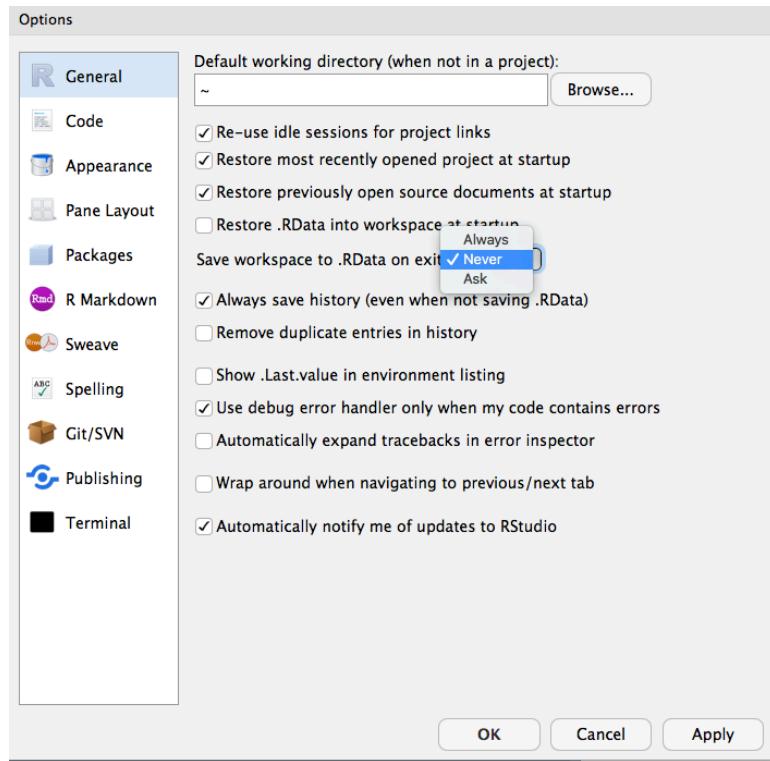
Para executar uma linha de cada vez, em vez de todo o *script*, você pode usar Control-Enter no Windows e Command-Return no Mac.

1.4.4 Como alterar opções globais

Você pode mudar bastante a aparência e a funcionalidade do RStudio.

Para alterar as opções globais, clique em *Tools* (ferramentas) e, em seguida, em *Global Options*....

Como exemplo, mostramos como fazer uma mudança que **recomendamos bastante**: altere a opção *Save workspace to .RData on exit* (salvar espaço de trabalho para .RData na saída) para *Never* (nunca) e desmarque a opção *Restore .RData into workspace at start* (restaurar o arquivo .RData no espaço de trabalho ao iniciar). Por padrão, quando alguém sai do R, o programa salva todos os objetos que ele criou em um arquivo chamado .RData. Isso ocorre para que, ao reiniciar a sessão no mesmo arquivo, o programa carregue esses objetos. No entanto, descobrimos que isso causa confusão, principalmente quando compartilhamos código com colegas e assumimos que eles têm esse arquivo .RData. Para alterar essas opções, faça com que suas configurações na aba *General* fiquem assim:



1.5 Instalando pacotes R

As funcionalidades fornecidas por uma versão do R recém-instalada são apenas uma pequena parte de tudo que é possível. De fato, nos referimos ao que você obtém após sua primeira instalação como *base R*. Funcionalidades adicionais podem ser adicionadas a partir de plugins (*add-ons*, em inglês) disponibilizados por desenvolvedores. Atualmente, existem centenas de plugins disponíveis no CRAN e muitos outros compartilhados em outros repositórios como o GitHub. No entanto, como nem todo mundo precisa de todas as funções disponíveis, o R disponibiliza diferentes componentes através de pacotes (*packages*, em inglês). O R facilita a instalação de pacotes dentro do próprio R. Por exemplo, para instalar o pacote **dslabs**, que usamos para compartilhar os conjuntos de dados e códigos relacionados a este livro, você deve escrever:

```
install.packages("dslabs")
```

No RStudio, você pode navegar para a guia *Tools* e selecionar *install packages* (instalar pacotes). Em seguida, podemos carregar o pacote em nossas sessões R usando a função *library*:

```
library(dslabs)
```

Ao ler este livro, você verá que carregamos pacotes sem instalá-los. Isso ocorre porque, quando um pacote é instalado, ele permanece instalado e só precisa ser carregado com *library*. O pacote permanece carregado até terminarmos a sessão R. Se você tentar car-

regar um pacote e obtiver uma mensagem de erro, provavelmente significa que ele não está instalado e você deve instalá-lo primeiro.

Podemos instalar mais de um pacote por vez, fornecendo uma lista de nomes para esta função:

```
install.packages(c("tidyverse", "dslabs"))
```

Observe que a instalação de **tidyverse** realmente instala vários pacotes. Isso geralmente ocorre quando um pacote possui *dependências* ou usa funções de outros pacotes. Quando você carrega um pacote usando **library**, você também carrega suas dependências.

Após a instalação dos pacotes, você pode carregá-los no R e não precisa instalá-los novamente, a menos que instale uma nova versão do R. Lembre-se de que os pacotes estão instalados no R e não no RStudio.

É útil manter uma lista de todos os pacotes necessários para o trabalho em um *script* porque, se você necessitar realizar uma nova instalação do R, você poderá reinstalar todos os pacotes simplesmente executando um *script*.

Você pode ver todos os pacotes que já instalou usando a seguinte função:

```
installed.packages()
```

Part I

R

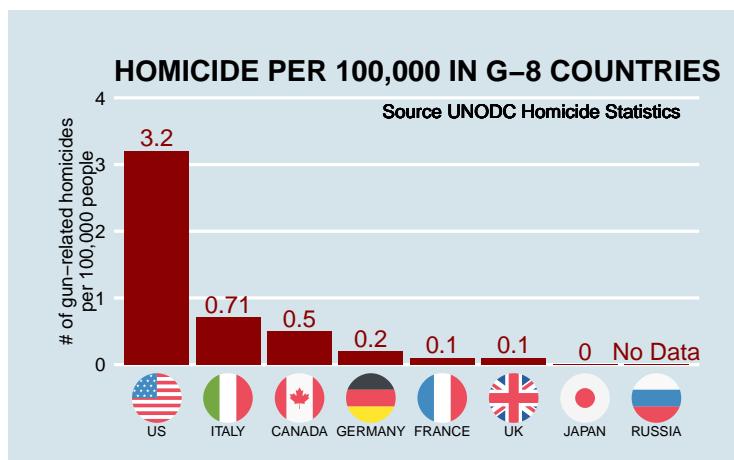
2

R básico

Neste livro, usaremos o ambiente de *software* do R para todas as nossas análises. Você aprenderá técnicas de análise de dados e R simultaneamente. Portanto, para continuar, você necessitará de acesso ao R. Também recomendamos o uso de um ambiente de desenvolvimento integrado (no inglês *Integrated Development Environment* ou, simplesmente, IDE), como o RStudio, para salvar seu trabalho. Observe que é comum que um curso ou *workshop* ofereça acesso a um ambiente R e um IDE por meio de um navegador da web, como faz o RStudio cloud¹. Se você tiver acesso a tal recurso, não será necessário instalar o R ou o RStudio. Entretanto, se você pretende se tornar um analista de dados especializado, recomendamos fortemente que instale essas ferramentas em seu computador². O R e o RStudio são gratuitos e estão disponíveis online.

2.1 Estudo de caso: assassinatos por armas nos EUA

Imagine que você viva na Europa e receba uma oferta de emprego em uma empresa americana com filiais em vários estados. É um ótimo trabalho, mas manchetes como ** Taxa de homicídio por arma de fogo nos Estados Unidos é mais alta do que em outros países desenvolvidos **³ têm preocupado você. Gráficos como este podem preocupá-lo ainda mais:



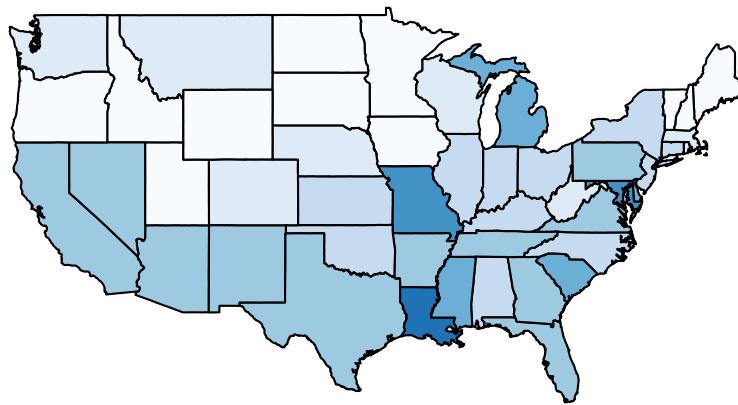
Mas então você se lembra que os Estados Unidos são um país grande e diversificado, com

¹<https://rstudio.cloud>

²<https://rafalab.github.io/dsbook/installing-r-rstudio.html>

³<http://abcnews.go.com/blogs/headlines/2012/12/us-gun-ownership-homicide-rate-higher-than-other-developed-countries/>

50 estados muito distintos, além do Distrito de Columbia ou simplesmente DC (equivalente a um distrito federal).



A Califórnia, por exemplo, tem uma população maior que o Canadá. Além disso, 20 estados dos EUA têm populações maiores que a Noruega. Em alguns aspectos, a variabilidade entre os estados dos EUA é semelhante à variabilidade entre países da Europa. Além disso, embora não esteja incluído nas tabelas acima, as taxas de homicídio na Lituânia, Ucrânia e Rússia são superiores a quatro por 100.000. Então, talvez as notícias que o preocupam sejam superficiais demais. Você tem opções de onde pode morar e quer determinar a segurança de cada estado em particular. Podemos ter uma percepção geral ao examinar dados relacionados a homicídios por armas de fogo nos EUA no ano de 2010 usando R.

Antes de começarmos o nosso exemplo, precisamos discutir a logística e alguns dos componentes necessários para obter habilidades mais avançadas de R. Esteja ciente de que a utilidade de alguns desses componentes nem sempre será imediatamente óbvia, entretanto, mais adiante neste livro, você apreciará dominar essas habilidades.

2.2 Os princípios básicos

Antes de começarmos com o conjunto de dados motivadores, precisamos revisar o básico de R.

2.2.1 Objetos

Suponha que alguns alunos do ensino médio nos solicitem ajuda para resolver várias equações quadráticas da forma $ax^2 + bx + c = 0$. A fórmula quadrática nos oferece as soluções:

$$\frac{-b - \sqrt{b^2 - 4ac}}{2a} \text{ e } \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

que, é claro, mudam dependendo dos valores de a , b e c . Uma vantagem das linguagens de programação é que podemos definir variáveis e escrever expressões com elas, semelhante à maneira como fazemos isso na matemática, mas obter uma solução numérica. Escreveremos um código geral para a equação quadrática abaixo, mas se formos solicitados a resolver $x^2 + x - 1 = 0$, então definimos:

```
a <- 1
b <- 1
c <- -1
```

que armazena os valores para uso posterior. Nós usamos `<-` para atribuir valores a variáveis.

Também podemos atribuir valores usando `=` ao invés de `<-`, mas recomendamos não usar `=` para evitar confusão.

Copie e cole o código acima no seu console para definir as três variáveis. Observe que R não imprime nada quando fazemos essa atribuição. Isso significa que os objetos foram definidos com sucesso. Caso tivesse cometido algum erro, você teria recebido uma mensagem de erro.

Para ver o valor armazenado em uma variável, simplesmente pedimos que R avalie `a` e isso mostra o valor armazenado:

```
a
#> [1] 1
```

Uma maneira mais explícita de pedir ao R para mostrar o valor armazenado em `a` é usar `print` desta forma:

```
print(a)
#> [1] 1
```

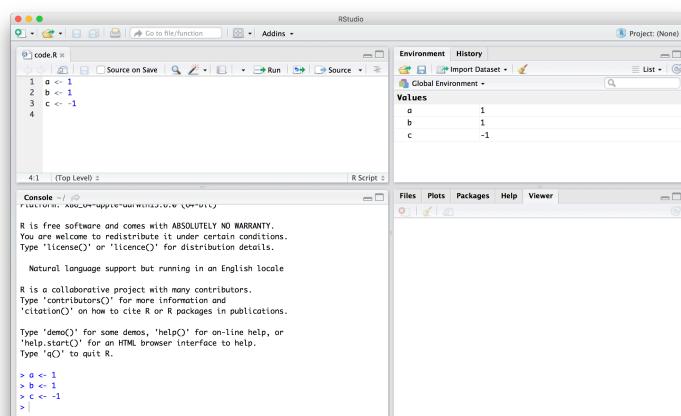
Usamos o termo *object* (objeto) para descrever coisas armazenadas em R. Variáveis são exemplos, mas objetos também podem ser entidades mais complicadas, tais como funções, que serão descritas mais adiante.

2.2.2 A área de trabalho (*workspace*)

Conforme definimos os objetos no console, estamos fazendo alterações na área de trabalho (*workspace* em inglês ou ainda espaço de trabalho). Você pode ver todas as variáveis salvas no sua área de trabalho digitando:

```
ls()
#> [1] "a"          "b"          "c"          "dat"        "img_path"    "murders"
```

No RStudio, a guia *Environment* mostra os valores:



Devemos ver **a**, **b** e **c**. Se tentar recuperar o valor de uma variável que não está em sua área de trabalho, você receberá uma mensagem de erro. Por exemplo, se você escrever **x**, verá a seguinte mensagem: **Error: object 'x' not found** (Erro: objeto 'x' não encontrado).

Agora, como esses valores são armazenados em variáveis, para resolver nossa equação, usamos a fórmula quadrática:

```
(-b + sqrt(b^2 - 4*a*c) )/ ( 2*a )
#> [1] 0.618
(-b - sqrt(b^2 - 4*a*c) )/ ( 2*a )
#> [1] -1.62
```

2.2.3 Funções

Uma vez definidas as variáveis, o processo de análise de dados geralmente pode ser descrito como uma série de *funções* aplicadas aos dados. R inclui diversas funções predefinidas e a maioria dos conjuntos de análise que construímos fazem uso extensivo delas.

Nós já usamos as funções `install.packages`, `library` e `ls`. Também usamos a função `sqrt` para resolver a equação quadrática acima. Existem muitas funções predefinidas e ainda mais podem ser adicionadas através de pacotes. Essas funções não aparecem na área de trabalho porque não foi você quem as definiu, mas elas estão disponíveis para uso imediato.

Em geral, precisamos usar parênteses para chamar uma função. Se você escrever `ls`, a função não é executada e, em vez disso, R mostra o código que a define. Se você escrever `ls()`, a função é executada e, como já mostrado, listamos os objetos na área de trabalho.

Diferente de `ls`, a maioria das funções requer um ou mais *argumentos*. Abaixo está um exemplo de como atribuímos um objeto como argumento da função `log`. Lembre-se de que anteriormente definimos `a` como 1:

```
log(8)
#> [1] 2.08
log(a)
#> [1] 0
```

Você pode descobrir o que a função espera e o que faz, revendo os muito úteis manuais incluídos no R. Você pode obter ajuda usando a função `help` assim:

```
help("log")
```

Para a maioria das funções, também podemos usar esta abreviação:

```
?log
```

A página de ajuda mostrará quais argumentos a função está esperando. Por exemplo, `log` precisa de um valor `x` e uma `base` para executar. Entretanto, alguns argumentos são obrigatórios e outros são opcionais. Você pode determinar quais argumentos são opcionais observando no documento de ajuda quais padrões são atribuídos com `=`. Definir tais argumentos é opcional. Por exemplo, a base da função `log` por padrão é `base = exp(1)` fazendo `log` o logaritmo natural padrão.

Para dar uma olhada rápida nos argumentos sem abrir o sistema de ajuda, você pode escrever:

```
args(log)
#> function (x, base = exp(1))
#> NULL
```

Você pode alterar os valores padrão simplesmente atribuindo outro objeto:

```
log(8, base = 2)
#> [1] 3
```

Lembre-se de que não é necessário atribuir o argumento `x` desta forma:

```
log(x = 8, base = 2)
#> [1] 3
```

O código acima funciona, mas também podemos economizar um pouco de digitação: se o nome do argumento não for especificado, R assume que você está inserindo argumentos na ordem em que são exibidos na página de ajuda ou nos `args`. Portanto, ao não usar os nomes, R assume que os argumentos são `x` seguido pela `base`:

```
log(8,2)
#> [1] 3
```

Se os nomes dos argumentos forem usados, então podemos incluí-los em qualquer ordem:

```
log(base = 2, x = 8)
#> [1] 3
```

Para especificar argumentos, devemos usar `=` e não `<-`.

Existem algumas exceções à regra de que funções precisam que parênteses para serem executadas. Entre essas exceções, os mais utilizados são os operadores aritméticos e relacionais. Por exemplo:

```
2^3
#> [1] 8
```

Você pode ver os operadores aritméticos digitando:

```
help("+")
```

ou

```
?"+"
```

e os operadores relacionais ao escrever:

```
help(">")
```

ou

```
?">"
```

2.2.4 Outros objetos pré-construídos

Existem diversos conjuntos de dados (*datasets*) incluídos para os usuários praticarem e testarem funções. Você pode ver todos os *datasets* disponíveis digitando:

```
data()
```

Isso mostra o nome do objeto para esses *datasets*. Esses conjuntos de dados são objetos que podem ser usados simplesmente digitando seu nome. Por exemplo, se você digitar:

```
co2
```

R mostrará os dados da concentração atmosférica de CO₂ de Mauna Loa.

Outros objetos pré-construídos (*prebuilt*) são as constantes matemáticas, como a constante π e ∞ :

```
pi
#> [1] 3.14
Inf+1
#> [1] Inf
```

2.2.5 Nomes de variáveis

Anteriormente, usamos as letras `a`, `b` e `c` como nomes de variáveis, mas nomes de variáveis podem ser quase qualquer coisa. Algumas regras básicas em R definem apenas que os nomes de variáveis devem começar com uma letra, não podem conter espaços e não devem ser variáveis predefinidas em R. Por exemplo, não nomeie uma de suas variáveis `install.packages` escrevendo algo como: `install.packages <- 2`.

Uma boa convenção a seguir é usar palavras significativas que descrevam o que é armazenado, use apenas letras minúsculas e sublinhados como substitutos de espaços. Para equações quadráticas, podemos usar algo como:

```
solucao_1 <- (-b + sqrt(b^2 - 4*a*c))/ (2*a)
solucao_2 <- (-b - sqrt(b^2 - 4*a*c))/ (2*a)
```

Para obter mais dicas, recomendamos estudar o guia de estilo de Hadley Wickham⁴.

2.2.6 Como salvar sua área de trabalho

Valores permanecem na área de trabalho até você encerrar a sessão ou excluí-los com a função `rm`. Mas as áreas de trabalho também podem ser salvas para uso posterior. De fato, ao sair do R, o programa pergunta se você deseja salvar seu *workspace*. Se você salvá-lo, na próxima vez que iniciar o R, o programa restaurará a área de trabalho.

Não recomendamos salvar a área de trabalho dessa maneira, porque, à medida que você começa a trabalhar em projetos diferentes, se tornará mais difícil acompanhar o que está salvo. Em vez disso, recomendamos que você atribua à área de trabalho um nome específico. Você pode fazer isso usando as funções `save` ou `save.image`. Para carregar, use a função `load`. Ao salvar um *workspace*, recomendamos os sufixos `rda` ou `RData`. No RStudio, você também pode fazer isso navegando até a guia *Session* e escolhendo a opção *Save Workspace as* (salvar área de trabalho como). Você pode carregá-los mais tarde usando as opções *Load Workspace* na mesma guia. Para saber mais, leia as páginas de ajuda de `save`, `save.image` e `load`.

⁴<http://adv-r.had.co.nz/Style.html>

2.2.7 Fundamentando o uso de *scripts*

Para resolver outra equação como $3x^2 + 2x - 1$, podemos copiar e colar o código acima, redefinir as variáveis e recalcular a solução:

```
a <- 3
b <- 2
c <- -1
(-b + sqrt(b^2 - 4*a*c))/ (2*a)
(-b - sqrt(b^2 - 4*a*c))/ (2*a)
```

Ao criar e salvar um *script* com o código acima, não precisaríamos redigitar tudo novamente e, em vez disso, simplesmente alterar os nomes das variáveis. Tente escrever o script acima em um editor e observe como é fácil alterar variáveis e obter uma resposta.

2.2.8 Como comentar seu código

Se uma linha de código R começar com o símbolo `#`, ela não é executada. Podemos usar isso para escrever lembretes sobre por que escrevemos um código específico. Por exemplo, no *script* acima, poderíamos adicionar:

```
## Código para calcular a solução de uma equação quadrática de forma ax^2 + bx + c
## define as variáveis
a <- 3
b <- 2
c <- -1

## agora, calcule a solução
(-b + sqrt(b^2 - 4*a*c))/ (2*a)
(-b - sqrt(b^2 - 4*a*c))/ (2*a)
```

2.3 Exercícios

1. Qual é a soma dos 100 primeiros números inteiros positivos? A fórmula para a soma dos números inteiros de 1 a n é $n(n + 1)/2$. Defina $n = 100$ e então use R para calcular a soma de 1 a 100 usando a fórmula. Qual é a soma?

2. Agora use a mesma fórmula para calcular a soma dos números inteiros de 1 a 1000.

3. Observe o resultado ao digitar o seguinte código em R:

```
n <- 1000
x <- seq(1, n)
sum(x)
```

Com base no resultado, o que você acha que as funções fazem `seq` e `sum`? Você pode usar a função `help`.

- `sum` cria uma lista de números e `seq` os soma.
- `seq` cria uma lista de números e `sum` os soma.
- `seq` cria uma lista aleatória e `sum` calcula a soma de 1 a 1000.

- d. `sum` sempre retorna o mesmo número.
4. Em matemática e programação, dizemos que executamos uma função quando substituímos o argumento por um determinado número. Então, se digitarmos `sqrt(4)`, executamos a função `sqrt`. Em R, você pode executar uma função dentro de outra função. As execuções acontecem de dentro para fora. Escreva uma linha de código para calcular o logaritmo, base 10, da raiz quadrada de 100.
5. Qual das seguintes opções sempre retornará o valor numérico armazenado em `x`? Você pode testar os exemplos abaixo ou usar o sistema de ajuda, se desejar.
- `log(10^x)`
 - `log10(x^10)`
 - `log(exp(x))`
 - `exp(log(x, base = 2))`
-

2.4 Tipos de dados

Variáveis em R podem ser de tipos diferentes. Por exemplo, precisamos distinguir números de sequências de caracteres, e tabelas de simples listas de números. A função `class` nos ajuda a determinar que tipo de objeto temos:

```
a <- 2
class(a)
#> [1] "numeric"
```

Para trabalhar com eficiência no R, é importante aprender os diferentes tipos de variáveis e o que podemos fazer com elas.

2.4.1 Data frames

Até agora, as variáveis que definimos são apenas numéricas. Isso não é muito útil para armazenar dados. A maneira mais comum de armazenar um conjunto de dados em R é usar um *data frame*. Conceitualmente, podemos pensar em um *data frame* como uma tabela com linhas que representam observações e com colunas que representam as diferentes variáveis coletadas para cada observação. *Data frames* são particularmente úteis para *datasets* pois permitem combinar diferentes tipos de dados em um único objeto.

Uma grande proporção dos desafios da análise de dados começa com os dados armazenados em um *data frame*. Por exemplo, armazenamos os dados do nosso exemplo motivador em um *data frame*. Você pode acessar esse *dataset* carregando a biblioteca `dslabs` e, em seguida, usando a função `data` carregar o *dataset* `murders` :

```
library(dslabs)
data(murders)
```

Para verificar se esse objeto é de fato um *data frame*, digitamos:

```
class(murders)
#> [1] "data.frame"
```

2.4.2 Examinando um objeto

A função `str` é útil para obter mais informações sobre a estrutura de um objeto:

```
str(murders)
#> #> 'data.frame': 51 obs. of 5 variables:
#> #> $ state : chr "Alabama" "Alaska" "Arizona" "Arkansas" ...
#> #> $ abb : chr "AL" "AK" "AZ" "AR" ...
#> #> $ region : Factor w/ 4 levels "Northeast", "South", ...: 2 4 4 2 4 4 1 2 2
#> #> 2 ...
#> #> $ population: num 4779736 710231 6392017 2915918 37253956 ...
#> #> $ total : num 135 19 232 93 1257 ...
```

Isso nos diz muito mais sobre o objeto. Vemos que a tabela possui 51 linhas (50 estados mais DC) e cinco variáveis. Podemos exibir as seis primeiras linhas usando a função `head`:

```
head(murders)
#> #>      state abb region population total
#> #> 1   Alabama AL   South     4779736   135
#> #> 2   Alaska  AK   West      710231    19
#> #> 3   Arizona AZ   West      6392017   232
#> #> 4   Arkansas AR   South     2915918    93
#> #> 5   California CA   West      37253956  1257
#> #> 6   Colorado CO   West      5029196    65
```

Nesse conjunto de dados, cada estado é considerado uma observação e cinco variáveis são reportadas para cada estado.

Antes de irmos além em responder à nossa pergunta original sobre os diferentes estados, vamos revisar um pouco mais sobre os componentes deste objeto.

2.4.3 O operador de acesso: \$

Para nossa análise, precisaremos acessar as diferentes variáveis representadas pelas colunas incluídas neste *data frame*. Para fazer isso, usamos o operador de acesso `$` da seguinte maneira:

```
murders$population
#> [1] 4779736 710231 6392017 2915918 37253956 5029196 3574097
#> [8] 897934 601723 19687653 9920000 1360301 1567582 12830632
#> [15] 6483802 3046355 2853118 4339367 4533372 1328361 5773552
#> [22] 6547629 9883640 5303925 2967297 5988927 989415 1826341
#> [29] 2700551 1316470 8791894 2059179 19378102 9535483 672591
#> [36] 11536504 3751351 3831074 12702379 1052567 4625364 814180
#> [43] 6346105 25145561 2763885 625741 8001024 6724540 1852994
#> [50] 5686986 563626
```

Mas como sabíamos que havia uma coluna denominada `population`? Anteriormente, ao aplicar a função `str` ao objeto `murders`, descobrimos os nomes de cada uma das cinco variáveis armazenadas nesta tabela. Podemos acessar rapidamente nomes de variáveis usando:

```
names(murders)
#> [1] "state"       "abb"        "region"      "population" "total"
```

É importante saber que a ordem das entradas em `murders$population` preserva a ordem

das linhas em nossa tabela de dados. Isso nos permitirá manipular uma variável com base nos resultados de outra. Por exemplo, podemos classificar os nomes dos estados de acordo com o número de assassinatos.

Dica: R vem com uma funcionalidade de preenchimento automático muito boa, que evita o trabalho de digitar todos os nomes. Escreva `murders$p` e pressione a tecla *tab* no teclado. Essa funcionalidade e muitos outros recursos úteis de preenchimento automático estão disponíveis no RStudio.

2.4.4 Vetores: numéricos, caracteres e lógicos

O objeto `murders$population` não é um número, mas vários. Chamamos esse tipo de objeto de *vectors* (vetores). Um número único é tecnicamente um vetor de comprimento 1, mas em geral usamos o termo vetores para se referir a objetos com várias entradas. A função `length` informa quantas entradas existem no vetor:

```
pop <- murders$population
length(pop)
#> [1] 51
```

Esse vetor específico é denominado numérico (*numeric*), pois os tamanhos da população são números:

```
class(pop)
#> [1] "numeric"
```

Em um vetor numérico, cada entrada deve ser um número.

Para armazenar uma cadeia de caracteres (*strings*), os vetores também podem ser da classe *character*. Por exemplo, os nomes dos estados são caracteres:

```
class(murders$state)
#> [1] "character"
```

Assim como nos vetores numéricos, todas as entradas em um vetor de caracteres devem ter um caractere.

Outro tipo importante de vetores são os vetores lógicos (*logical*). Estes devem ser `TRUE` (verdadeiro) ou `FALSE` (falso).

```
z <- 3 == 2
z
#> [1] FALSE
class(z)
#> [1] "logical"
```

Aqui o `==` é um operador relacional que pergunta se 3 é igual a 2. Em R, use apenas um `=` atribuir uma variável, mas se usar duas `==` então você estará avaliando se os objetos são iguais.

Você pode ver outros operadores relacionais escrevendo:

```
?Comparison
```

Nas próximas seções, você verá como os operadores relacionais podem ser úteis.

Discutiremos as características mais importantes dos vetores após os exercícios a seguir.

Avançado: matematicamente, os valores em `pop` são números inteiros e há uma classe para números inteiros (`integer`) em R. No entanto, por padrão, os números recebem a classe numérica (`numeric`) mesmo quando são números inteiros redondos. Por exemplo, `class(1)` retorna `numeric`. Você pode convertê-lo para classe `integer` com a função `as.integer()` ou adicionando um L assim: `1L`. Observe a classe escrevendo: `class(1L)`

2.4.5 Fatores (*factors*)

No dataset `murders`, nós poderíamos esperar que a região também fosse um vetor de caracteres. Entretanto, não é:

```
class(murders$region)
#> [1] "factor"
```

É um *fator*. Fatores são úteis para armazenar dados categóricos. Podemos ver que existem apenas quatro regiões ao usar a função `levels`:

```
levels(murders$region)
#> [1] "Northeast"      "South"           "North Central" "West"
```

Em segundo plano, R armazena esses *levels* (níveis) como números inteiros (*integers*) e mantém um mapa para acompanhar os rótulos. Isso é mais eficiente quando se considera o uso de memória do que armazenar todos os caracteres.

Observe que os níveis têm uma ordem que é diferente da ordem de aparecimento em um objeto *factor*. Em R, por padrão, os níveis são organizados em ordem alfabética. No entanto, geralmente queremos que os níveis sigam uma ordem diferente. Você pode especificar a ordem através do argumento `levels` quando criar o fator com a função `factor`. Por exemplo, no conjunto de dados de assassinatos, as regiões são ordenadas de leste a oeste. A função `reorder` permite alterar a ordem dos níveis de uma variável *factor* baseado em um resumo calculado em um vetor numérico. Agora, demonstraremos isso usando um simples exemplo. Veremos exemplos mais avançados na seção de Visualização de Dados do livro.

Vamos supor que desejemos que os níveis da região sejam ordenados de acordo com o número total de assassinatos, e não por ordem alfabética. Se houver valores associados a cada nível, podemos usar `reorder` para determinar a ordem. O código a seguir pega a soma do total de assassinatos em cada região e reordena o fator de acordo com essas somas.

```
region <- murders$region
value <- murders$total
region <- reorder(region, value, FUN = sum)
levels(region)
#> [1] "Northeast"      "North Central" "West"           "South"
```

A nova ordem é consistente com o fato de que há menos assassinatos no nordeste e mais no sul.

Aviso: os fatores podem causar confusão, pois às vezes eles se comportam como caracteres e às vezes não. Como resultado, esse tipo de dados são uma fonte comum de erros.

2.4.6 Listas

Data frames são um caso especial de listas (do inglês *lists*). Abordaremos as listas com mais detalhes posteriormente, mas saiba que elas são úteis porque podem armazenar qualquer

combinação de diferentes tipos de dados. Abaixo, apresentamos um exemplo de uma lista que criamos para você:

```
record
#> $name
#> [1] "João das Couves"
#>
#> $student_id
#> [1] 1234
#>
#> $grades
#> [1] 95 82 91 97 93
#>
#> $final_grade
#> [1] "A"
class(record)
#> [1] "list"
```

Assim como em *data frames*, você pode extrair os componentes de uma lista com o operador de acesso: `$`. De fato, *data frames* são um tipo de lista.

```
record$student_id
#> [1] 1234
```

Também podemos usar colchetes duplos (`[[]]`) assim:

```
record[["student_id"]]
#> [1] 1234
```

Você deve se acostumar com o fato de que geralmente existem várias maneiras em R de fazer a mesma coisa, como acessar entradas.

Você também pode encontrar listas sem nomes de variáveis:

```
record2
#> [[1]]
#> [1] "João das Couves"
#>
#> [[2]]
#> [1] 1234
```

Se uma lista não tiver nomes, você não pode extrair os elementos com `$`, mas você pode ainda usar o método de colchete. Em vez de fornecer o nome da variável, você pode fornecer o índice da lista da seguinte maneira:

```
record2[[1]]
#> [1] "João das Couves"
```

Não usaremos listas até mais tarde, mas você pode encontrar uma em suas próprias explorações de R. É por isso que mostramos alguns conceitos básicos aqui.

2.4.7 Matrizes

Matrizes são outro tipo comum de objeto em R. Matrizes são semelhantes a *data frames*, pois são bidimensionais: possuem linhas e colunas. No entanto, como vetores numéricos,

caracteres e lógicos, as entradas nas matrizes devem ser do mesmo tipo. Por esse motivo, *data frames* são muito mais úteis para armazenar dados, pois permitem armazenar caracteres, fatores e números.

Entretanto, as matrizes têm uma grande vantagem sobre os *data frames*: podemos executar operações de álgebra matricial, uma poderosa técnica matemática. Não descrevemos essas operações neste livro, mas muito do que acontece em segundo plano quando você executa uma análise de dados envolve matrizes. Abordamos matrizes em mais detalhes nos próximos capítulos. Entretanto, as discutiremos brevemente aqui, uma vez que algumas das funções que aprenderemos retornam matrizes.

Podemos definir uma matriz usando a função `matrix`. Precisamos especificar o número de linhas e colunas:

```
mat <- matrix(1:12, 4, 3)
mat
#>      [,1] [,2] [,3]
#> [1,]    1    5    9
#> [2,]    2    6   10
#> [3,]    3    7   11
#> [4,]    4    8   12
```

Você pode acessar entradas específicas em uma matriz usando colchetes (`[]`). Por exemplo, se você deseja o valor armazenado na segunda linha, terceira coluna, use:

```
mat[2, 3]
#> [1] 10
```

Se você deseja todos os itens da segunda linha, deixe o valor da coluna vazio:

```
mat[2, ]
#> [1] 2 6 10
```

Observe que isso retorna um vetor e não uma matriz.

Da mesma forma, se você quiser a terceira coluna inteira, deixe o valor da linha vazio:

```
mat[, 3]
#> [1] 9 10 11 12
```

Este também é um vetor, não uma matriz.

Você pode acessar mais de uma coluna ou mais de uma linha se desejar. Isso lhe dará uma nova matriz.

```
mat[, 2:3]
#>      [,1] [,2]
#> [1,]    5    9
#> [2,]    6   10
#> [3,]    7   11
#> [4,]    8   12
```

Você pode criar subconjuntos de linhas e colunas:

```
mat[1:2, 2:3]
#>      [,1] [,2]
```

```
#> [1,] 5 9
#> [2,] 6 10
```

Podemos converter as matrizes em *data frames* usando a função `as.data.frame`:

```
as.data.frame(mat)
#>   V1 V2 V3
#> 1  1  5  9
#> 2  2  6 10
#> 3  3  7 11
#> 4  4  8 12
```

Também podemos usar colchetes individuais (`[]`) para acessar as linhas e colunas de um *data frame*:

```
data("murders")
murders[25, 1]
#> [1] "Mississippi"
murders[2:3, ]
#>   state abb region population total
#> 2 Alaska AK    West      710231     19
#> 3 Arizona AZ    West     6392017    232
```

2.5 Exercícios

- Carregue o conjunto de dados de assassinatos nos EUA.

```
library(dslabs)
data(murders)
```

Use a função `str` para examinar a estrutura do objeto `murders`. Qual das opções a seguir descreve melhor as variáveis representadas nesse *data frame*?

- Os 51 estados.
 - Taxas de assassinato para todos os 50 estados e DC.
 - O nome do estado, a abreviação do nome do estado, região e população do estado e o número total de assassinatos em 2010 no estado.
 - `str` não exibe informações relevantes.
- Quais são os nomes das colunas usadas pelo *data frame* para essas cinco variáveis?
 - Use o operador de acesso `$` para extrair as abreviações dos estados e atribuí-las ao objeto `a`. Qual é a classe desse objeto?
 - Agora use os colchetes para extrair as abreviações dos estados e atribuí-las ao objeto `b`. Use a função `identical` para determinar se `a` e `b` são iguais.
 - Vimos que a coluna `region` armazena um fator. Você pode verificar isso digitando:

```
class(murders$region)
```

Com uma linha de código, use as funções `levels` e `length` para determinar o número de regiões definidas por esse conjunto de dados.

6. A função `table` pega um vetor e retorna a frequência de cada elemento. Você pode ver rapidamente quantos estados existem em cada região aplicando esta função. Use essa função em uma linha de código para criar uma tabela de estados por região.

2.6 Vetores

Em R, os objetos mais básicos disponíveis para armazenar dados são os *vectors* (vetores). Como vimos, conjuntos de dados complexos geralmente podem ser divididos em componentes que são vetores. Por exemplo, em um *data frame*, cada coluna é um vetor. Aqui nós aprendemos mais sobre essa importante classe.

2.6.1 Criando vetores

Podemos criar vetores usando a função `c`, que significa *concatenate* (concatenar). Nós usamos `c` para concatenar entradas da seguinte maneira:

```
codes <- c(380, 124, 818)
codes
#> [1] 380 124 818
```

Também podemos criar vetores de caracteres. Usamos aspas para indicar que as entradas são caracteres e não nomes de variáveis.

```
country <- c("italy", "canada", "egypt")
```

Em R, também pode-se usar apóstrofos:

```
country <- c('italy', 'canada', 'egypt')
```

Mas tome cuidado para não confundir o apóstrofo (*single quote*) ' com o acento grave (*back quote*) `.

Até agora você deve saber que se você escrever:

```
country <- c(italy, canada, egypt)
```

você receberá um erro porque as variáveis `italy`, `canada` e `egypt` não estão definidos. Se não usarmos as aspas, R procurará variáveis com esses nomes e retornará um erro.

2.6.2 Nomes

Às vezes, é útil nomear as entradas de um vetor. Por exemplo, ao definir um vetor de códigos de países, podemos usar os nomes para conectar os dois:

```
codes <- c(italy = 380, canada = 124, egypt = 818)
codes
#> italy canada egypt
#> 380     124     818
```

O objeto `codes` continua sendo um vetor numérico:

```
class(codes)
#> [1] "numeric"
```

mas com nomes:

```
names(codes)
#> [1] "italy"  "canada" "egypt"
```

Se o uso de *strings* sem aspas parecer confuso, saiba que você também pode usar aspas:

```
codes <- c("italy" = 380, "canada" = 124, "egypt" = 818)
codes
#> italy  canada  egypt
#> 380    124     818
```

Não há diferença entre essa chamada de função (*function call* em inglês) e a anterior. Essa é uma das muitas maneiras pelas quais R é peculiar em comparação com outras linguagens.

Também podemos atribuir nomes usando a função `names`:

```
codes <- c(380, 124, 818)
country <- c("italy", "canada", "egypt")
names(codes) <- country
codes
#> italy  canada  egypt
#> 380    124     818
```

2.6.3 Sequências

Outra função útil para criar vetores gerando sequências:

```
seq(1, 10)
#> [1] 1 2 3 4 5 6 7 8 9 10
```

O primeiro argumento define o começo e o segundo define o fim da sequência que será incluída. O padrão é aumentar a sequência por incrementos de 1. Entretanto, um terceiro argumento nos permite determinar quanto incrementar:

```
seq(1, 10, 2)
#> [1] 1 3 5 7 9
```

Se quisermos números inteiros consecutivos, podemos usar a seguinte abreviação:

```
1:10
#> [1] 1 2 3 4 5 6 7 8 9 10
```

Quando usamos essas funções, R produz números do tipo *integer*, e não do tipo *numeric*, uma vez que são tipicamente usados para indexar algo:

```
class(1:10)
#> [1] "integer"
```

No entanto, se criarmos uma sequência que inclua números não-inteiros, a classe mudará:

```
class(seq(1, 10, 0.5))
#> [1] "numeric"
```

2.6.4 Criando subconjuntos

Usamos colchetes para acessar elementos específicos de um vetor. Para o vetor `codes` que definimos anteriormente, podemos acessar o segundo elemento usando:

```
codes[2]
#> canada
#> 124
```

Você pode obter mais de uma entrada usando um vetor de múltiplas entradas como índice:

```
codes[c(1,3)]
#> italy egypt
#> 380 818
```

As sequências definidas acima são particularmente úteis se precisarmos acessar, digamos, os dois primeiros elementos:

```
codes[1:2]
#> italy canada
#> 380 124
```

Se os elementos tiverem nomes, também podemos acessar as entradas usando esses nomes. Aqui estão dois exemplos:

```
codes["canada"]
#> canada
#> 124
codes[c("egypt","italy")]
#> egypt italy
#> 818 380
```

2.7 Conversão forçada

Em geral, uma conversão forçada (também denominada coerção ou no original *coercion*) é uma tentativa do R de ser flexível com os tipos de dados. Quando uma entrada não corresponde ao esperado, algumas das funções predefinidas do R tentam adivinhar o que se estava tentando fazer antes de retornar uma mensagem de erro. Isso também pode causar confusão. Não entendendo a conversão forçada, os programadores podem enlouquecer quando codificam em R, pois o R se comporta de maneira bem diferente da maioria das linguagens nesse sentido. Vamos aprender mais com alguns exemplos.

Dissemos que os vetores devem ser todos do mesmo tipo. Portanto, se tentarmos combinar, por exemplo, números e caracteres, você pode esperar um erro:

```
x <- c(1, "canada", 3)
```

Mas o código acima não retorna um erro. Nem mesmo um aviso! O que aconteceu? Observe `x` e sua classe:

```
x
#> [1] "1"      "canada" "3"
class(x)
#> [1] "character"
```

R forçou uma conversão dos dados em caracteres. Como inserimos uma sequência de caracteres no vetor, R adivinhou que nossa intenção era que 1 e 3 fossem as *strings* "1" e "3". O fato de nem sequer emitir um aviso é um exemplo de como a conversão forçada pode causar muitos erros que não serão percebidos no R.

R também oferece funções para mudar de um tipo para outro. Por exemplo, pode-se converter números em caracteres com:

```
x <- 1:5
y <- as.character(x)
y
#> [1] "1" "2" "3" "4" "5"
```

Pode-se reverter isso com `as.numeric`:

```
as.numeric(y)
#> [1] 1 2 3 4 5
```

Essa função é realmente bastante útil, uma vez que *datasets* que incluem tanto valores numéricos, quanto cadeias de caracteres, são comuns.

2.7.1 Valores NA

Quando uma função tenta forçar uma conversão de um tipo para outro e encontra um caso impossível, R geralmente nos dá um aviso e converte a entrada em um valor especial chamado `NA` o que significa “não disponível” (*Not Available* em inglês). Por exemplo:

```
x <- c("1", "b", "3")
as.numeric(x)
#> Warning: NAs introduced by coercion
#> [1] 1 NA 3
```

R não tem qualquer palpite sobre o número que você queria quando escreveu `b`, por isso ele nem tenta adivinhar.

Como cientista de dados, você encontrará `NAs` frequentemente, uma vez que tais valores, geralmente, são usados para representar dados ausentes, um problema comum em *datasets* do mundo real.

2.8 Exercícios

1. Use a função `c` para criar um vetor com as temperaturas médias altas em janeiro para Pequim, Lagos, Paris, Rio de Janeiro, San Juan e Toronto, com 35, 88, 42, 84, 81 e 30 graus Fahrenheit. Chame o objeto de `temp`.

2. Agora crie um vetor com os nomes das cidades e chame o objeto de `city`.
3. Use a função `names` e os objetos definidos nos exercícios anteriores para associar os dados de temperatura à cidade correspondente.
4. Use operadores `[` e `:` para acessar a temperatura das três primeiras cidades da lista.
5. Use o operador `[` para acessar a temperatura de Paris e San Juan.
6. Use o operador `:` para criar uma sequência de números $12, 13, 14, \dots, 73$.
7. Crie um vetor contendo todos os números ímpares positivos menores que 100.
8. Crie um vetor de números que comece em 6, não ultrapasse 55 e adicione números em incrementos de $4/7$: $6, 6 + 4/7, 6 + 8/7$, e assim por diante. Quantos números a lista possui? Dica: use `seq` e `length`.
9. Qual é a classe do seguinte objeto: `a <- seq(1, 10, 0.5)`?
10. Qual é a classe do seguinte objeto: `a <- seq(1, 10)`?
11. A classe de `class(a<-1)` é *numeric*, e não *integer*. R por padrão usa o tipo *numeric*. Para forçar o uso do tipo *integer*, você deve adicionar a letra `L`. Confirme se a classe de `1L` é *integer*.
12. Defina o seguinte vetor:

```
x <- c("1", "3", "5")
```

a seguir, converta esse vetor para o tipo *integer*.

2.9 Ordenação

Agora que dominamos alguns conhecimentos básicos de R, vamos tentar obter alguns *insights* sobre segurança em diferentes estados no contexto de assassinatos por armas de fogo.

2.9.1 sort

Digamos que desejemos classificar os estados, do menor para o maior, com base no índice de assassinatos por armas. A função `sort` classifica um vetor em ordem crescente. Portanto, podemos ver o maior número de assassinatos por arma, escrevendo:

```
library(dslabs)
data(murders)
sort(murders$total)
#> [1]  2   4   5   5   7   8   11  12  12  16  19  21  22
#> [14] 27  32  36  38  53  63  65  67  84  93  93  97  97
#> [27] 99 111 116 118 120 135 142 207 219 232 246 250 286
#> [40] 293 310 321 351 364 376 413 457 517 669 805 1257
```

Entretanto, isso não nos fornece informações vinculando quais estados têm quais índices de assassinatos. Por exemplo, não sabemos em que estado apresentou 1257 assassinatos.

2.9.2 order

A função `order` é mais apropriada para o que queremos fazer. `order` recebe um vetor como entrada e retorna um vetor de índices que classifica o vetor de entrada. Isso pode parecer um pouco confuso, então vamos analisar um exemplo simples. Podemos criar um vetor e ordená-lo:

```
x <- c(31, 4, 15, 92, 65)
sort(x)
#> [1] 4 15 31 65 92
```

Em vez de ordenar o vetor de entrada, a função `order` retorna o índice que classifica o vetor de entrada:

```
index <- order(x)
x[index]
#> [1] 4 15 31 65 92
```

Este é o mesmo resultado retornado por `sort(x)`. Se olharmos para os índices, veremos porque isso funciona:

```
x
#> [1] 31 4 15 92 65
order(x)
#> [1] 2 3 1 5 4
```

A segunda entrada de `x` é o menor valor, logo `order(x)` começa com o índice 2. O próximo menor valor é a terceira entrada, portanto, o segundo índice é 3, e assim por diante.

Como isso nos ajuda a classificar os estados por assassinato? Primeiro, lembre-se de que as entradas de vetores que você acessa com \$ seguem a mesma ordem que as linhas da tabela. Por exemplo, estes dois vetores que contêm os nomes dos estados e suas abreviações, respectivamente, seguem a mesma ordem:

```
murders$state[1:6]
#> [1] "Alabama"      "Alaska"       "Arizona"      "Arkansas"     "California"
#> [6] "Colorado"
murders$abb[1:6]
#> [1] "AL" "AK" "AZ" "AR" "CA" "CO"
```

Isso significa que podemos classificar os nomes dos estados com base no total de assassinatos. Primeiro, obtemos o índice que ordena os vetores pelo número total de assassinatos. A seguir, indexamos o vetor de nomes de estados:

```
ind <- order(murders$total)
murders$abb[ind]
#> [1] "VT" "ND" "NH" "WY" "HI" "SD" "ME" "ID" "MT" "RI" "AK" "IA" "UT"
#> [14] "WV" "NE" "OR" "DE" "MN" "KS" "CO" "NM" "NV" "AR" "WA" "CT" "WI"
#> [27] "DC" "OK" "KY" "MA" "MS" "AL" "IN" "SC" "TN" "AZ" "NJ" "VA" "NC"
#> [40] "MD" "OH" "MO" "LA" "IL" "GA" "MI" "PA" "NY" "FL" "TX" "CA"
```

De acordo com o código acima, a Califórnia teve o maior número de assassinatos.

2.9.3 max e which.max

Se estivermos interessados apenas na entrada com o valor mais alto, podemos usar `max` para obter esse valor:

```
max(murders$total)
#> [1] 1257
```

e `which.max` para o índice de valor mais alto:

```
i_max <- which.max(murders$total)
murders$state[i_max]
#> [1] "California"
```

Para determinar o menor valor, podemos usar `min` e `which.min` do mesmo modo.

Isso significa que a Califórnia é o estado mais perigoso? Em uma próxima seção, argumentamos que devemos considerar proporções e não totais. Antes de fazer isso, apresentaremos uma última função relacionada à ordem: `rank`.

2.9.4 rank

Embora não seja usada com tanta frequência quanto `order` e `sort`, a função `rank` também pode ser utilizada para classificação e pode ser bastante útil. Para qualquer vetor, `rank` retorna um outro vetor com a posição de classificação dos itens do vetor de entrada quando ordenados. Aqui está um exemplo simples:

```
x <- c(31, 4, 15, 92, 65)
rank(x)
#> [1] 3 1 2 5 4
```

Para resumir, vejamos os resultados das três funções que apresentamos:

	original	sort	order	rank
	31	4	2	3
	4	15	3	1
	15	31	1	2
	92	65	5	5
	65	92	4	4

2.9.5 Cuidado com a reciclagem

Outra fonte comum de erros que podem passar despercebidos no R é o uso de reciclagem (*recycling* no inglês). Vimos que os vetores são adicionados elemento a elemento. Portanto, se os vetores não coincidirem em comprimento, é natural supor que receberemos um erro. Mas esse não é o caso. Veja o que acontece:

```
x <- c(1, 2, 3)
y <- c(10, 20, 30, 40, 50, 60, 70)
x+y
#> Warning in x + y: longer object length is not a multiple of shorter
#> object length
#> [1] 11 22 33 41 52 63 71
```

Recebemos um aviso, mas não há erro. Para o resultado, R reciclou os números em `x`. Veja

o último dígito dos números na saída (ou seja, nessa soma a reciclagem faz `x` ser equivalente a `c(1, 2, 3, 1, 2, 3, 1)`).

2.10 Exercícios

Para os exercícios a seguir, usaremos o *dataset* de assassinatos nos EUA. Certifique-se de carregá-lo antes de começar.

```
library(dslabs)
data("murders")
```

1. Use o operador `$` para acessar os dados de tamanho da população e armazená-los em um objeto denominado `pop`. Então use a função `sort` ordenar `pop`. Por fim, use o operador `[` para indicar o menor tamanho populacional.
2. Agora, em vez de determinar o valor do menor tamanho populacional, encontre o índice da entrada com o menor tamanho populacional. Dica: use `order` ao invés de `sort`.
3. Podemos realizar a mesma operação que no exercício anterior usando a função `which.min`. Escreva uma linha de código que faça isso.
4. Agora sabemos o quão pequeno é o menor estado e qual linha o representa. Que estado é esse? Defina uma variável denominada `states` para receber os nomes dos estados do *data frame* `murders`. Digite o nome do estado com a menor população.
5. Você pode criar um *data frame* usando a função `data.frame`. Aqui está um exemplo:

```
temp <- c(35, 88, 42, 84, 81, 30)
city <- c("Beijing", "Lagos", "Paris", "Rio de Janeiro",
"San Juan", "Toronto")
city_temps <- data.frame(name = city, temperature = temp)
```

Use a função `rank` para determinar a faixa populacional de cada estado, do menos populoso ao mais populoso. Armazene esses intervalos em um objeto chamado `ranks`, e crie um *data frame* com o nome do estado e sua posição no `rank`. Nomeie o *data frame* como `my_df`.

6. Repita o exercício anterior, mas desta vez ordene `my_df` para que os estados estejam na ordem de menos populosos para os mais populosos. Dica: crie um objeto `ind` que armazena os índices necessários para ordenar os valores da população. Em seguida, use o operador de suporte `[` para reordenar cada coluna no *data frame*.
7. O vetor `na_example` representa uma série de contagens. Você pode navegar rapidamente pelo objeto usando:

```
data("na_example")
str(na_example)
#>  int [1:1000] 2 1 3 2 1 3 1 4 3 2 ...
```

No entanto, quando calculamos a média com a função `mean`, obtemos um `NA`:

```
mean(na_example)
#> [1] NA
```

A função `is.na` retorna um vetor lógico que nos diz quais entradas são `NA`. Atribua esse vetor lógico a um objeto chamado `ind` e determine quantas entradas `NA` existem em `na_example`.

8. Agora calcule a média novamente apenas para as entradas que não são `NA`. Dica: lembre-se do operador `!`.

2.11 Aritmética vetorial

A Califórnia teve mais assassinatos, mas isso significa que é o estado mais perigoso? E se ela apenas possuir muito mais pessoas do que qualquer outro estado? Podemos confirmar rapidamente que a Califórnia tem a maior população:

```
library(dslabs)
data("murders")
murders$state[which.max(murders$population)]
#> [1] "California"
```

com mais do que 37 milhões de habitantes. Portanto, é injusto comparar os totais se estivermos interessados em saber quão seguro é o estado. O que realmente devemos calcular são os assassinatos per capita. Os relatórios que descrevemos na seção motivadora usam assassinatos por 100.000 habitantes como unidade. Para calcular essa quantidade, usamos os poderosos recursos aritméticos vetoriais de R.

2.11.1 Reescalonando um vetor

Em R, operações aritméticas em vetores ocorrem elemento a elemento. Como exemplo, suponha que tenhamos as seguintes alturas em polegadas (1 polegada é equivalente a 2,54 centímetros):

```
inches <- c(69, 62, 66, 70, 70, 73, 67, 73, 67, 70)
```

e queremos convertê-las em centímetros. Observe o que acontece quando multiplicamos `inches` por 2,54:

```
inches * 2.54
#> [1] 175 157 168 178 178 185 170 185 170 178
```

Acima, multiplicamos cada elemento por 2,54. Da mesma forma, se para cada entrada queremos calcular o quanto cada uma é maior ou menor que 69 polegadas (a altura média para homens), podemos subtraí-las de cada entrada desta maneira:

```
inches - 69
#> [1] 0 -7 -3 1 1 4 -2 4 -2 1
```

2.11.2 Dois vetores

Se tivermos dois vetores de mesmo comprimento e os somá-los usando R, eles serão somados entrada por entrada da seguinte maneira:

$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} + \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix} = \begin{pmatrix} a+e \\ b+f \\ c+g \\ d+h \end{pmatrix}$$

O mesmo se aplica a outras operações matemáticas, como `-`, `*` e `/`.

Isso implica que, para calcular as taxas de homicídio (*murder rates* em inglês), podemos simplesmente escrever:

```
murder_rate <- murders$total / murders$population * 100000
```

Ao fazer isso, percebemos que a Califórnia não está mais perto do topo da lista. De fato, podemos usar o que aprendemos para colocar os estados em ordem com base na taxa de homicídios:

```
murders$abb[order(murder_rate)]
#> [1] "VT" "NH" "HI" "ND" "IA" "ID" "UT" "ME" "WY" "OR" "SD" "MN" "MT"
#> [14] "CO" "WA" "WV" "RI" "WI" "NE" "MA" "IN" "KS" "NY" "KY" "AK" "OH"
#> [27] "CT" "NJ" "AL" "IL" "OK" "NC" "NV" "VA" "AR" "TX" "NM" "CA" "FL"
#> [40] "TN" "PA" "AZ" "GA" "MS" "MI" "DE" "SC" "MD" "MO" "LA" "DC"
```

2.12 Exercícios

1. Criamos anteriormente este *data frame*:

```
temp <- c(35, 88, 42, 84, 81, 30)
city <- c("Beijing", "Lagos", "Paris", "Rio de Janeiro",
"San Juan", "Toronto")
city_temps <- data.frame(name = city, temperature = temp)
```

Recrie o *data frame* usando o código acima, mas adicione uma linha que converta a temperatura de Fahrenheit em Celsius. A conversão é $C = \frac{5}{9} \times (F - 32)$.

2. Qual é o resultado da seguinte soma? $1 + 1/2^2 + 1/3^2 + \dots 1/100^2$? Dica: Graças a Euler, sabemos que o resultado deve ser próximo de $\pi^2/6$.

3. Calcule a taxa de homicídios por 100.000 habitantes para cada estado e armazene-a no objeto `murder_rate`. Em seguida, encontre a taxa média de homicídios nos EUA com função `mean`. Qual é a média?

2.13 Indexação

R fornece uma maneira poderosa e conveniente para indexar vetores. Podemos, por exemplo, criar um subconjunto de um vetor com base nas propriedades de outro vetor. Nesta seção, continuaremos a trabalhar em nosso exemplo de assassinato nos EUA, que podemos carregar assim:

```
library(dslabs)
data("murders")
```

2.13.1 Criando subconjuntos com objetos do tipo lógico

Agora calculamos a taxa de homicídios usando:

```
murder_rate <- murders$total / murders$population * 100000
```

Imagine se mudar da Itália onde, segundo uma reportagem do canal de notícias ABC News, a taxa de homicídios é de apenas 0,71 por 100.000. Você preferiria se mudar para um estado com uma taxa de homicídios semelhante. Outro recurso poderoso do R é que podemos usar operadores lógicos para indexar vetores. Se compararmos um vetor com um único número, R realiza o teste para cada entrada. Aqui está um exemplo relacionado à questão anterior:

```
ind <- murder_rate < 0.71
```

Se, em vez disso, queremos saber se um valor é menor ou igual, podemos usar:

```
ind <- murder_rate <= 0.71
```

Note que recuperamos um vetor lógico com TRUE para cada entrada menor ou igual a 0,71. Para ver quais são esses estados, podemos tirar proveito do fato de que vetores podem ser indexados com objetos do tipo lógico.

```
murders$state[ind]
#> [1] "Hawaii"           "Iowa"                 "New Hampshire" "North Dakota"
#> [5] "Vermont"
```

Para contar quantas entradas são verdadeiras (*TRUE*), a função `sum` retorna a soma das entradas de um vetor e força a conversão de vetores lógicos em números com *TRUE* codificado como 1 e *FALSE* como 0. Portanto, podemos contar os estados usando:

```
sum(ind)
#> [1] 5
```

2.13.2 Operadores lógicos

Suponha que gostemos de montanhas e que desejemos mudar para um estado seguro na região oeste do país. Queremos que a taxa de homicídios seja no máximo 1. Nesse caso, queremos que duas coisas diferentes sejam verdadeiras. Aqui podemos usar o operador lógico *and*, que em R é representado por `&`. Esta operação resulta em *TRUE* somente quando ambas as lógicas são *TRUE*, isto é, verdadeiras. Para ver isso, considere este exemplo:

```
TRUE & TRUE
#> [1] TRUE
TRUE & FALSE
#> [1] FALSE
FALSE & FALSE
#> [1] FALSE
```

Para o nosso exemplo, podemos formar duas condições lógicas:

```
west <- murders$region == "West"
safe <- murder_rate <= 1
```

e podemos usar o `&` para obter um vetor lógico que nos diz quais estados satisfazem ambas as condições:

```
ind <- safe & west
murders$state[ind]
#> [1] "Hawaii"   "Idaho"    "Oregon"   "Utah"     "Wyoming"
```

2.13.3 which

Suponha que queremos ver a taxa de homicídios da Califórnia. Para esse tipo de operação, é conveniente converter vetores lógicos em índices em vez de manter vetores lógicos longos. A função `which` nos diz quais entradas de um vetor lógico são verdadeiras. Então podemos escrever:

```
ind <- which(murders$state == "California")
murder_rate[ind]
#> [1] 3.37
```

2.13.4 match

Se, em vez de um único estado, desejamos descobrir as taxas de assassinatos de vários estados, digamos Nova York, Flórida e Texas, podemos usar a função `match`. Essa função nos diz quais índices de um segundo vetor correspondem a cada uma das entradas de um primeiro vetor:

```
ind <- match(c("New York", "Florida", "Texas"), murders$state)
ind
#> [1] 33 10 44
```

Agora podemos ver as taxas de homicídio:

```
murder_rate[ind]
#> [1] 2.67 3.40 3.20
```

2.13.5 %in%

Se, em vez de um índice, desejamos que uma expressão lógica nos diga se cada elemento de um primeiro vetor está em um segundo vetor, podemos usar a função `%in%`. Vamos imaginar que você não tem certeza se Boston, Dakota e Washington são estados. Você pode descobrir assim:

```
c("Boston", "Dakota", "Washington") %in% murders$state
#> [1] FALSE FALSE TRUE
```

Lembre-se de que usaremos `%in%` frequentemente ao longo do livro.

Avançado: existe uma conexão entre `match` e `%in%` através de `which`. Para ver isso, observe que as duas linhas a seguir produzem o mesmo índice (embora em ordem diferente):

```
match(c("New York", "Florida", "Texas"), murders$state)
#> [1] 33 10 44
which(murders$state %in% c("New York", "Florida", "Texas"))
#> [1] 10 33 44
```

2.14 Exercícios

Comece carregando a biblioteca e os dados.

```
library(dslabs)
data(murders)
```

1. Calcule a taxa de homicídios por 100.000 habitantes para cada estado e armazene-a em um objeto chamado `murder_rate`. Em seguida, use operadores lógicos para criar um vetor lógico chamado `low` que nos diz quais entradas de `murder_rate` são menores que 1.
 2. Agora use os resultados do exercício anterior e a função `which` para determinar os índices de `murder_rate` associados a valores menores que 1.
 3. Use os resultados do exercício anterior para indicar os nomes dos estados com taxas de homicídio menores que 1.
 4. Agora estenda o código dos Exercícios 2 e 3 para indicar os estados do nordeste com taxas de homicídios menores que 1. Dica: Use o vetor lógico predefinido `low` e o operador lógico `&`.
 5. Em um exercício anterior, calculamos a taxa de homicídios de cada estado e a média desses números. Quantos estados estão abaixo da média?
 6. Use a função `match` para identificar estados com as abreviações AK, MI e IA. Dica: Comece definindo um índice das entradas em `murders$abb` que correspondem às três abreviações. Então use o operador `[` para extrair os estados.
 7. Use o operador `%in%` para criar um vetor lógico que responda à pergunta: quais das seguintes abreviações são reais: MA, ME, MI, MO, MU?
 8. Estenda o código usado no Exercício 7 para descobrir a única entrada que **não** é uma abreviação real. Dica: use o operador `!`, que converte FALSE para TRUE e vice-versa, e depois `which` para obter um índice.
-

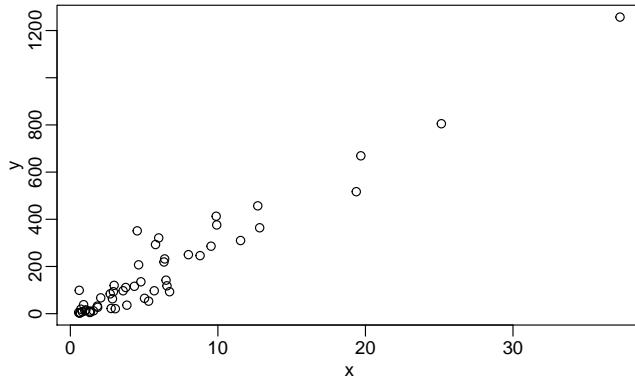
2.15 Gráficos básicos

No capítulo 7 descrevemos um pacote complementar que oferece uma abordagem poderosa para a produção de gráficos (*plots* em inglês) em R. Em seguida, temos uma parte completa sobre “Visualização de dados”, na qual oferecemos muitos exemplos. Aqui, descrevemos brevemente algumas das funções disponíveis em uma instalação básica do R.

2.15.1 `plot`

A função `plot` pode ser usada para criar diagramas de dispersão (*scatterplots* em inglês). Aqui está um gráfico do total de assassinatos versus população.

```
x <- murders$population/ 10^6
y <- murders$total
plot(x, y)
```



Para criar um gráfico rápido que não acessa variáveis duas vezes, podemos usar a função `with`:

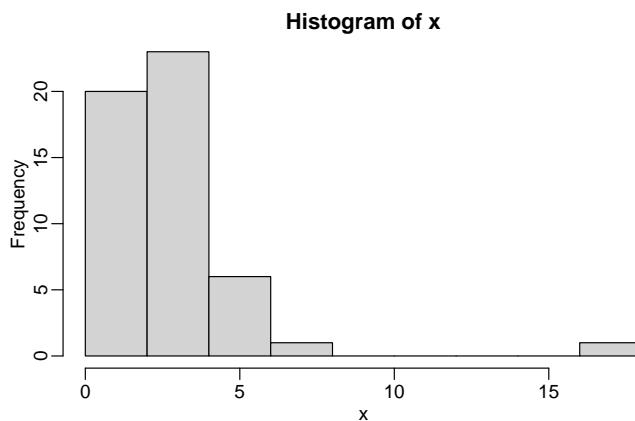
```
with(murders, plot(population, total))
```

A função `with` nos permite usar os nomes das colunas de `murders` na função `plot`. Também funciona com qualquer *data frame* e qualquer função.

2.15.2 `hist`

Vamos descrever os histogramas relacionados à distribuição na seção de “Visualização de dados” deste livro. Aqui, observaremos simplesmente que os histogramas são um resumo gráfico poderoso de uma lista de números que fornece uma visão geral dos tipos de valores que você possui. Podemos fazer um histograma de nossas taxas de assassinatos simplesmente digitando:

```
x <- with(murders, total/ population * 100000)
hist(x)
```



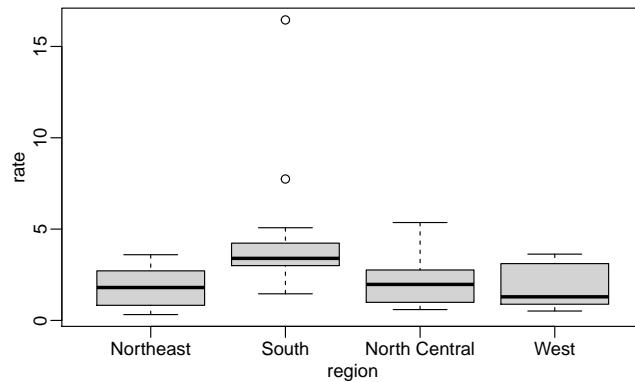
Podemos ver que há uma grande variedade de valores, com a maioria entre 2 e 3, e um caso muito extremo com uma taxa de homicídios acima de 15:

```
murders$state[which.max(x)]
#> [1] "District of Columbia"
```

2.15.3 boxplot

Os diagramas de caixa (*boxplots* em inglês) também serão descritos na parte “Visualização de dados” do livro. Eles fornecem um resumo mais conciso do que os histogramas, mas são mais fáceis de comparar com outros *boxplots*. Por exemplo, aqui podemos usá-los para comparar diferentes regiões:

```
murders$rate <- with(murders, total/ population * 100000)
boxplot(rate~region, data = murders)
```

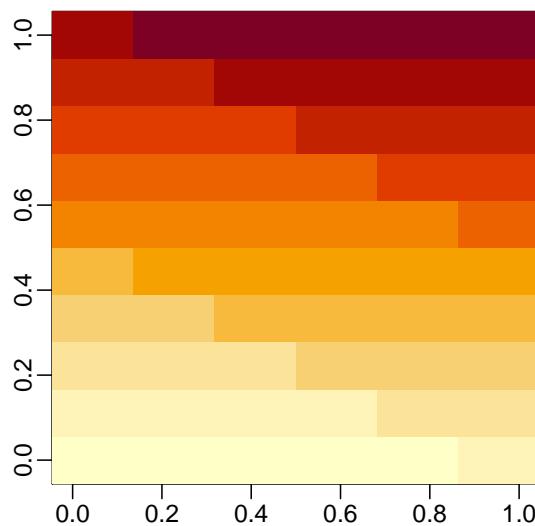


Podemos ver que o Sul tem taxas mais altas de assassinatos do que as outras três regiões.

2.15.4 image

A função *image* exibe os valores em uma matriz usando cores. Aqui está um exemplo rápido:

```
x <- matrix(1:120, 12, 10)
image(x)
```



2.16 Exercícios

1. Fizemos um gráfico do total de assassinatos versus população e notamos um forte relacionamento. Não é de se surpreender que os estados com populações maiores tenham mais assassinatos.

```
library(dslabs)
data(murders)
population_in_millions <- murders$population/10^6
total_gun_murders <- murders$total
plot(population_in_millions, total_gun_murders)
```

Lembre-se de que muitos estados têm populações inferiores a 5 milhões e estão agrupados. Podemos obter mais informações criando esse gráfico na escala logarítmica. Transforme as variáveis usando transformação `log10` e depois crie um gráfico com os resultados.

2. Crie um histograma das populações dos estados.
3. Gere *boxplots* das populações estaduais por região.

3

Conceitos básicos de programação

Ensinamos R pelo fato de ele facilitar significativamente a análise de dados, o tema principal deste livro. Ao programar em R, podemos executar eficientemente a análise exploratória de dados, criar fluxos de análise de dados e preparar a visualização de dados para comunicar resultados. No entanto, R não é apenas um ambiente de análise de dados, mas uma linguagem de programação. Programadores avançados de R podem desenvolver pacotes complexos e até melhorar o R, embora não abordemos estes tópicos neste livro. No entanto, nesta seção, apresentamos três conceitos-chave de programação: expressões condicionais, execução em laço do tipo `for` e funções. Estes não são apenas os principais componentes da programação avançada, mas, às vezes, são úteis durante a análise de dados. Também observamos que existem várias funções amplamente usadas para programação em R, mas que não discutiremos neste livro. Elas incluem `split`, `cut`, `do.call` e `Reduce`, bem como o pacote `data.table`. Vale a pena aprender como usá-las, se você quiser se tornar um programador especialista em R.

3.1 Expressões condicionais

Expressões condicionais são uma das características básicas da programação. Elas são usados para o que é chamado de controle de fluxo. A expressão condicional mais comum é a instrução `if-else`. Em R, podemos fazer muitas análises de dados sem expressões condicionais. No entanto, eles aparecem ocasionalmente e você precisará delas assim que começar a escrever suas próprias funções e pacotes.

Aqui está um exemplo muito simples que mostra a estrutura geral de uma instrução `if-else`. A idéia básica é retornar o inverso de `a`, a menos que `a` seja 0:

```
a <- 0

if(a!=0){
  print(1/a)
} else{
  print("Não há inverso de 0.")
}
#> [1] "Não há inverso de 0."
```

Vejamos outro exemplo usando o conjunto de dados de assassinatos nos EUA:

```
library(dslabs)
data(murders)
murder_rate <- murders$total/ murders$population*100000
```

Aqui está um exemplo muito simples que nos diz quais estados, se houver, têm uma taxa

de homicídios inferior a 0,5 por 100.000 habitantes. As declarações `if` protegem-nos de situações em que nenhum estado satisfaz a condição.

```
ind <- which.min(murder_rate)

if(murder_rate[ind] < 0.5){
  print(murders$state[ind])
} else{
  print("Nenhum estado tem taxa de homicídios abaixo deste nível")
}
#> [1] "Vermont"
```

Se tentarmos novamente com uma taxa de 0,25, obteremos uma resposta diferente:

```
if(murder_rate[ind] < 0.25){
  print(murders$state[ind])
} else{
  print("Nenhum estado tem taxa de homicídios abaixo deste nível")
}
#> [1] "Nenhum estado tem taxa de homicídios abaixo deste nível"
```

Uma função relacionada que é muito útil é `ifelse`. Ela usa três argumentos: um lógico e duas respostas possíveis. Se o lógico for `TRUE`, retorna o valor no segundo argumento e, se for `FALSE`, retorna o valor no terceiro argumento. Aqui está um exemplo:

```
a <- 0
ifelse(a > 0, 1/a, NA)
#> [1] NA
```

Essa função é particularmente útil porque é otimizada para vetores. Ela examina cada entrada do vetor lógico e retorna elementos do vetor fornecido no segundo argumento, se a entrada for `TRUE`, ou elementos do vetor fornecido no terceiro argumento, se a entrada for `FALSE`.

```
a <- c(0, 1, 2, -4, 5)
result <- ifelse(a > 0, 1/a, NA)
```

Esta tabela nos ajuda a ver o que aconteceu:

a	is_a_positive	answer1	answer2	result
0	FALSE	Inf	NA	NA
1	TRUE	1.00	NA	1.0
2	TRUE	0.50	NA	0.5
-4	FALSE	-0.25	NA	NA
5	TRUE	0.20	NA	0.2

Aqui está um exemplo de como essa função pode ser facilmente usada para substituir todos os valores ausentes em um vetor por zeros:

```
data(na_example)
no_nas <- ifelse(is.na(na_example), 0, na_example)
sum(is.na(no_nas))
#> [1] 0
```

Duas outras funções úteis são `any` e `all`. A função `any` pega um vetor com valores lógicos

e retorna TRUE se alguma das entradas for TRUE. A função `all` pega um vetor de lógicas e retorna TRUE se todas as entradas forem TRUE. Aqui está um exemplo:

```
z <- c(TRUE, TRUE, FALSE)
any(z)
#> [1] TRUE
all(z)
#> [1] FALSE
```

3.2 Como definir funções

À medida que ganha mais experiência, você observará que realizará as mesmas operações repetidamente. Um exemplo simples é o cálculo de médias. Podemos calcular a média de um vetor `x` usando funções `sum` e `length`: `sum(x)/length(x)`. Como fazemos isso repetidamente, é muito mais eficiente escrever uma função que execute essa operação. Essa operação específica é tão comum que alguém já escreveu a função `mean`, que está incluída na pacote `base` do R. No entanto, você encontrará situações em que a função ainda não existe e o R permite que você escreva uma. Uma versão simplificada de uma função que calcula a média pode ser implementada da seguinte maneira:

```
avg <- function(x){
  s <- sum(x)
  n <- length(x)
  s/n
}
```

Agora `avg` é uma função que calcula a média:

```
x <- 1:100
identical(mean(x), avg(x))
#> [1] TRUE
```

Observe que as variáveis definidas em uma função não são gravadas no espaço de trabalho. Então, enquanto usamos `s` e `n` quando chamamos (*call* em inglês) `avg`, os valores são criados e alterados apenas durante a chamada. Aqui podemos ver um exemplo ilustrativo:

```
s <- 3
avg(1:10)
#> [1] 5.5
s
#> [1] 3
```

Note como `s` ainda é 3 depois que executa-se `avg`.

Em geral, funções são objetos, portanto, atribuímos nomes de variáveis a eles com `<-`. A função `function` diz ao R que ele está prestes a definir uma função. A forma geral da definição de uma função é assim:

```
my_function <- function(VARIABLE_NAME){
  perform operations on VARIABLE_NAME and calculate VALUE
  VALUE
}
```

As funções que você define podem ter vários argumentos, bem como valores padrão. Por exemplo, podemos definir uma função que calcula a média aritmética ou geométrica, dependendo de uma variável definida pelo usuário como esta:

```
avg <- function(x, arithmetic = TRUE){
  n <- length(x)
  ifelse(arithmetic, sum(x)/n, prod(x)^(1/n))
}
```

Aprenderemos mais sobre como criar funções através da experiência, à medida em que avançamos para tarefas mais complexas.

3.3 Namespaces

À medida em que você se torna mais experiente em R, é mais provável que você precise carregar vários pacotes para algumas de suas análises. Quando começar a fazer isso, você observará que é possível que dois pacotes diferentes utilizem o mesmo nome para duas funções distintas. E muitas vezes essas funções fazem coisas completamente diferentes. Na verdade, você já encontrou este problema, visto que ambos os pacotes **stats** e **dplyr** definem uma função **filter**. Existem outros cinco exemplos em **dplyr**. Sabemos disso porque, quando carregamos **dplyr** pela primeira vez, vemos a seguinte mensagem:

The following objects are masked from ‘package:stats’:

filter, **lag**

The following objects are masked from ‘package:base’:

intersect, **setdiff**, **setequal**, **union**

Então, o que R faz quando digitamos **filter**? Ele usa a função definida no **dplyr** ou a do **stats**? Por nossa experiência anterior, sabemos que ele usa **dplyr**. Mas e se quisermos usar a do **stats**?

Essas funções vivem em diferentes *namespaces*. O R seguirá uma determinada ordem ao procurar uma função nesses *namespaces*. Você pode ver a ordem digitando:

```
search()
```

A primeira entrada nesta lista é o ambiente global que inclui todos os objetos que você define.

E se quisermos usar o **filter** definido no pacote **stats** ao invés daquela definida no **dplyr**, mesmo com o **dplyr** aparecendo primeiro na lista de pesquisa? Você pode forçar o uso de um *namespace* específico usando dois pontos duplos (::), assim:

```
stats::filter
```

Se quisermos ter certeza absoluta de que utilizaremos o **filter** do **dplyr**, podemos usar:

```
dplyr::filter
```

Lembre-se de que, se quisermos usar uma função de um pacote sem carregá-lo inteiramente, também podemos usar dois pontos duplos.

Para obter mais informações sobre esse tópico mais avançado, recomendamos o livro *R packages*¹.

3.4 Laços do tipo for

A fórmula para a soma da série $1 + 2 + \dots + n$ é $n(n + 1)/2$. E se não tivéssemos certeza de que essa era a função correta? Como poderíamos verificar? Usando o que aprendemos sobre funções, podemos criar uma que calcule S_n :

```
compute_s_n <- function(n){
  x <- 1:n
  sum(x)
}
```

Como podemos calcular S_n para vários valores de n , digamos $n = 1, \dots, 25$? Escrevemos 25 linhas de código executando `compute_s_n`? Não. É para isso que serve a execução em laço (*loop*). Nesse caso, a mesma tarefa é realizada repetidamente, e a única coisa muda é o valor de n . Os laços do tipo *for* permitem-nos definir o intervalo de mudança da nossa variável (no nosso exemplo $n = 1, \dots, 10$), alterar o valor e reavaliar a expressão.

Talvez o exemplo mais simples de um loop for seja esse código inútil:

```
for(i in 1:5){
  print(i)
}
#> [1] 1
#> [1] 2
#> [1] 3
#> [1] 4
#> [1] 5
```

Aqui está o laço do tipo *for* para o nosso exemplo S_n :

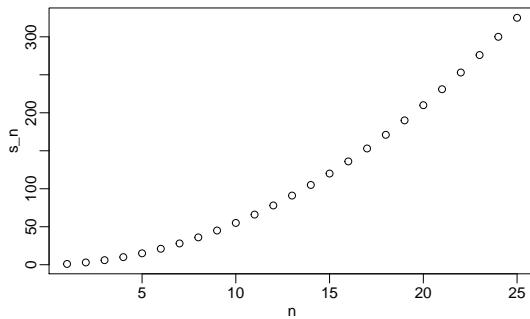
```
m <- 25
s_n <- vector(length = m) # create an empty vector
for(n in 1:m){
  s_n[n] <- compute_s_n(n)
}
```

Em cada iteração $n = 1, n = 2$, etc ..., calculamos S_n e mantemos na entrada n do `s_n`.

Agora podemos criar um gráfico para procurar um padrão:

```
n <- 1:m
plot(n, s_n)
```

¹<http://r-pkgs.had.co.nz/namespaces.html>



Se você percebeu que parece quadrático, está no caminho certo porque a fórmula é $n(n+1)/2$.

3.5 Vectorização e funcionais

Embora os laços do tipo *for* sejam um conceito importante para entender, eles não são muito usados em R. À medida em que você aprende mais R, descobre que vetorização é preferível aos laços/loops, pois resulta em um código mais curto e claro. Já vimos exemplos na seção de aritmética vetorial. Uma função vetorizada é uma função que aplicará a mesma operação a cada um dos seus elementos.

```
x <- 1:10
sqrt(x)
#> [1] 1.00 1.41 1.73 2.00 2.24 2.45 2.65 2.83 3.00 3.16
y <- 1:10
x*y
#> [1] 1 4 9 16 25 36 49 64 81 100
```

Para fazer esse cálculo, não precisamos de loops de for. No entanto, nem todas as funções funcionam dessa maneira. Por exemplo, a função que acabamos de escrever, `compute_s_n`, não é vetorizada, pois espera um escalar. Por exemplo, o código abaixo não executa a função para todas as entradas de `n`:

```
n <- 1:25
compute_s_n(n)
```

Os funcionais são funções que nos ajudam a aplicar a mesma função a cada entrada em um vetor, matriz, *data.frame* ou lista. Aqui, abordamos o funcional que opera em vetores numéricos, lógicos e de caracteres: `sapply`.

A função `sapply` nos permite executar operações na base de elemento a elemento (*element-wise*, em inglês) em qualquer função. Aqui podemos ver como funciona:

```
x <- 1:10
sapply(x, sqrt)
#> [1] 1.00 1.41 1.73 2.00 2.24 2.45 2.65 2.83 3.00 3.16
```

Cada elemento de `x` é passado para a função `sqrt` e retorna o resultado. Esses resultados são concatenados. Nesse caso, o resultado é um vetor do mesmo comprimento que o original, `x`. Isso implica que o loop *for* acima pode ser escrito da seguinte maneira:

```
n <- 1:25
s_n <- sapply(n, compute_s_n)
```

Outros funcionais são `apply`, `lapply`, `tapply`, `mapply`, `vapply` e `replicate`. Utilizamos principalmente `sapply`, `apply` e `replicate` neste livro, mas recomendamos os demais sejam estudados, pois podem ser muito úteis.

3.6 Exercícios

1. O que essa expressão condicional retornará?

```
x <- c(1,2,-3,4)
```

```
if(all(x>0)){
  print("All Postives")
} else{
  print("Not all positives")
}
```

2. Qual das seguintes expressões é sempre FALSE quando pelo menos uma entrada de um vetor lógico `x` é TRUE?

- a. `all(x)`
- b. `any(x)`
- c. `any(!x)`
- d. `all(!x)`

3. A função `nchar` informa quantos caracteres um vetor de caracteres possui. Escreva uma linha de código que atribua o objeto `new_names` a abreviação de estado quando o nome do estado tiver mais de 8 caracteres.

4. Crie uma função `sum_n` que, por qualquer valor, digamos `n`, calcule a soma dos números inteiros de 1 a `n` (inclusive). Use a função para determinar a soma dos números inteiros de 1 a 5.000.

5. Crie uma função `altman_plot` que possua dois argumentos, `x` e `y`, e faça um gráfico da diferença em relação à soma.

6. Depois de executar o código a seguir, qual é o valor de `x`?

```
x <- 3
my_func <- function(y){
  x <- 5
  y + 5
}
```

7. Escreva uma função `compute_s_n` que, para qualquer `n`, calcule a soma $S_n = 1^2 + 2^2 + 3^2 + \dots n^2$. Qual é o valor da soma quando `n = 10`?

8. Definir um vetor numérico vazio, `s_n`, de tamanho 25, usando `s_n <-`

`vector("numeric", 25)` e armazene os resultados de S_1, S_2, \dots, S_{25} , usando um laço do tipo `for`.

9. Repita o Exercício 8, mas, desta vez, use `sapply`.
10. Repita o Exercício 8, mas, desta vez, use `map_dbl`.
11. Apresente um gráfico de S_n versus n , para $n = 1, \dots, 25$.
12. Confirme que a fórmula para esta soma é $S_n = n(n + 1)(2n + 1)/6$.

4

tidyverse

Até agora, manipulamos vetores reorganizando-os e criando subconjuntos indexando. No entanto, quando iniciamos as análises mais avançadas, a unidade preferida para armazenamento de dados não é o vetor, mas o *data frame*. Neste capítulo, aprenderemos a trabalhar diretamente com *data frames*, o que facilita muito a organização da informação. Usaremos *data frames* para a maior parte deste livro. Vamos nos concentrar em um formato de dados específico chamado *tidy e uma coleção específica de pacotes que são particularmente úteis para trabalhar com dados tidy chamados tidyverse*.

Podemos carregar todos os pacotes `_tidyverse` de uma só vez instalando e carregando o pacote `tidyverse`:

```
library(tidyverse)
```

Aprendemos como implementar a abordagem *tidyverse* ao longo do livro, mas antes de nos aprofundarmos nos detalhes, neste capítulo, apresentaremos alguns dos aspectos mais comuns do *tidyverse*, começando com o pacote `dplyr` para manipular os quadros de dados e o pacote `purrr` para trabalhar com As funções. Observe que o *tidyverse* também inclui um pacote gráfico, `ggplot2`, que apresentaremos mais adiante neste capítulo. 7 na parte “Visualização de dados” do livro; o pacote `readr` discutido no capítulo 5; e muitos outros. Neste capítulo, primeiro apresentamos o conceito de dados *tidy e depois demonstramos como usamos tidyverse para trabalhar com data frames* neste formato.

4.1 Data *tidy*

Dizemos que uma tabela de dados está no formato *tidy* se cada linha representa uma observação e as colunas representam as diferentes variáveis disponíveis para cada uma dessas observações. O conjunto de dados `murders` é um exemplo de um *tidy data frame*.

```
#>      state abb region population total
#> 1    Alabama AL   South  4779736   135
#> 2     Alaska AK   West   710231    19
#> 3   Arizona AZ   West  6392017   232
#> 4  Arkansas AR   South  2915918    93
#> 5 California CA   West 37253956  1257
#> 6 Colorado CO   West  5029196    65
```

Cada linha representa um estado com cada uma das cinco colunas fornecendo uma variável diferente relacionada a esses estados: nome, abreviação, região, população e número total de assassinatos.

Para ver como as mesmas informações podem ser fornecidas em diferentes formatos, considere o seguinte exemplo:

```
#>      country year fertility
#> 1    Germany 1960     2.41
#> 2 South Korea 1960     6.16
#> 3    Germany 1961     2.44
#> 4 South Korea 1961     5.99
#> 5    Germany 1962     2.47
#> 6 South Korea 1962     5.79
```

Este conjunto de dados *tidy* oferece taxas de fertilidade para dois países ao longo dos anos. Um conjunto de dados arrumado é considerado porque cada linha apresenta uma observação com as três variáveis: país, ano e taxa de fertilidade. No entanto, esse conjunto de dados originalmente veio em um formato diferente e o remodelamos para distribuição através do pacote **dslabs**. Originalmente, os dados estavam no seguinte formato:

```
#>      country 1960 1961 1962
#> 1    Germany 2.41 2.44 2.47
#> 2 South Korea 6.16 5.99 5.79
```

A mesma informação é fornecida, mas há duas diferenças importantes no formato: 1) cada linha inclui várias observações e 2) uma das variáveis, ano, é armazenada no cabeçalho. Para que os pacotes *tidyverse* sejam utilizados da melhor maneira, precisamos alterar a forma dos dados para que estejam no formato *tidy*, que eles aprenderão na seção “*Wrangling data*” do livro. Até lá, usaremos exemplos de conjuntos de dados que já estão no formato *tidy*.

Embora não seja imediatamente óbvio, ao longo do livro, você começará a apreciar os benefícios de trabalhar em uma estrutura em que as funções usam os formatos *tidy_for_inputs_and_outputs*. Você verá como isso permite que os analistas de dados se concentrem nos aspectos mais importantes da análise, e não no formato dos dados.

4.2 Exercícios

1. Examine o conjunto de dados incluído na base R **co2**. Qual dos seguintes é verdadeiro?

para. **co2** *tidy* data: possui um ano para cada linha. b. **co2** not *tidy*: precisamos de pelo menos uma coluna com um vetor de caracteres. c. **co2** não é *tidy*: é uma matriz em vez de um *data frame*. d. **co2** não é arrumado: para arrumar, teríamos que mudar a forma (em inglês) para ter três colunas (ano, mês e valor), e então cada observação de CO2 teria uma linha.

2. Examine o conjunto de dados incluído na base R **ChickWeight**. Qual dos seguintes é verdadeiro?

para. **ChickWeight** não *tidy*: cada pintinho tem mais de uma linha. b. **ChickWeight** is *_tidy_*: cada observação (um peso) é representada por uma linha. O pintinho de onde essa medição veio é uma das variáveis. c. **ChickWeight** não é arrumado: estamos perdendo a coluna do ano. d. **ChickWeight** é *tidy*: é armazenado em um *data frame*.

3. Examine o conjunto de dados predefinido **BOD**. Qual dos seguintes é verdadeiro?

para. **BOD** não é arrumado: tem apenas seis linhas. b. **BOD** not *tidy*: a primeira coluna é

apenas um índice. c. `BOD` é *tidy*: cada linha é uma observação com dois valores (tempo e demanda) d. `BOD` é *tidy*: todos os conjuntos de dados pequenos são *tidy* por definição.

4. Qual dos seguintes conjuntos de dados internos é *tidy*? Você pode escolher mais de um.
 para. `BJsales` b. `EuStockMarkets` c. `DNase` d. `Formaldehyde` e. `Orange` f. `UCBAdmissions`
-

4.3 Como manipular *data frames*

O pacote `dplyr` oferece funções que executam algumas das operações mais comuns ao trabalhar com *data frames* e usa nomes para essas funções que são relativamente fáceis de lembrar. Por exemplo, para alterar a tabela de dados adicionando uma nova coluna, usamos `mutate`. Para filtrar a tabela de dados para um subconjunto de linhas, usamos `filter`. Por fim, para subdividir os dados selecionando colunas específicas, usamos `select`.

4.3.1 Como adicionar uma coluna com `mutate`

Queremos que todas as informações necessárias para nossa análise sejam incluídas na tabela de dados. Portanto, a primeira tarefa é adicionar as taxas de assassinato ao nosso quadro de dados de assassinatos. A função `mutate` pegue *data frame* como primeiro argumento e nome da variável e valores como segundo argumento usando a convenção `name = values`. Então, para adicionar taxas de assassinatos, usamos:

```
library(dslabs)
data("murders")
murders <- mutate(murders, rate = total/ population * 100000)
```

Lembre-se que aqui usamos `total` e `population` dentro da função, que são objetos **não** definidos em nossa área de trabalho. Mas por que não recebemos um erro?

Este é um dos principais recursos do `dplyr`. As funções neste pacote, como `mutate` eles sabem como procurar variáveis no *data frame* que o primeiro argumento fornece. Na chamada para `mutate` que vemos acima, `total` terá os valores de `murders$total`. Essa abordagem torna o código muito mais legível.

Podemos ver que a nova coluna foi adicionada:

```
head(murders)
#>      state abb region population total rate
#> 1   Alabama  AL   South    4779736  135 2.82
#> 2   Alaska  AK   West     710231   19 2.68
#> 3  Arizona  AZ   West    6392017  232 3.63
#> 4 Arkansas  AR   South   2915918   93 3.19
#> 5 California CA   West   37253956  1257 3.37
#> 6 Colorado  CO   West   5029196   65 1.29
```

Embora tenhamos substituído o objeto original `murders`, isso não altera o objeto que foi carregado com `data(murders)`. Se carregarmos os dados `murders` novamente, o original substituirá nossa versão mutada.

4.3.2 Como criar subconjuntos com `filter`

Agora, suponha que desejamos filtrar a tabela de dados para mostrar apenas as entradas para as quais a taxa de homicídios é menor que 0,71. Para fazer isso, usamos a função `filter`, que usa a tabela de dados como o primeiro argumento e, em seguida, a instrução condicional como o segundo. O mesmo que com `mutate`, podemos usar nomes de variáveis sem aspas de `murders` dentro da função e ele saberá que estamos nos referindo às colunas e não aos objetos na área de trabalho.

```
filter(murders, rate <= 0.71)
#>      state abb      region population total    rate
#> 1    Hawaii  HI        West   1360301     7 0.515
#> 2    Iowa   IA North Central  3046355    21 0.689
#> 3 New Hampshire NH Northeast 1316470      5 0.380
#> 4 North Dakota ND North Central  672591     4 0.595
#> 5 Vermont  VT Northeast  625741      2 0.320
```

4.3.3 Como selecionar colunas com `select`

Embora nossa tabela de dados tenha apenas seis colunas, algumas tabelas de dados incluem centenas. Se queremos ver apenas algumas colunas, podemos usar a função `select` de `dplyr`. No código a seguir, selecionamos três, atribuímos o resultado a um novo objeto e filtramos esse novo objeto:

```
new_table <- select(murders, state, region, rate)
filter(new_table, rate <= 0.71)
#>      state      region    rate
#> 1    Hawaii       West 0.515
#> 2    Iowa  North Central 0.689
#> 3 New Hampshire  Northeast 0.380
#> 4 North Dakota North Central 0.595
#> 5 Vermont       Northeast 0.320
```

Na chamada para `select`, o primeiro argumento `murders` é um objeto mas `state`, `region` e `rate` eles são nomes de variáveis.

4.4 Exercícios

- Carregue o pacote `dplyr` e o conjunto de dados de assassinato nos EUA.

```
library(dplyr)
library(dslabs)
data(murders)
```

Você pode adicionar colunas usando a função `mutate` de `dplyr`. Esta função reconhece os nomes das colunas e, dentro da função, você pode chamá-los sem aspas:

```
murders <- mutate(murders, population_in_millions = population/ 10^6)
```

Nós podemos escrever `population` em vez de `murders$population`. A função `mutate` sabe-mos que estamos agarrando colunas de `murders`.

Usar função `mutate` para adicionar uma coluna de assassinato chamada `rate` com a taxa de homicídios por 100.000, como no código do exemplo acima. Certifique-se de redefinir `murders` como foi feito no código do exemplo anterior (`assassinatos <- [seu código]`) para que possamos continuar usando essa variável.

2. Sim `rank(x)` fornece o alcance das entradas `x` do menor para o maior, `rank(-x)` fornece os intervalos do maior para o menor. Usar função `mutate` adicionar uma coluna `rank` a contendo a faixa de taxa de homicídios da maior para a menor. Certifique-se de redefinir `murders` para continuar usando esta variável.

3. Com `dplyr`, podemos usar `select` para mostrar apenas determinadas colunas. Por exemplo, com este código, mostrariamos apenas os estados e tamanhos da população:

```
select(murders, state, population) %>% head()
```

Usar `select` para exibir nomes e abreviações de estado em `murders`. Não redefina `murders`, apenas mostre os resultados.

4. A função `filter` __dplyr__ é usado para escolher linhas específicas do _data frame_ para salvar. A diferença de `select` que é para colunas, `filter` é para linhas. Por exemplo, você pode exibir apenas a linha de Nova York da seguinte maneira:

```
filter(murders, state == "New York")
```

Você pode usar outros vetores lógicos para filtrar linhas.

Usar `filter` para mostrar os cinco estados com as maiores taxas de homicídio. Depois de adicionar a taxa e o intervalo de assassinatos, não altere o conjunto de dados de assassinatos nos EUA. EUA, apenas mostre o resultado. Lembre-se de que você pode filtrar com base na coluna `rank`.

5. Podemos excluir linhas usando o operador `!=`. Por exemplo, para remover a Flórida, faríamos o seguinte:

```
no_florida <- filter(murders, state != "Florida")
```

Crie um novo *data frame* com o nome `no_south` isso elimina os estados da região sul. Quantos estados há nesta categoria? Você pode usar a função `nrow` para isto.

6. Também podemos usar `%in%` para filtrar com `dplyr`. Portanto, você pode visualizar os dados de Nova York e Texas como este:

```
filter(murders, state %in% c("New York", "Texas"))
```

Crie um novo *data frame* chamado `murders_nw` apenas com os estados nordeste e oeste. Quantos estados há nesta categoria?

7. Suponha que você queira morar no nordeste ou oeste e que a taxa de homicídios seja menor que 1. Queremos ver os dados dos estados que satisfazem essas opções. Observe que você pode usar operadores lógicos com `filter`. Aqui está um exemplo em que filtramos para manter apenas pequenos estados na região nordeste.

```
filter(murders, population < 5000000 & region == "Northeast")
```

Certifique-se de que `murders` foi definido com `rate` e `rank` e ainda tem todos os estados. Crie uma tabela chamada `my_states` contendo linhas para estados que satisfazem ambas as condições: é no nordeste ou oeste e a taxa de homicídios é menor que 1. `select` para exibir apenas o nome, a taxa e o intervalo do estado.

4.5 O cano_: %>%

Com **dplyr**, podemos executar uma série de operações, por exemplo **select** e então **filter**, enviando os resultados de uma função para outra usando o que é chamado de *pipe operator*: **%>%**. Alguns detalhes estão incluídos abaixo.

Escrevemos o código acima para mostrar três variáveis (estado, região, taxa) para estados com taxas de homicídio abaixo de 0,71. Para isso, definimos o objeto intermediário **new_table**. Em **dplyr**, podemos escrever um código mais parecido com uma descrição do que queremos fazer sem objetos intermediários:

original data → select → filter

Para essa operação, podemos usar o *pipe* **%>%**. O código fica assim:

```
murders %>% select(state, region, rate) %>% filter(rate <= 0.71)
#>           state      region    rate
#> 1        Hawaii       West  0.515
#> 2        Iowa North Central 0.689
#> 3 New Hampshire     Northeast 0.380
#> 4 North Dakota North Central 0.595
#> 5    Vermont     Northeast 0.320
```

Essa linha de código é equivalente às duas linhas de código anteriores. O que está acontecendo aqui?

Em geral, o *pipe* envia o resultado que está no lado esquerdo do *pipe* para ser o primeiro argumento da função no lado direito do *pipe*. Aqui está um exemplo simples:

```
16 %>% sqrt()
#> [1] 4
```

Podemos continuar canalizando valores (*piping* em inglês) ao longo de:

```
16 %>% sqrt() %>% log2()
#> [1] 2
```

A declaração acima é equivalente a `log2(sqrt(16))`.

Lembre-se de que o *pipe* envia valores para o primeiro argumento, para que possamos definir outros argumentos como se o primeiro argumento já estivesse definido:

```
16 %>% sqrt() %>% log(base = 2)
#> [1] 2
```

Portanto, ao usar `_pipe_com_data frames_edplyr`, não precisamos mais especificar o primeiro argumento necessário, pois as funções **dplyr** que descrevemos pegam todos os dados como o primeiro argumento. No código que escrevemos:

```
murders %>% select(state, region, rate) %>% filter(rate <= 0.71)
```

`murders` é o primeiro argumento da função **select** e o novo *data frame* (anteriormente **new_table**) é o primeiro argumento da função **filter**.

Observe que o *pipe* funciona bem com funções em que o primeiro argumento são os dados de

entrada. As funções nos pacotes `tidyverse` e `dplyr` têm esse formato e podem ser facilmente usadas com o `pipe`.

4.6 Exercícios

- O cano `%>%` ele pode ser usado para executar operações sequencialmente sem precisar definir objetos intermediários. Comece redefinindo `murders` para incluir a taxa e o intervalo.

```
murders <- mutate(murders, rate = total/ population * 100000,  
rank = rank(-rate))
```

Na solução do exercício anterior, fizemos o seguinte:

```
my_states <- filter(murders, region %in% c("Northeast", "West") &  
rate < 1)  
  
select(my_states, state, rate, rank)
```

O cano `%>%` nos permite executar as duas operações sequencialmente sem precisar definir uma variável intermediária `my_states`. Então, poderíamos ter mudado e selecionado na mesma linha assim:

```
mutate(murders, rate = total/ population * 100000,  
rank = rank(-rate)) %>%  
select(state, rate, rank)
```

Eu sinto isso `select` ele não possui mais um *data frame* como seu primeiro argumento. O primeiro argumento é assumido como o resultado da operação realizada imediatamente antes `%>%`.

Repita o exercício anterior, mas agora, em vez de criar um novo objeto, mostre o resultado e inclua apenas as colunas de status, velocidade e intervalo. Use um *pipe* `%>%` para fazer isso em uma linha.

- Reinic平 murders para a tabela original usando `data(murders)`. Use um *pipe* para criar um novo *data frame* chamado `my_states` ele considera apenas os estados do nordeste ou oeste que têm uma taxa de homicídios menor que 1 e contém apenas as colunas de estado, taxa e faixa. O *pipe* também deve ter quatro componentes separados por três `%>%`. O código deve se parecer com o seguinte:

```
my_states <- murders %>%  
mutate SOMETHING %>%  
filter SOMETHING %>%  
select SOMETHING
```

4.7 Como resumir dados

Uma parte importante da análise exploratória de dados é resumir os dados. A média e o desvio padrão são dois exemplos de estatísticas resumidas amplamente usadas. É pos-

sível obter resumos mais informativos, dividindo primeiro os dados em grupos. Nesta seção, abordamos dois novos verbos **dplyr** que facilitam esses cálculos: **summarize** e **group_by**. Aprendemos a acessar os valores resultantes usando a função **pull**.

4.7.1 summarize

A função **summarize** o de **dplyr** oferece uma maneira de calcular estatísticas resumidas com código intuitivo e legível. Começamos com um exemplo simples baseado em alturas. O conjunto de dados **heights** inclui as alturas e o sexo relatados pelos alunos em uma pesquisa de classe.

```
library(dplyr)
library(dslabs)
data(heights)
```

O código a seguir calcula a média e o desvio padrão para mulheres:

```
s <- heights %>%
  filter(sex == "Female") %>%
  summarize(average = mean(height), standard_deviation = sd(height))
s
#>   average standard deviation
#> 1      64.9             3.76
```

Isso leva nossa tabela de dados original como entrada, a filtra para incluir apenas as linhas que representam as fêmeas e, em seguida, produz uma nova tabela de resumo com apenas a média e o desvio padrão das alturas. Podemos escolher os nomes das colunas da tabela resultante. Por exemplo, acima, decidimos usar **average** e **standard_deviation**, mas poderíamos ter usado outros nomes da mesma maneira.

Como a tabela resultante armazenada em **s** é um *data frame*, podemos acessar os componentes com o operador de acesso **\$**:

```
s$average
#> [1] 64.9
s$standard_deviation
#> [1] 3.76
```

Como na maioria das outras funções **dplyr**, **summarize** conhece os nomes das variáveis e podemos usá-las diretamente. Então, quando escrevemos **mean(height)** dentro da chamada de função **summarize**, a função acessa a coluna denominada “height” ou **height** e calcula a média do vetor numérico resultante. Podemos calcular qualquer outro resumo que opere em vetores e retorne um único valor. Por exemplo, podemos adicionar as medianas, alturas mínima e máxima desta maneira:

```
heights %>%
  filter(sex == "Female") %>%
  summarize(median = median(height), minimum = min(height),
            maximum = max(height))
#>   median minimum maximum
#> 1      65       51       79
```

Podemos obter esses três valores com apenas uma linha usando a função **quantile**: por exemplo, **quantile(x, c(0,0.5,1))** retorna o mínimo (percentil 0), mediana (percentil

50) e máximo (percentil 100) do vetor x. No entanto, se tentarmos usar uma função como essa que retorne dois ou mais valores dentro `summarize`:

```
heights %>%
filter(sex == "Female") %>%
summarize(range = quantile(height, c(0, 0.5, 1)))
```

receberemos um erro: `Error: expecting result of length one, got : 2.` Com função `summarize`, podemos chamar apenas funções que retornam um único valor. Na seção 4.12 vamos aprender a lidar com funções que retornam mais de um valor.

Para outro exemplo de como podemos usar a função `summarize` vamos calcular a taxa média de homicídios nos Estados Unidos. Lembre-se de que nossa tabela de dados inclui o total de assassinatos e o tamanho da população de cada estado e já usamos `dplyr` para adicionar uma coluna de taxa de assassinatos:

```
murders <- murders %>% mutate(rate = total/population*100000)
```

Lembre-se que a taxa de homicídios nos EUA EUA **não** é a média das taxas de homicídio do estado:

```
summarize(murders, mean(rate))
#>   mean(rate)
#> 1      2.78
```

Isso ocorre porque no cálculo anterior, estados pequenos têm o mesmo peso que estados grandes. A taxa de assassinatos nos Estados Unidos é o número total de assassinatos nos Estados Unidos dividido pela população total. Portanto, o cálculo correto é:

```
us_murder_rate <- murders %>%
summarize(rate = sum(total)/ sum(population) * 100000)
us_murder_rate
#>   rate
#> 1 3.03
```

Este cálculo conta estados maiores proporcionalmente ao seu tamanho, resultando em um valor maior.

4.7.2 `pull`

O objeto `us_murder_rate` definido acima representa apenas um número. No entanto, estamos armazenando-o em um *data frame*:

```
class(us_murder_rate)
#> [1] "data.frame"
```

desde que, como a maioria das funções `dplyr`, `summarize` sempre retorna um *data frame*.

Isso pode ser problemático se quisermos usar esse resultado com funções que requerem um valor numérico. Aqui está um truque útil para acessar os valores armazenados nos dados quando usamos *pipes*: quando um objeto de dados é canalizado (`_` é canalizado em inglês), esse objeto e suas colunas podem ser acessados usando a função `pull`. Para entender o que queremos dizer, considere esta linha de código:

```
us_murder_rate %>% pull(rate)
#> [1] 3.03
```

Isso retorna o valor na coluna `rate` do `us_murder_rate` tornando-o equivalente a `us_murder_rate$rate`.

Para obter um número da tabela de dados original com uma linha de código, podemos escrever:

```
us_murder_rate <- murders %>%
  summarize(rate = sum(total) / sum(population) * 100000) %>%
  pull(rate)

us_murder_rate
#> [1] 3.03
```

que agora é numérico:

```
class(us_murder_rate)
#> [1] "numeric"
```

4.7.3 Como agrupar e resumir com `group_by`

Uma operação comum na exploração de dados é primeiro dividir os dados em grupos e depois calcular resumos para cada grupo. Por exemplo, podemos querer calcular a média e o desvio padrão para as alturas de homens e mulheres separadamente. A função `group_by` nos ajuda a fazer isso.

Se escrevermos isso:

```
heights %>% group_by(sex)
#> # A tibble: 1,050 x 2
#> # Groups:   sex [2]
#>   sex     height
#>   <fct>  <dbl>
#> 1 Male      75
#> 2 Male      70
#> 3 Male      68
#> 4 Male      74
#> 5 Male      61
#> # ... with 1,045 more rows
```

O resultado não parece muito diferente de `heights` exceto que vemos `Groups: sex [2]` quando imprimimos o objeto. Embora não seja imediatamente óbvio a partir de sua aparência, agora é um *data frame_especial* chamado *data de dados agrupados*, e as funções de `dplyr`, em particular `summarize`, eles se comportarão de maneira diferente quando agirem sobre esse objeto. Conceitualmente, eles podem pensar nessa tabela como muitas tabelas, com as mesmas colunas, mas não necessariamente com o mesmo número de linhas, empilhadas em um objeto. Quando resumimos os dados após o agrupamento, é o que acontece:

```
heights %>%
  group_by(sex) %>%
  summarize(average = mean(height), standard_deviation = sd(height))
#> `summarise()` ungrouping output (override with `groups` argument)
#> # A tibble: 2 x 3
#>   sex     average standard deviation
#>   <fct>    <dbl>           <dbl>
```

```
#> 1 Female      64.9          3.76
#> 2 Male        69.3          3.61
```

A função `summarize` aplique o resumo a cada grupo separadamente.

Para ver outro exemplo, vamos calcular a taxa média de homicídios nas quatro regiões do país:

```
murders %>%
  group_by(region) %>%
  summarize(median_rate = median(rate))
#> `summarise()` ungrouping output (override with `.`groups` argument)
#> # A tibble: 4 x 2
#>   region      median_rate
#>   <fct>        <dbl>
#> 1 Northeast    1.80
#> 2 South        3.40
#> 3 North Central 1.97
#> 4 West         1.29
```

4.8 Como encomendar os *data frames*

Ao examinar um conjunto de dados, geralmente é conveniente classificar numericamente ou alfabeticamente, com base em uma ou mais das colunas da tabela. Conhecemos as funções `order` e `sort`, mas para classificar tabelas inteiras, a função `arrange dplyr` é útil. Por exemplo, aqui ordenamos os estados de acordo com o tamanho da população:

```
murders %>%
  arrange(population) %>%
  head()
#> #>       state abb      region population total    rate
#> 1 Wyoming  WY      West     563626    5 0.887
#> 2 District of Columbia DC      South    601723   99 16.453
#> 3 Vermont   VT      Northeast 625741    2 0.320
#> 4 North Dakota ND      North Central 672591    4 0.595
#> 5 Alaska    AK      West     710231   19 2.675
#> 6 South Dakota SD      North Central 814180    8 0.983
```

Com `arrange` podemos decidir qual coluna usar para solicitar. Para ver os estados por população, do menor para o maior, organizamos pela `rate`:

```
murders %>%
  arrange(rate) %>%
  head()
#> #>       state abb      region population total    rate
#> 1 Vermont   VT      Northeast 625741    2 0.320
#> 2 New Hampshire NH      Northeast 1316470   5 0.380
#> 3 Hawaii    HI      West     1360301   7 0.515
#> 4 North Dakota ND      North Central 672591    4 0.595
```

```
#> 5      Iowa IA North Central    3046355   21 0.689
#> 6     Idaho ID        West    1567582   12 0.766
```

Observe que o comportamento padrão é classificar em ordem crescente. Em **dplyr**, a função **desc** transformar um vetor para que fique em ordem decrescente. Para classificar a tabela em ordem decrescente, podemos escrever:

```
murders %>%
  arrange(desc(rate))
```

4.8.1 Como encomendar aninhado

Se estivermos ordenando uma coluna quando houver empates, podemos usar uma segunda coluna para quebrar o empate. Da mesma forma, uma terceira coluna pode ser usada para romper os laços entre a primeira e a segunda, e assim por diante. Aqui nós ordenamos por **region**, na região, ordenamos por taxa de homicídio:

```
murders %>%
  arrange(region, rate) %>%
  head()
#>           state abb   region population total  rate
#> 1      Vermont VT Northeast    625741     2 0.320
#> 2 New Hampshire NH Northeast  1316470     5 0.380
#> 3       Maine ME Northeast  1328361    11 0.828
#> 4 Rhode Island RI Northeast 1052567    16 1.520
#> 5 Massachusetts MA Northeast 6547629   118 1.802
#> 6      New York NY Northeast 19378102   517 2.668
```

4.8.2 Os primeiros n

No código acima, usamos a função **head** para impedir que a página seja preenchida com todo o conjunto de dados. Se queremos ver uma proporção maior, podemos usar a função **top_n**. Essa função usa um *data frame* como o primeiro argumento, o número de linhas a serem exibidas no segundo e a variável a ser filtrada no terceiro. Aqui está um exemplo de como visualizar as 5 principais linhas:

```
murders %>% top_n(5, rate)
#>           state abb   region population total  rate
#> 1 District of Columbia DC      South    601723    99 16.45
#> 2       Louisiana LA      South   4533372   351  7.74
#> 3       Maryland MD      South   5773552   293  5.07
#> 4      Missouri MO North Central  5988927   321  5.36
#> 5 South Carolina SC      South   4625364   207  4.48
```

Observe que as linhas não são ordenadas por **rate**, apenas filtrado. Se quisermos pedir, precisamos usar **arrange**. Lembre-se de que, se o terceiro argumento for deixado em branco, **top_n** filtrar pela última coluna.

4.9 Exercícios

Para esses exercícios, usaremos os dados da pesquisa coletados pelo Centro Nacional de Estatísticas da Saúde dos Estados Unidos (NCHS). Este centro realiza uma série de pesquisas em saúde e nutrição desde a década de 1960. Desde 1999, cerca de 5.000 indivíduos de todas as idades foram entrevistadas a cada ano e concluíram o componente de triagem de saúde da pesquisa. Alguns dos dados estão disponíveis no pacote **NHANES**. Depois de instalar o pacote **NHANES**, eles podem carregar os dados da seguinte maneira:

```
library(NHANES)
data(NHANES)
```

Os dados **NHANES** têm muitos valores ausentes. As funções `mean` e `sd` retornará `NA` se alguma das entradas do vetor de entrada for uma `NA`. Aqui está um exemplo:

```
library(dslabs)
data(na_example)
mean(na_example)
#> [1] NA
sd(na_example)
#> [1] NA
```

Para ignorar o `NA`s podemos usar o argumento `na.rm`:

```
mean(na_example, na.rm = TRUE)
#> [1] 2.3
sd(na_example, na.rm = TRUE)
#> [1] 1.22
```

Vamos agora explorar os dados **NHANES**.

1. Oferecemos algumas informações básicas sobre pressão arterial. Primeiro, vamos selecionar um grupo para definir o padrão. Usaremos mulheres de 20 a 29 anos. `AgeDecade` é uma variável categórica com essas idades. Observe que a categoria está codificada “20-29”, com um espaço na frente! Qual é a média e desvio padrão da pressão arterial sistólica, conforme armazenado na variável `BPSysAve`? Salve-o em uma variável chamada `ref`.

Dica: use `filter` e `summarize` e use o argumento `na.rm = TRUE` ao calcular a média e o desvio padrão. Você também pode filtrar valores de `NA` usando `filter`.

2. Usando um `pipe`, atribua a média a uma variável numérica `ref_avg`. Dica: use o código semelhante ao acima e depois `pull`.

3. Agora insira os valores mínimo e máximo para o mesmo grupo.

4. Calcule a média e o desvio padrão para as mulheres, mas para cada faixa etária separadamente, em vez de uma década selecionada, como na pergunta 1. Observe que as faixas etárias são definidas por `AgeDecade`. Dica: em vez de filtrar por idade e sexo, filtре por `Gender` e depois use `group_by`.

5. Repita o exercício 4 para os meninos.

6. Podemos combinar os dois resumos dos exercícios 4 e 5 em uma linha de código. Isto é porque `group_by` permite agrupar por mais de uma variável. Obtenha uma excelente tabela de resumo usando `group_by(AgeDecade, Gender)`.

7. Para homens entre 40 e 49 anos, compare a pressão arterial sistólica por raça, conforme aparece na variável `Race1`. Encomende a tabela resultante com base na pressão arterial sistólica média mais baixa para a mais alta.

4.10 Tibbles

Os dados `tidy` devem ser armazenados em `data frames`. Discutimos o `data frame` na Seção 2.4.1 e estamos usando o `data frame` `murders` ao longo do livro. Na seção 4.7.3 nós introduzimos a função `group_by`, que permite estratificar os dados antes de calcular as estatísticas de resumo. Mas onde estão as informações do grupo armazenadas no `data frame`?

```
murders %>% group_by(region)
#> # A tibble: 51 x 6
#> # Groups:   region [4]
#>   state     abb   region population total    rate
#>   <chr>     <chr> <fct>        <dbl> <dbl>   <dbl>
#> 1 Alabama   AL    South       4779736   135   2.82
#> 2 Alaska    AK    West        710231    19   2.68
#> 3 Arizona   AZ    West        6392017   232   3.63
#> 4 Arkansas  AR    South      2915918    93   3.19
#> 5 California CA    West      37253956  1257   3.37
#> # ... with 46 more rows
```

Observe que não há colunas com essas informações. Mas se você olhar para a saída acima, verá a linha `A tibble` seguido por dimensões. Podemos aprender a classe do objeto retornado usando:

```
murders %>% group_by(region) %>% class()
#> [1] "grouped_df" "tbl_df"      "tbl"        "data.frame"
```

O `tbl` é um tipo especial de `data frame`. As funções `group_by` e `summarize` sempre retornam esse tipo de `data frame`. A função `group_by` retorna um tipo especial de `tbl`, a `grouped_df`. Discutiremos isso mais tarde. Para consistência, os verbos de manipulação `dplyr` (`select`, `filter`, `mutate` e `arrange`) preserva a classe `input`: se eles recebem um `data frame` regular, eles retornam um `data frame` regular, enquanto que se eles recebem um `tibble`, eles retornam um `tibble`. Mas `tibbles` é o formato `preferido_tidyverse` e, como resultado, funções `tidyverse` que produzem um `data frame` do zero retornam uma `tibble`. Por exemplo, no capítulo 5 veremos que as funções `tidyverse_usadas` para importar dados criam `tibbles`.

`Tibbles` são muito semelhantes aos `data frames`. De fato, eles podem pensar neles como uma versão moderna de `data frames`. No entanto, existem três diferenças importantes que descreveremos abaixo.

4.10.1 Tibbles parece melhor

O método de impressão para `tibbles` é mais legível que o de `um_data frame`. Para ver isso, compare a saída da digitação `murders` e a saída dos assassinatos, se os tornarmos uma tagarelice. Podemos fazer isso usando `as_tibble(murders)`. Se você usa o RStudio, a saída_de_tabela_ ajusta-se ao tamanho da sua janela. Para ver isso, altere a largura do seu console R e observe como mais/ menos colunas são exibidas.

4.10.2 Tibbles_subconjuntos_são_tibbles

Se criarmos subconjuntos das colunas de um *data frame*, eles poderão retornar um objeto que não seja *data frame*, como um vetor ou escalar. Por exemplo:

```
class(murders[,4])
#> [1] "numeric"
```

não é um *data frame*. Com *tibbles* isso não acontece:

```
class(as_tibble(murders)[,4])
#> [1] "tbl_df"     "tbl"        "data.frame"
```

Isso é útil em *tidyverse*, pois as funções requerem *data frames_como_input*.

Com *tibbles*, se você deseja acessar o vetor que define uma coluna e não recuperar um *data frame*, deve usar o operador de acesso \$:

```
class(as_tibble(murders)$population)
#> [1] "numeric"
```

Um recurso relacionado é que *tibbles* avisará se eles tentarem acessar uma coluna que não existe. Por exemplo, se escrevermos acidentalmente *Population* ao invés de *population* nós vemos que:

```
murders$Population
#> NULL
```

retorna um NULL sem aviso, o que pode dificultar a depuração. Por outro lado, se tentarmos isso com uma *tibble*, obteremos um aviso informativo:

```
as_tibble(murders)$Population
#> Warning: Unknown or uninitialised column: `Population`.
#> NULL
```

4.10.3 Tibbles_pode_ter_entradas_complexas

Embora as colunas do *data frame* devam ser vetores de números, cadeias ou valores lógicos, *tibbles* pode ter objetos mais complexos, como listas ou funções. Além disso, podemos criar *tibbles* com funções:

```
tibble(id = c(1, 2, 3), func = c(mean, median, sd))
#> # A tibble: 3 x 2
#>       id   func
#>   <dbl> <list>
#> 1     1 <fn>
#> 2     2 <fn>
#> 3     3 <fn>
```

4.10.4 Tibbles_podem_ser_agrupados

A função *group_by* retorna um tipo especial de *tibble*: um *tibble* agrupado. Essa classe armazena informações que permitem saber quais linhas estão em quais grupos. Funções *Tidyverse*, em particular a função *summarize*, estão cientes das informações do grupo.

4.10.5 Como criar um *tibble* usando *tibble* ao invés de *data.frame*

Às vezes, é útil criarmos nossos próprios *data frames*. Para criar um *data frame_no formato_tibble*, você pode usar a função *tibble*.

```
grades <- tibble(names = c("John", "Juan", "Jean", "Yao"),
exam_1 = c(95, 80, 90, 85),
exam_2 = c(90, 85, 85, 90))
```

Observe que a base R (nenhum pacote carregado) tem uma função com um nome muito semelhante, *data.frame*, que pode ser usado para criar um *data frame_regular* em vez de um *tibble*. Outra diferença importante é que, por padrão *data.frame* forçar a conversão de caracteres em fatores sem fornecer um aviso ou mensagem:

```
grades <- data.frame(names = c("John", "Juan", "Jean", "Yao"),
exam_1 = c(95, 80, 90, 85),
exam_2 = c(90, 85, 85, 90))
class(grades$names)
#> [1] "character"
```

Para evitar isso, usamos o argumento bastante complicado *stringsAsFactors*:

```
grades <- data.frame(names = c("John", "Juan", "Jean", "Yao"),
exam_1 = c(95, 80, 90, 85),
exam_2 = c(90, 85, 85, 90),
stringsAsFactors = FALSE)
class(grades$names)
#> [1] "character"
```

Para converter um *data frame_normal em um_tibble*, você pode usar a função *as_tibble*.

```
as_tibble(grades) %>% class()
#> [1] "tbl_df"     "tbl"        "data.frame"
```

4.11 O operador de ponto

Uma das vantagens de usar o *pipe %>%* é que não precisamos continuar nomeando novos objetos enquanto manipulamos o *data frame*. Lembre-se de que, se quisermos calcular a taxa média de homicídios nos estados do sul, em vez de escrever:

```
tab_1 <- filter(murders, region == "South")
tab_2 <- mutate(tab_1, rate = total/ population * 10^5)
rates <- tab_2$rate
median(rates)
#> [1] 3.4
```

podemos evitar definir novos objetos intermediários escrevendo:

```
filter(murders, region == "South") %>%
mutate(rate = total/ population * 10^5) %>%
summarize(median = median(rate)) %>%
```

```
pull(median)
#> [1] 3.4
```

Podemos fazer isso porque cada uma dessas funções usa um *data frame* como o primeiro argumento. Mas e se quisermos acessar um componente do *data frame*? Por exemplo, e se a função `pull` não está disponível e queremos acessar `tab_2$rate`? Que nome de quadro de dados usamos? A resposta é o operador de ponto (*dot operator* em inglês).

Por exemplo, para acessar o vetor de velocidade sem a função `pull`, poderíamos usar:

```
rates <- filter(murders, region == "South") %>%
  mutate(rate = total/ population * 10^5) %>%
  .$rate
median(rates)
#> [1] 3.4
```

Na próxima seção, veremos outras instâncias nas quais usar o `.` é útil.

4.12 do

As funções *tidyverse_sabem como interpretar_tibbles agrupados*. Além disso, para facilitar o script através do *pipe* `%>%` as funções *tidyverse_retornam constantemente_data frames*, pois isso garante que a saída de uma função seja aceita como a entrada de outra. Mas a maioria das funções R não reconhece *tibbles_agrupados nem retorna_data frames*. A função `quantile` é um exemplo que descrevemos na seção 4.7.1. A função `do` serve como uma ponte entre as funções de R, como `quantile` e o *tidyverse*. A função `do` entende *tibbles_agrupados e sempre retorna um_data frame*.

Na seção 4.7.1, percebemos que, se tentarmos usar `quantile` para obter o mínimo, a mediana e o máximo de uma chamada, receberemos um erro: `Error: expecting result of length one, got : 2.`

```
data(heights)
heights %>%
  filter(sex == "Female") %>%
  summarize(range = quantile(height, c(0, 0.5, 1)))
```

Nós podemos usar a função `do` para corrigir isso.

Primeiro, precisamos escrever uma função que se ajuste ao foco do *tidyverse*: isto é, ele recebe um *data frame_e retorna um_data frame*.

```
my_summary <- function(dat){
  x <- quantile(dat$height, c(0, 0.5, 1))
  tibble(min = x[1], median = x[2], max = x[3])
}
```

Agora podemos aplicar a função ao conjunto de dados de altura para obter os resumos:

```
heights %>%
  group_by(sex) %>%
  my_summary
```

```
#> # A tibble: 1 x 3
#>   min median max
#>   <dbl> <dbl> <dbl>
#> 1     50    68.5  82.7
```

Mas não é isso que queremos. Queremos um resumo para cada gênero e o código retornou apenas um resumo. Isto é porque `my_summary` não faz parte do `tidyverse` e não sabe como lidar com os `tibbles` agrupados. `do` faz esta conexão:

```
heights %>%
group_by(sex) %>%
do(my_summary())
#> # A tibble: 2 x 4
#> Groups:   sex [2]
#>   sex      min median max
#>   <fct> <dbl> <dbl> <dbl>
#> 1 Female    51    65.0  79
#> 2 Male      50    69    82.7
```

Lembre-se de que aqui precisamos usar o operador de ponto. O `tibble` criado por `group_by` é canalizado para `do`. Dentro da chamada para `do`, o nome dessa `tibble` é `.` e queremos enviá-lo para `my_summary`. Se eles não usarem o ponto, então `my_summary` ele não tem argumento e retorna um erro nos dizendo que o argumento "dat". Você pode ver o erro digitando:

```
heights %>%
group_by(sex) %>%
do(my_summary())
```

Se eles não usarem parênteses, a função não será executada e, em vez disso, `do` tenta retornar a função. Isso dá um erro porque `do` você sempre deve retornar um *data frame*. Você pode ver o erro digitando:

```
heights %>%
group_by(sex) %>%
do(my_summary)
```

4.13 O pacote purrr

Na seção 3.5 nós aprendemos sobre função `sapply`, o que nos permitiu aplicar a mesma função a cada elemento de um vetor. Criamos uma função e usamos `sapply` para calcular a soma do primeiro `n` números inteiros para vários valores de `n` assim:

```
compute_s_n <- function(n){
  x <- 1:n
  sum(x)
}
n <- 1:25
s_n <- sapply(n, compute_s_n)
```

Esse tipo de operação, que aplica a mesma função ou procedimento aos elementos de um

objeto, é bastante comum na análise de dados. O pacote **purrr** inclui funções semelhantes a **sapply** mas elas interagem melhor com outras funções *tidyverse*. A principal vantagem é que podemos controlar melhor o tipo de resultado das funções. Por contraste, **sapply** você pode retornar vários tipos diferentes de objetos, convertendo-os quando conveniente. As funções **purrr** nunca farão isso: elas retornarão objetos de um tipo específico ou retornarão um erro se isso não for possível.

A primeira função de **purrr** que aprenderemos é **map**, que funciona muito semelhante a **sapply** mas sempre, sem exceção, ele retorna uma lista:

```
library(purrr)
s_n <- map(n, compute_s_n)
class(s_n)
#> [1] "list"
```

Se queremos um vetor numérico, podemos usar **map_dbl** que sempre retorna um vetor de valores numéricos.

```
s_n <- map_dbl(n, compute_s_n)
class(s_n)
#> [1] "numeric"
```

Isso produz os mesmos resultados que a chamada **sapply** que vemos acima.

Uma função *purrr* particularmente útil para interagir com o resto do *tidyverse* é **map_df**, que sempre retorna um *tibble data frame*. No entanto, a função chamada deve retornar um vetor ou uma lista com nomes. Por esse motivo, o código a seguir resultaria em um erro `Argument 1 must have names:`:

```
s_n <- map_df(n, compute_s_n)
```

Precisamos alterar a função para corrigir isso:

```
compute_s_n <- function(n){
  x <- 1:n
  tibble(sum = sum(x))
}
s_n <- map_df(n, compute_s_n)
```

O pacote **purrr** oferece muito mais funcionalidades não discutidas aqui. Para mais detalhes, você pode consultar [este recurso online] (<https://jennybc.github.io/purrr-tutorial/>).

4.14 Os condicionais *tidyverse*

Uma análise de dados típica geralmente envolve uma ou mais operações condicionais. Na seção 3.1 nós descrevemos a função **ifelse**, que usaremos extensivamente neste livro. Nesta seção, apresentamos duas funções **dplyr** que oferecem funcionalidade adicional para executar operações condicionais.

4.14.1 case_when

A função `case_when` é útil para vetorizar instruções condicionais. Isso é semelhante a `ifelse` mas pode gerar qualquer número de valores, em vez de apenas TRUE ou FALSE. Aqui está um exemplo que divide os números em negativo, positivo e 0:

```
x <- c(-2, -1, 0, 1, 2)
case_when(x < 0 ~ "Negative", x > 0 ~ "Positive", TRUE ~ "Zero")
#> [1] "Negative" "Negative" "Zero"      "Positive" "Positive"
```

Um uso comum dessa função é definir variáveis categóricas com base nas variáveis existentes. Por exemplo, suponha que desejamos comparar as taxas de homicídios em quatro grupos de estados: Nova Inglaterra, Costa Oeste, Sul e Outro. Para cada estado, primeiro perguntamos se está na Nova Inglaterra. Se a resposta for não, perguntamos se está na Costa Oeste e, se não, perguntamos se está no Sul e, se não, atribuímos uma das opções acima (*Outro*). Aqui vemos como usamos `case_when` para fazer isso:

```
murders %>%
  mutate(group = case_when(
    abb %in% c("ME", "NH", "VT", "MA", "RI", "CT") ~ "New England",
    abb %in% c("WA", "OR", "CA") ~ "West Coast",
    region == "South" ~ "South",
    TRUE ~ "Other"))
  ) %>%
  group_by(group) %>%
  summarize(rate = sum(total)/ sum(population) * 10^5)
#> `summarise()` ungrouping output (override with `groups` argument)
#> # A tibble: 4 x 2
#>   group        rate
#>   <chr>     <dbl>
#> 1 New England  1.72
#> 2 Other        2.71
#> 3 South        3.63
#> 4 West Coast   2.90
```

4.14.2 between

Uma operação comum na análise de dados é determinar se um valor cai dentro de um intervalo. Podemos verificar isso usando condicionais. Por exemplo, para verificar se os elementos de um vetor `x` estão entre `a` e `b` nós podemos escrever:

```
x >= a & x <= b
```

No entanto, isso pode ficar complicado, especialmente dentro da abordagem *tidyverse*. A função `between` executa a mesma operação:

```
between(x, a, b)
```

4.15 Exercícios

- Carregar o conjunto de dados `murders`. Qual dos seguintes é verdadeiro?

para. `murders` está no formato *tidy* e é armazenado em *uma tibble*. b. `murders` está no formato *tidy* e é armazenado em *um data frame*. c. `murders` não está no formato *tidy* e é armazenado em *um tibble*. d. `murders` não está no formato *tidy* e é armazenado em *um data frame*.

2. Usar `as_tibble` converter a tabela de dados `murders` em uma *tibble* e salve-a em um objeto chamado `murders_tibble`.

3. Use a função `group_by` converter `murders` em uma *tibble* que é agrupada por região.

4. Escreva o código *tidyverse* que é equivalente a este código:

```
exp(mean(log(murders$population)))
```

Escreva usando o *pipe* para que cada função seja chamada sem argumentos. Use o operador de ponto para acessar a população. Dica: o código deve começar com `murders %>%`.

5. Use o `map_df` para criar um *data frame* com três colunas denominadas `n`, `s_n` e `s_n_2`. A primeira coluna deve conter os números de 1 a 100. A segunda e a terceira coluna devem conter a soma de 1 a 100. `n` com `n` representando o número da linha.

5

Importando dados

Até agora, temos usado conjuntos de dados já disponibilizados como objetos do R. Entretanto, cientistas de dados raramente têm a mesma sorte e, geralmente, precisam importar dados no R de arquivos, bancos de dados ou outras fontes. Atualmente, uma das maneiras mais comuns de armazenar e compartilhar dados para análise é por meio de planilhas eletrônicas. Uma planilha armazena dados em linhas e colunas. Basicamente, é uma versão em arquivo de um *data frame*. Ao salvar essa tabela em um arquivo, é necessário definir como novas linhas e colunas começam e terminam. Isso, por sua vez, define as células nas quais os valores individuais são armazenados.

Ao criar planilhas com arquivos de texto, como aqueles criados com um editor de texto simples, uma nova linha é definida com uma quebra de linha (*enter*) e as colunas são separadas com um caractere especial predefinido. Os caracteres mais comuns são vírgulas (,), ponto e vírgula (;), espaço () e tabulação (um número predeterminado de espaços ou \t). Aqui está um exemplo da aparência de um arquivo separado por vírgula se o abrirmos com um editor de texto básico:

murders.csv			
state	abb	region	population, total
Alabama	AL	South	4779736, 135
Alaska	AK	West	710231, 19
Arizona	AZ	West	6392017, 232
Arkansas	AR	South	2915918, 93
California	CA	West	7253956, 1257
Colorado	CO	West	5023219, 65
Connecticut	CT	Northeast	574097, 97
Delaware	DE	South	897934, 38
District of Columbia	DC	South	601723, 99
Florida	FL	South	19687653, 669
Georgia	GA	South	9920000, 376
Hawaii	HI	West	1360301, 7
Idaho	ID	West	1567582, 12
Illinois	IL	North Central	12836632, 364
Indiana	IN	North Central	6483802, 142
Iowa	IA	North Central	3046355, 21
Kansas	KS	North Central	2903000, 63
Kentucky	KY	South	430367, 116
Louisiana	LA	South	4533372, 351
Maine	ME	Northeast	1328361, 11
Maryland	MD	South	5773552, 293
Massachusetts	MA	Northeast	6547629, 118
Michigan	MI	North Central	9883640, 413
Minnesota	MN	North Central	5303925, 53
Mississippi	MS	South	2967297, 120
Missouri	MO	North Central	5988927, 321
Montana	MT	West	989415, 12
Nebraska	NE	North Central	1826341, 32
Nevada	NV	West	2700551, 84

A primeira linha contém os nomes das colunas. Nos referimos a isso como o cabeçalho (*header* em inglês). Quando lemos dados de uma planilha, é importante saber se o arquivo tem um cabeçalho ou não. A maioria das funções de leitura assume que existe um cabeçalho. Para descobrir se o arquivo possui ou não cabeçalho, deve-se abrir o arquivo antes de tentar lê-lo. Isso pode ser feito usando um editor de texto ou usando o RStudio. No RStudio, podemos fazer isso abrindo o arquivo no editor ou navegando até o local do arquivo, clicando duas vezes sobre o arquivo e pressionando *View File*.

No entanto, nem todos os arquivos de planilha estão em formato de texto. O Google Sheets, por exemplo, permite acessar planilhas por um navegador. Outro exemplo é o formato

proprietário usado pelo Microsoft Excel. Planilhas nesse formato não podem ser exibidas em um editor de texto. Apesar disso, devido à popularidade do Microsoft Excel, esse formato é amplamente utilizado.

Começamos este capítulo descrevendo as diferenças entre arquivos de texto (ASCII), Unicode e binários, mostrando como essa diferença afeta a maneira como importamos tais arquivos. Em seguida, explicamos os conceitos de caminhos de arquivos e diretórios de trabalho, que são essenciais para entender como importar dados de maneira eficaz. Então, apresentamos os pacotes **readr** e **readxl**, além das funções disponíveis para importar planilhas para o R. Por fim, oferecemos algumas recomendações sobre como armazenar e organizar dados em arquivos. Desafios mais complexos, no entanto, como extrair dados de páginas da Web ou documentos em PDF, serão discutidos na seção do livro *Wrangling data*.

5.1 Caminhos e diretório de trabalho

A primeira etapa na importação de dados de uma planilha é localizar o arquivo que contém os dados. Embora não seja muito recomendável, você pode usar uma abordagem semelhante à usada para abrir arquivos no Microsoft Excel, clicando no menu “Arquivo” (*File*) no RStudio, clicando em “Importar conjunto de dados” (*Import Dataset*) e depois navegando por pastas até encontrar o arquivo. Queremos que vocês estejam aptos a desenvolver códigos em vez de apenas selecionar e clicar em pastas. As chaves e os conceitos que precisaremos para aprender a fazer isso são descritos em detalhes na seção “Ferramentas de produtividade” deste livro. Aqui, fornecemos uma visão geral do básico.

O principal desafio desta primeira etapa é permitir que as funções R de importação saibam onde procurar o arquivo que contém os dados. A maneira mais fácil de fazer isso é ter uma cópia do arquivo na pasta em que as funções de importação pesquisam por padrão. Depois disso, basta fornecer o nome do arquivo para a função de importação.

O pacote **dslabs** inclui uma planilha contendo dados de assassinatos nos EUA. A localização desse arquivo não é óbvia, mas as seguintes linhas de código copiam o arquivo para a pasta que R procura por padrão. Abaixo, explicamos como essas linhas funcionam.

```
filename <- "murders.csv"
dir <- system.file("extdata", package = "dslabs")
fullpath <- file.path(dir, filename)
file.copy(fullpath, "murders.csv")
```

Esse código não importa os dados para o R, apenas copia um arquivo. Entretanto, uma vez que o arquivo é copiado, podemos importar os dados com apenas uma linha de código. Aqui, usamos a função **read_csv** do pacote **readr**, que faz parte do **tidyverse**.

```
library(tidyverse)
dat <- read_csv(filename)
```

Os dados são importados e armazenados em **dat**. O restante desta seção define alguns conceitos importantes e fornece uma visão geral de como escrevemos códigos para que o R possa encontrar os arquivos que queremos importar. O capítulo ?? fornece mais detalhes sobre esse tópico.

5.1.1 Sistema de arquivos

Você pode pensar no sistema de arquivos do seu computador como uma série de pastas aninhadas, cada uma contendo outras pastas e arquivos. Cientistas de dados se referem às pastas como *diretórios*. Nos referimos à pasta que contém todas as outras pastas como o diretório raiz (*root*). O diretório em que estamos localizados é chamado de diretório de trabalho (*working directory*). Portanto, o diretório de trabalho muda à medida que você percorre as pastas: pense nele como sua localização atual.

5.1.2 Caminhos relativos e absolutos

O endereço ou caminho (*path* em inglês) de um arquivo é uma lista de nomes de diretórios que podem ser considerados instruções sobre qual ordem de pastas clicar para localizar o arquivo. Se essas instruções forem para encontrar o arquivo no diretório raiz, vamos nos referir a elas como caminho absoluto (ou caminho completo). Se as instruções forem para encontrar o arquivo com base no diretório atual, nós referimos a isso como caminho relativo. A seção ?? oferece mais detalhes sobre esse tópico.

Para ver um exemplo de caminho completo em seu sistema, digite o seguinte:

```
system.file(package = "dslabs")
```

Os caracteres separados por barras são os nomes dos diretórios. A primeira barra representa o diretório raiz e sabemos que esse é um caminho completo porque começa com uma barra. Se o primeiro nome do diretório aparecer sem uma barra, R assume que o caminho é relativo. Podemos usar a função `list.files` para ver exemplos de rotas relativas:

```
dir <- system.file(package = "dslabs")
list.files(path = dir)
#> [1] "data"          "DESCRIPTION"   "extdata"      "help"
#> [5] "html"          "INDEX"        "Meta"         "NAMESPACE"
#> [9] "R"             "script"
```

Esses caminhos relativos nos fornecem a localização dos arquivos ou diretórios se iniciarmos no diretório com o caminho completo. Por exemplo, o caminho completo para o diretório `help` no exemplo acima, é: `/Library/Frameworks/R.framework/Versions/3.5/Resources/library/dslabs/help`.

Nota: você provavelmente não fará muito uso da função `system.file` no trabalho diário em análise de dados. Apresentamos isso nesta seção para facilitar o compartilhamento de planilhas, uma vez que as incluímos no pacote **dslabs**. Raramente você terá o luxo de dados sendo incluído nos pacotes já instalados. No entanto, você geralmente precisa navegar por caminhos completos e relativos e importar dados no formato de planilha.

5.1.3 O diretório de trabalho

Fortemente recomendamos escrever apenas caminhos relativos no seu código, pois os caminhos completos são exclusivos para o seu computador e você deve desejar que seu código seja portátil. Você pode obter o caminho completo do diretório de trabalho usando a função `getwd`:

```
wd <- getwd()
```

Se você precisar alterar o diretório de trabalho, poderá usar a função `setwd` ou pode alterá-lo através do RStudio, clicando em “Sessão” (*Session*).

5.1.4 Gerando nomes de caminhos

Outro exemplo de como obter um caminho completo sem escrever explicitamente foi apresentado acima quando criamos o objeto `fullpath` desta maneira:

```
filename <- "murders.csv"
dir <- system.file("extdata", package = "dslabs")
fullpath <- file.path(dir, filename)
```

A função `system.file` fornece o caminho completo da pasta que contém todos os arquivos e diretórios relevantes para o pacote especificado pelo argumento `package`. Ao procurar diretórios em `dir`, descobrimos que `extdata` contém o arquivo que queremos:

```
dir <- system.file(package = "dslabs")
filename %in% list.files(file.path(dir, "extdata"))
#> [1] TRUE
```

A função `system.file` permite que forneçamos um subdiretório como primeiro argumento, assim podemos obter o caminho completo do diretório `extdata` assim:

```
dir <- system.file("extdata", package = "dslabs")
```

A função `file.path` é usada para combinar os nomes dos diretórios para produzir o caminho completo do arquivo que queremos importar.

```
fullpath <- file.path(dir, filename)
```

5.1.5 Copiando arquivos usando caminhos

A última linha do código que utilizamos para copiar o arquivo em nosso diretório inicial usou a função `file.copy`. Isso requer dois argumentos: o nome do arquivo a ser copiado e o nome a ser usado no novo diretório.

```
file.copy(fullpath, "murders.csv")
#> [1] TRUE
```

Se o arquivo for copiado com sucesso, a função `file.copy` retorna a mensagem `TRUE` (verdadeiro). Note que estamos dando o mesmo nome ao arquivo, `murders.csv`, mas poderíamos ter dado qualquer outro nome a ele. Note também que, ao não iniciar a sequência com uma barra, R assume que esse é um caminho relativo e copia o arquivo para o diretório de trabalho.

Você deve poder visualizar o arquivo em seu diretório de trabalho usando:

```
list.files()
```

5.2 Os pacotes `readr` e `readxl`

Nesta seção, apresentamos as principais funções de importação do *tidyverse*. Vamos usar o arquivo `murders.csv` do pacote `dslabs` como um exemplo. Para simplificar a ilustração, copiaremos o arquivo para nosso diretório de trabalho usando o seguinte código:

```
filename <- "murders.csv"
dir <- system.file("extdata", package = "dslabs")
fullpath <- file.path(dir, filename)
file.copy(fullpath, "murders.csv")
```

5.2.1 `readr`

O pacote `readr` inclui funções para ler dados de planilhas armazenados em arquivos de texto. `readr` faz parte do pacote `tidyverse`, ou você pode carregá-lo diretamente assim:

```
library(readr)
```

As seguintes funções estão disponíveis para leitura de planilhas:

Função	Formato	Extensão
<code>read_table</code>	valores separados por espaços em branco	txt
<code>read_csv</code>	valores separados por vírgula	csv
<code>read_csv2</code>	valores separados por ponto e vírgula	csv
<code>read_tsv</code>	valores separados por tabulação	tsv
<code>read_delim</code>	formato geral de arquivo de texto, você deve definir delimitador	txt

Embora a extensão geralmente nos diga que tipo de arquivo é, não há garantia de que eles sempre correspondam. Podemos abrir o arquivo para dar uma olhada ou usar a função `read_lines` para ver algumas linhas:

```
read_lines("murders.csv", n_max = 3)
#> [1] "state,abb,region,population,total"
#> [2] "Alabama,AL,South,4779736,135"
#> [3] "Alaska,AK,West,710231,19"
```

Isso também mostra que existe um cabeçalho. Agora estamos prontos para ler dados em R. A partir da extensão .csv e da visualização das três primeiras linhas do arquivo, sabemos que precisamos usar `read_csv`:

```
dat <- read_csv(filename)
#>
#> -- Column specification --
#> cols(
#>   state = col_character(),
#>   abb = col_character(),
#>   region = col_character(),
#>   population = col_double(),
```

```
#>   total = col_double()
#> )
```

Observe que recebemos uma mensagem informando que tipos de dados foram usados para cada coluna. Observe também que `dat` é um `tibble`, não apenas um *data frame*. Isso ocorre porque `read_csv` é um *parser* (analisador) do *tidyverse*. Podemos confirmar que os dados foram lidos da seguinte forma:

```
View(dat)
```

Por fim, note também que podemos usar o caminho completo para o arquivo:

```
dat <- read_csv(fullpath)
```

5.2.2 readxl

Você pode carregar o pacote `readxl` usando:

```
library(readxl)
```

O pacote oferece funções para a leitura de arquivos nos formatos do Microsoft Excel:

Função	Formato	Extensão
<code>read_excel</code>	detecta automaticamente o formato	<code>xls</code> , <code>xlsx</code>
<code>read_xls</code>	formato original	<code>xls</code>
<code>read_xlsx</code>	novo formato	<code>xlsx</code>

Os formatos do Microsoft Excel permitem que você tenha mais de uma planilha em um arquivo. As funções listadas acima leem por padrão a primeira folha, mas também podemos ler as outras. A função `excel_sheets` nos fornece os nomes de todas as planilhas em um arquivo do Excel. Esses nomes podem ser passados pelo argumento `sheet` das três funções apresentadas anteriormente, permitindo assim a leitura de outras planilhas do arquivo além da primeira.

5.3 Exercícios

1. Use a função `read_csv` para ler cada um dos arquivos que o código a seguir armazena no objeto `files`:

```
path <- system.file("extdata", package = "dslabs")
files <- list.files(path)
files
```

2. Observe que o último arquivo (`olive`), retorna um aviso. Isso ocorre porque a primeira linha do arquivo não contém o cabeçalho da primeira coluna.

Leia a página de ajuda para `read_csv` para aprender a ler o arquivo sem ler esse cabeçalho. Se você omitir o cabeçalho, você não deverá receber esse aviso. Salve o resultado em um objeto chamado `dat`.

3. Um problema com a abordagem acima é que não sabemos o que as colunas representam. Digite:

```
names(dat)
```

para confirmar que os nomes não são informativos.

Use a função `readLines` para ler apenas a primeira linha (mais tarde aprenderemos como extrair valores do *output*).

5.4 Como fazer *download* de arquivos

Outro local comum onde os dados residem é na internet. Quando esses dados estão em arquivos, podemos baixá-los e importá-los, ou mesmo lê-los diretamente da web. Por exemplo, observamos que, como o pacote `dslabs` está no GitHub, o arquivo que baixamos com o pacote tem uma URL:

```
url <- "https://raw.githubusercontent.com/rafalab/dslabs/master/inst/
extdata/murders.csv"
```

A função `read_csv` pode ler esses arquivos diretamente:

```
dat <- read_csv(url)
```

Se deseja ter uma cópia local do arquivo, você pode usar a função `download.file`:

```
download.file(url, "murders.csv")
```

Isso fará o *download* do arquivo e o salvará no seu sistema com o nome `murders.csv`. Você pode usar qualquer nome aqui, não necessariamente `murders.csv`. Tenha cuidado ao usar o comando `download.file`, pois ele **irá substituir arquivos existentes com mesmo nome sem aviso prévio**.

Duas funções que às vezes são úteis ao baixar dados da internet são `tempdir` e `tempfile`. A primeira cria um diretório com um nome aleatório e provavelmente único. Da mesma forma, `tempfile` também cria um nome aleatório que será usado para renomear o arquivo. Você pode executar o seguinte comando para excluir os arquivos temporários depois que os dados forem importados:

```
tmp_filename <- tempfile()
download.file(url, tmp_filename)
dat <- read_csv(tmp_filename)
file.remove(tmp_filename)
```

5.5 Funções básicas de importação do R

O pacote base do R também fornece funções de importação. Eles têm nomes semelhantes aos do *tidyverse*, por exemplo, `read.table`, `read.csv` e `read.delim`. No entanto, existem

algumas diferenças importantes. Para mostrar isso, vamos ler os dados com uma função básica do R:

```
dat2 <- read.csv(filename)
```

Uma diferença importante é que os caracteres são convertidos em fatores (*factors*):

```
class(dat2$abb)
#> [1] "character"
class(dat2$region)
#> [1] "character"
```

Isso pode ser evitado definindo o argumento `stringsAsFactors` como `FALSE`.

```
dat <- read.csv("murders.csv", stringsAsFactors = FALSE)
class(dat$state)
#> [1] "character"
```

Em nossa experiência, isso pode ser motivo de confusão, uma vez que uma variável que foi salva como caracteres no arquivo se é obrigatoriamente convertida em *factor*, independente do que a variável represente. De fato, é **altamente** recomendável definir `stringsAsFactors=FALSE` para ser a abordagem padrão ao usar o *parser* da base R. Você pode facilmente converter colunas desejadas em fatores após a importação de dados.

5.5.1 `scan`

Ao ler planilhas, muitas coisas podem dar errado. O arquivo pode ter um cabeçalho de múltiplas linhas, algumas células podem estar ausentes ou o arquivo pode estar utilizando uma codificação inesperada¹. Recomendamos que você leia este artigo sobre problemas comuns com caracteres: <https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-every-software-developer-absolutely-positively-must-know-about-unicode-and-character-sets-no-excuses/>.

Com a experiência, você aprenderá a lidar com diferentes desafios. Além disso, ler atentamente os arquivos de ajuda para as funções discutidas aqui poderá ser útil. Com a função `scan` você pode ler cada célula de um arquivo. Aqui está um exemplo:

```
path <- system.file("extdata", package = "dslabs")
filename <- "murders.csv"
x <- scan(file.path(path, filename), sep=",", what = "c")
x[1:10]
#> [1] "state"      "abb"        "region"     "population" "total"
#> [6] "Alabama"    "AL"         "South"      "4779736"    "135"
```

Note que o *tidyverse* fornece `read_lines`, uma função igualmente útil.

5.6 Arquivos de texto versus arquivos binários

Na ciência de dados, arquivos geralmente podem ser classificados em duas categorias: arquivos de texto (também conhecidos como arquivos ASCII) e arquivos binários. Você já tem

¹https://pt.wikipedia.org/wiki/Codifica%C3%A7%C3%A3o_de_caracteres

trabalhado com arquivos de texto. Todos os seus *scripts R* são arquivos de texto, assim como os arquivos *R markdown* usados para criar este livro. As tabelas csv que você leu também são arquivos de texto. Uma grande vantagem desses arquivos é que podemos facilmente “olhá-los”, sem precisar comprar nenhum tipo de software especial ou seguir instruções complicadas. Qualquer editor de texto pode ser usado para navegar em um arquivo de texto, incluindo editores disponíveis gratuitamente, como RStudio, Notepad, textEdit, vi, emacs, nano e pico. Para ver isso, tente abrir um arquivo csv com a ferramenta do RStudio “*Open file*” (abrir arquivo). Você deverá poder ver o conteúdo diretamente no editor. No entanto, se tentar abrir, digamos, um arquivo xls do Excel, ou imagem nos formatos jpg ou png, você não poderá ver nada imediatamente útil. Esses são arquivos binários. Os arquivos do Excel são, na verdade, pastas compactadas com vários arquivos de texto. A principal distinção aqui é que os arquivos de texto podem ser facilmente navegados.

Embora R inclua ferramentas para ler arquivos binários amplamente utilizados, como arquivos xls, geralmente é melhor encontrar conjuntos de dados armazenados em arquivos de texto. Da mesma forma, ao compartilhar dados, é melhor disponibilizá-los como arquivos de texto, desde que o armazenamento não seja um problema (os arquivos binários são mais eficientes para economizar de espaço em sua unidade de armazenamento). Em geral, os formatos de texto facilitam a troca de dados, pois não requerem *software* comercial para trabalhar com os dados.

Extrair dados de uma planilha armazenada como um arquivo de texto é talvez a maneira mais fácil de trazer dados de um arquivo para uma sessão R. Infelizmente, planilhas nem sempre estão disponíveis e o fato de você poder visualizar arquivos de texto não implica necessariamente que será fácil extrair dados dele. Na parte *_ Data Wrangling_* do livro, aprenderemos como extrair dados de arquivos de texto mais complexos, como arquivos html.

5.7 Unicode versus ASCII

Uma armadilha na ciência de dados é assumir que um dado arquivo se trata de um arquivo de texto ASCII, quando na verdade é outra coisa que pode se parecer muito com um arquivo de texto ASCII: um arquivo de texto Unicode.

Para entender a diferença entre eles, lembre-se de que tudo no computador precisa ser convertido em 0s e 1s. ASCII é uma codificação que define uma correspondência entre caracteres e números. O ASCII usa 7 bits (0s e 1s), o que resulta em $2^7 = 128$ elementos únicos, que é suficiente para codificar todos os caracteres de um teclado do idioma inglês. No entanto, outros idiomas, como o português, usam caracteres não incluídos nessa codificação. Por exemplo, o é na palavra México não é codificado em ASCII. Por esse motivo, foi definida uma nova codificação que usa mais do que 7 bits: Unicode. Ao usar o Unicode, você pode escolher entre as codificações UTF-8, UTF-16 ou UTF-32 que utilizam 8, 16 ou 32 bits, respectivamente. O RStudio usa a codificação UTF-8 por padrão.

Embora não entremos em detalhes sobre como lidar com diferentes codificações aqui, é importante que você saiba que existem codificações diferentes para que você possa diagnosticar melhor caso encontre um problema. Um exemplo de problema é quando surgem caracteres “de aparência estranha” e que você não esperava. Esta discussão do StackOverflow é um exemplo: <https://stackoverflow.com/questions/18789330/r-on-windows-character-encoding-hell>.

5.8 Como organizar dados com planilhas

Embora este livro se concentre quase exclusivamente na análise de dados, o gerenciamento de dados também é uma parte importante da ciência de dados. Como explicamos na introdução, não cobrimos esse tema. No entanto, muitas vezes analistas de dados precisam realizar coletas ou trabalhar com outras pessoas que coletam dados, portanto, a maneira mais conveniente de armazená-los é em uma planilha. Embora o preenchimento manual de uma planilha seja uma prática que não recomendamos e preferimos que o processo seja automatizado o máximo possível, às vezes não há outra opção. Portanto, nesta seção, oferecemos recomendações sobre como organizar dados em uma planilha. Embora existam pacotes R projetados para ler planilhas do Microsoft Excel, geralmente devemos evitar esse formato. Recomendamos o *Google Sheets* por ser uma ferramenta gratuita. Abaixo resumimos as recomendações feitas em um artigo publicado por Karl Broman e Kara Woo². Para obter mais detalhes, leia o artigo completo.

- **Seja consistente** - Antes de começar a inserir dados, tenha um plano. Depois de estabelecer o plano, seja consistente e siga-o.
- **Escolha bons nomes para as coisas**: os nomes que você escolher para objetos, arquivos e diretórios devem ser memoráveis, fáceis de soletrar e descriptivos. Esse é um equilíbrio difícil de alcançar e requer tempo e reflexão. Uma regra importante a seguir é **não use espaços**, use sublinhados _ ou hífens -. Além disso, evite símbolos e caracteres especiais (como acentos e cedilha); é melhor usar letras e números.
- **Insira as datas como AAAA-MM-DD** - Para evitar confusão, recomendamos o uso do padrão global ISO 8601.
- **Evite células vazias** - Preencha todas as células. Você pode usar algum código comum para representar dados ausentes.
- **Coloque apenas uma coisa em cada célula** - É melhor adicionar colunas para armazenar informações adicionais em vez de ter mais de uma informação em uma célula.
- **Faça um retângulo** - A planilha deve ser um retângulo.
- **Crie um dicionário de dados** - Se você precisar explicar as coisas, por exemplo, quais são as colunas ou os rótulos usados para variáveis categóricas, faça isso em um arquivo separado.
- **Não faça cálculos em arquivos de dados brutos** - O Excel permite que você faça cálculos. Não faça disso parte de sua planilha. Códigos para realização de cálculos devem estar em um *script* separado.
- **Não use cores ou realces como dados** - A maioria das funções de importação não consegue importar essas informações. Em vez disso, codifique-as como variáveis.
- **Faça backup de suas informações**: faça backup de seus dados com frequência.
- **Utilize a validação de dados para evitar erros** - Aproveite as ferramentas de seu programa de edição de planilhas para tornar o processo o mais livre possível de erros e de lesões por esforço repetitivo.
- **Salve dados como arquivos de texto** - Salve os arquivos para compartilhar em formato delimitado por vírgula ou tabulações.

²<https://www.tandfonline.com/doi/abs/10.1080/00031305.2017.1375989>

5.9 Exercícios

1. Escolha uma medida que você possa mensurar regularmente. Por exemplo, seu peso diário ou quanto tempo leva para percorrer 8 km. Mantenha uma planilha que inclua data, hora, medida e outras variáveis informativas que considere valiosas. Faça isso por 2 semanas. Então faça um gráfico.

Part II

Exibição de dados

6

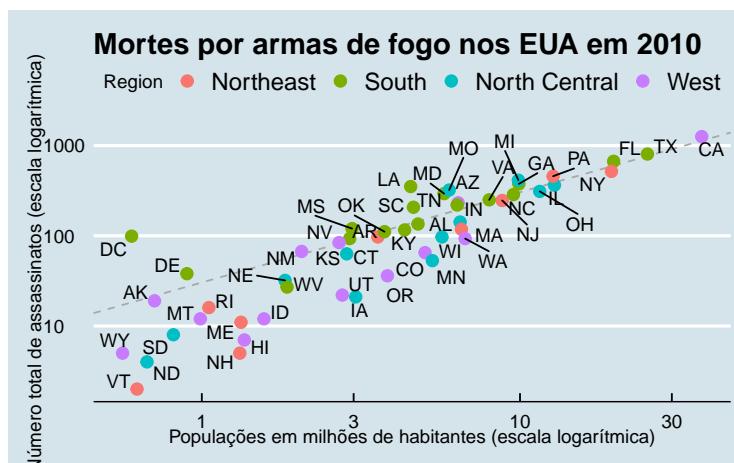
Introdução à visualização de dados

Observar os números e as *strings* que definem um conjunto de dados é raramente útil. Para se convencer disso, imprima e observe a tabela de dados de assassinato nos Estados Unidos:

```
library(dslabs)
data(murders)
head(murders)

#>      state abb region population total
#> 1    Alabama AL   South     4779736  135
#> 2    Alaska AK   West      710231   19
#> 3  Arizona AZ   West     6392017  232
#> 4 Arkansas AR   South    2915918   93
#> 5 California CA   West    37253956 1257
#> 6 Colorado CO   West    5029196   65
```

O que você aprende ao ver essa tabela? Com que rapidez você pode determinar quais estados têm as maiores populações? Quais estados têm as menores? Qual o tamanho populacional típico de um estado? Existe alguma relação entre o tamanho da população e o total de assassinatos? Como as taxas de homicídio variam entre as regiões do país? Para a maioria dos cérebros humanos, é bastante difícil extrair essas informações simplesmente observando os números. Em vez disso, as respostas para todas as perguntas acima estão prontamente disponíveis examinando este gráfico:



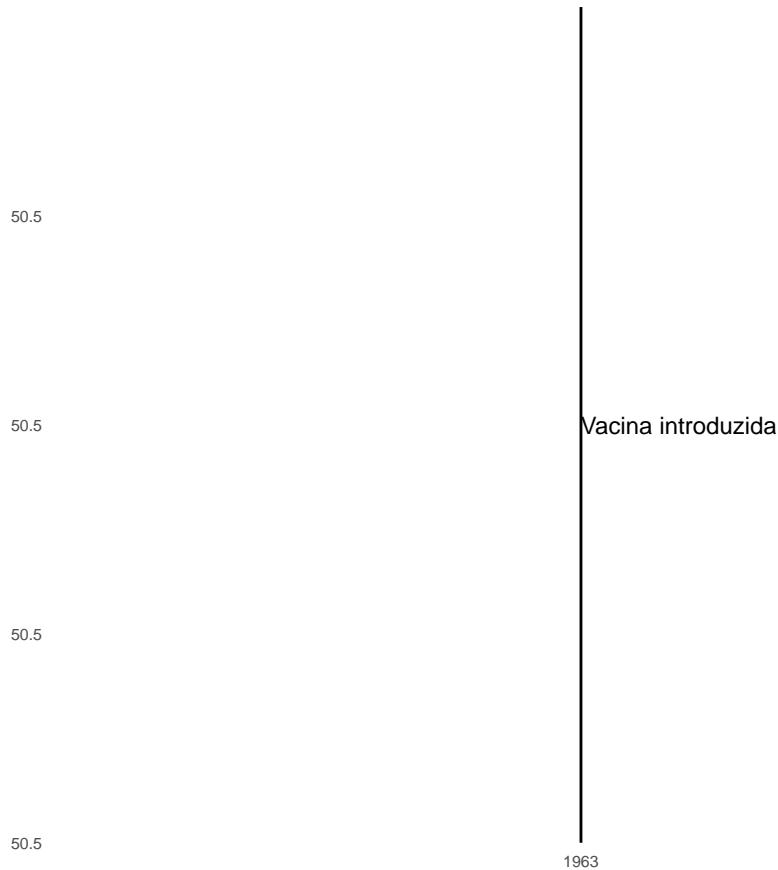
Isso nos lembra o ditado “uma imagem vale mais que mil palavras”. A visualização de dados fornece uma poderosa forma de comunicar descobertas com base em dados. Em alguns casos, a visualização é tão convincente que nenhuma análise complementar é necessária.

A crescente disponibilidade de conjuntos de dados informativos e ferramentas de *software* levou a uma maior dependência de visualizações de dados em muitos setores da indústria,

academia e governo. Um excelente exemplo são as organizações de notícias, que estão cada vez mais adotando o jornalismo de dados e incluindo infográficos eficazes como parte de seus relatórios.

Um exemplo particularmente eficaz é um artigo do *Wall Street Journal*¹ mostrando dados relacionados ao impacto das vacinas na luta contra a doenças infecciosas. Um dos gráficos mostra casos de sarampo por estado dos EUA ao longo dos anos com uma linha vertical indicando quando a vacina foi introduzida.

Sarampo



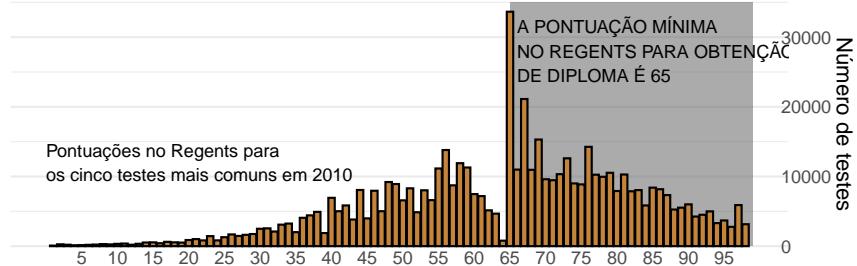
Outro exemplo notável vem de um gráfico do jornal *New York Times*² que resume os resultados dos *Regents Exams* da cidade de Nova York. De acordo com o artigo³, essas pontuações são coletadas por vários motivos, inclusive para determinar se um aluno está se formando no ensino médio. Na cidade de Nova York, é necessária uma pontuação mínima de 65 para passar. A distribuição dos resultados dos testes nos obriga a notar algo um pouco problemático:

¹http://graphics.wsj.com/infectious-diseases-and-vaccines/?mc_cid=711ddeb86e

²<http://graphics8.nytimes.com/images/2011/02/19/nyregion/19schoolsch/19schoolsch-popup.gif>

³<https://www.nytimes.com/2011/02/19/nyregion/19schools.html>

Aprovados com a nota mínima



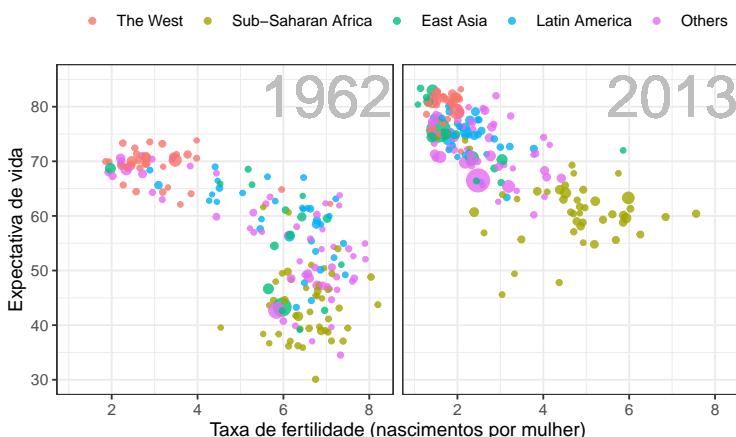
A pontuação mais comum no teste é a nota mínima para aprovação, com muito poucas notas logo abaixo desse limiar. Esse inesperado resultado é consistente com a hipótese de que alunos com notas próximas da aprovação tiveram suas pontuações aumentadas.

Esse é um exemplo de como a visualização de dados pode levar a descobertas que, de outra forma, seriam perdidas se simplesmente submetéssemos os dados a uma série de ferramentas ou procedimentos de análise de dados. A visualização de dados é a ferramenta mais eficaz na chamada “análise exploratória de dados”, ou no inglês *exploratory data analysis* (EDA). John W. Tukey⁴, considerado o pai da EDA, disse uma vez:

“O maior valor de uma imagem é quando ela nos força a perceber o que nunca esperávamos ver”.

Muitas das ferramentas de análise de dados mais usadas foram inicialmente desenvolvidas graças à análise exploratória de dados. Esta é talvez a parte mais importante da análise de dados, no entanto, é frequentemente ignorada.

Visualização de dados agora é onipresente também em organizações filantrópicas e educacionais. Nas apresentações *New Insights on Poverty*⁵ (na tradução “Novas Percepções sobre a Pobreza”) e *The Best Stats You’ve Ever Seen*⁶ (na tradução “As Melhores Estatísticas que Você Nunca Viu”), Hans Rosling nos obriga a perceber o inesperado com uma série de gráficos relacionados à saúde e à economia global. Em seus vídeos, Rosling usa gráficos animados para demonstrar como o mundo está mudando e como antigas narrativas não são mais verdadeiras.



⁴https://en.wikipedia.org/wiki/John_Tukey

⁵https://www.ted.com/talks/hans_rosling_reveals_new_insights_on_poverty?language=en

⁶https://www.ted.com/talks/hans_rosling_shows_the_best_stats_you_ve_ever_seen

Também é importante lembrar que equívocos, preconceitos, erros sistemáticos e outros problemas inesperados frequentemente levam a dados que devem ser cuidadosamente analisados. Fracassar em descobrir tais problemas pode levar a análises falhas e descobertas falsas. Como exemplo, considere que instrumentos de medição algumas vezes falham e que a maioria dos procedimentos de análise de dados não foi projetada para detectar tais falhas. Contudo, esses procedimentos de análise de dados ainda lhe darão uma resposta. O fato de que possa ser difícil, ou mesmo impossível, perceber um erro apenas com base nos resultados relatados fazem da visualização de dados particularmente importante.

Nesta parte do livro, aprenderemos os conceitos básicos de visualização de dados e análise exploratória de dados usando três exemplos motivadores. Usaremos o pacote **ggplot2** para codificar. Para aprender o básico, usaremos um exemplo um tanto artificial: as alturas relatadas por estudantes. Em seguida, discutiremos dois exemplos mencionados anteriormente: 1) saúde e economia mundiais, e 2) tendências de doenças infecciosas nos Estados Unidos.

Obviamente, há muito mais na visualização de dados do que o que abordamos aqui. Aqui estão algumas referências para quem deseja aprender mais:

- ER Tufte (1983) *The visual display of quantitative information*. Graphics Press.
- ER Tufte (1990) *Envisioning information*. Graphics Press.
- ER Tufte (1997) *Visual explanations*. Graphics Press.
- WS Cleveland (1993) *Visualizing data*. Hobart Press.
- WS Cleveland (1994) *The elements of graphing data*. CRC Press.
- A Gelman, C Pasarica, R Dodhia (2002) *Let's practice what we preach: Turning tables into graphs*. The American Statistician 56:121-130.
- NB Robbins (2004) *Creating more effective graphs*. Wiley.
- A Cairo (2013) *The functional art: An introduction to information graphics and visualization*. New Riders.
- N Yau (2013) *Data points: Visualization that means something*. Wiley.

Por fim, não discutiremos gráficos interativos, um tópico muito avançado para este livro. Abaixo estão alguns recursos úteis para aqueles interessados em aprender mais sobre isso:

- <https://shiny.rstudio.com>
- <https://d3js.org>

7

ggplot2

A visualização exploratória de dados é talvez a maior vantagem do R. Com R pode-se rapidamente converter uma ideia em um gráfico, manipulando os dados com um balanço único entre flexibilidade e facilidade. Por exemplo, usar o Excel pode ser mais fácil do que o R para alguns gráficos, mas não é nem de longe tão flexível. D3.js pode ser mais flexível e poderoso que o R, mas leva muito mais tempo para gerar um gráfico.

Ao longo deste livro, criaremos gráficos usando o pacote **ggplot2**¹.

```
library(dplyr)  
library(ggplot2)
```

Muitos outros métodos para criar gráficas estão disponíveis no R. De fato, os recursos gráficos que vêm com uma instalação básica do R já são bastante poderosos. Também existem outros pacotes para criar gráficos como o **grid** e o **lattice**. Neste livro, decidimos usar o **ggplot2** porque ele divide gráficos em componentes que permitem a iniciantes criar gráficos relativamente complexos e esteticamente agradáveis usando sintaxe intuitiva e relativamente fácil de lembrar.

Um dos motivos pelo qual **ggplot2** é considerado mais intuitivo para iniciantes é o fato de ele usar uma gramática gráfica², o *gg* em **ggplot2**. Isso é análogo à forma como a aprender gramática pode ajudar um iniciante a criar centenas de diferentes frases apenas aprendendo um punhado de verbos, substantivos e adjetivos, em vez de memorizar cada frase específica. Da mesma forma, aprendendo uma pequena quantidade dos componentes básicos do **ggplot2** e sua gramática, você poderá criar centenas de gráficos diferentes.

Outra razão pela qual **ggplot2** é fácil para iniciantes é que seu comportamento padrão foi cuidadosamente escolhido para satisfazer a grande maioria dos casos e também é visualmente agradável. Como resultado, é possível criar gráficos informativos e elegantes com código relativamente simples e legível.

Uma limitação do **ggplot2** é que ele foi projetado para funcionar exclusivamente com tabelas de dados no formato *tidy* (onde linhas são observações e colunas são variáveis). No entanto, uma porcentagem substancial dos *datasets* com os quais os iniciantes trabalham estão nesse formato ou podem ser convertidos para ele. Uma vantagem dessa abordagem é que, desde que nossos dados estejam “arrumados” (*tidy* em inglês), **ggplot2** simplifica o aprendizado de código e gramática de plotagem para uma variedade de gráficos.

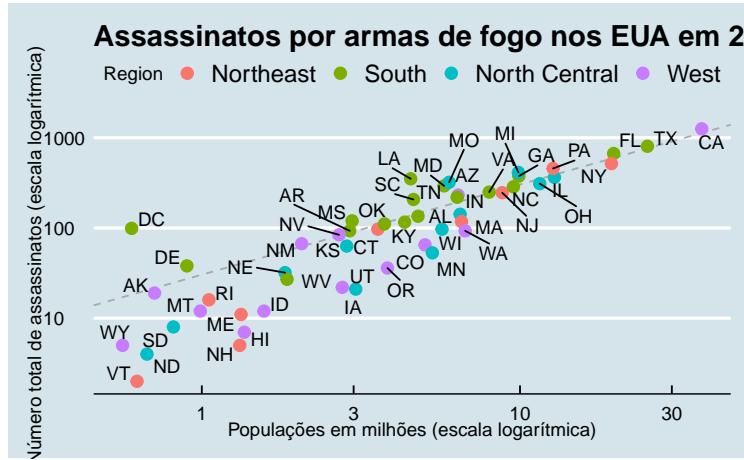
Para usar **ggplot2**, você terá que aprender uma série de funções e argumentos. Como é difícil memorizar, recomendamos que você tenha a folha de referência do ggplot2 à mão. Você pode obter uma cópia aqui: <https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf> ou simplesmente faça uma pesquisa na Internet por “*ggplot2 cheat sheet*”.

¹<https://ggplot2.tidyverse.org/>

²<http://www.springer.com/us/book/9780387245447>

7.1 Os componentes de um gráfico

Construiremos um gráfico que resume o *dataset* de assassinatos por armas nos Estados Unidos da seguinte forma:



Podemos ver claramente o quanto os estados variam de acordo com o tamanho da população e o número total de assassinatos. Não é de surpreender que também exista uma relação clara entre o total de assassinatos e o tamanho da população. Estados plotados sobre a linha cinza tracejada têm a mesma taxa de homicídios que a média dos EUA. As quatro regiões geográficas são indicadas pelas cores (*Northeast*: nordeste; *South*: sul; *North Central*: central norte; *West*: oeste), o que mostra como a maioria dos estados do sul tem taxas de homicídio acima da média.

Essa visualização de dados mostra praticamente todas as informações na tabela de dados. Além disso, o código necessário para criar esse gráfico é relativamente simples. Vamos aprender a criá-lo parte por parte.

O primeiro passo para aprender **ggplot2** é ser capaz de separar um gráfico em componentes. Começaremos examinando o gráfico acima e introduzindo algumas das terminologias do **ggplot2**. Os três principais componentes a serem considerados são:

- **Dados:** A tabela de dados de assassinatos nos EUA está sendo resumida. Nós nos referimos a isso como o componente de **dados** (*data*).
- **Geometria:** O gráfico acima é um diagrama de dispersão. Isso é chamado de componente de **geometria** (*geometry*). Outras geometrias possíveis são gráfico de barras (*barplot*), histograma, densidades suaves (*smooth densities*), gráfico Q-Q (*qqplot*) e diagrama de caixa (*boxplot*).
- **Mapeamento estético:** o gráfico usa vários sinais visuais para representar as informações fornecidas pelo *dataset*. Os dois sinais mais importantes neste gráfico são as posições dos pontos nos eixos x e y, representando o tamanho da população e o número total de assassinatos, respectivamente. Cada ponto representa uma observação diferente. Assim, mapeamos os dados dessas observações para tais sinais visuais, como por exemplo, as escalas x e y. A cor é outro sinal visual que mapeamos para a região. Nós nos referimos a isso como o componente **mapeamento estético** (em inglês *Aesthetic mapping*). A definição do mapeamento depende de qual **geometria** estamos usando.

Também observamos que:

- Os pontos são rotulados com as abreviações dos estados.
- Os intervalos dos eixos x e y parecem ser definidos pelo intervalo dos dados. Ambos estão em escalas logarítmicas.
- Existem rótulos, um título, uma lenda e usamos o estilo da revista “*The Economist*”.

Agora, vamos construir o gráfico parte por parte. Vamos começar carregando o *dataset*:

```
library(dslabs)
data(murders)
```

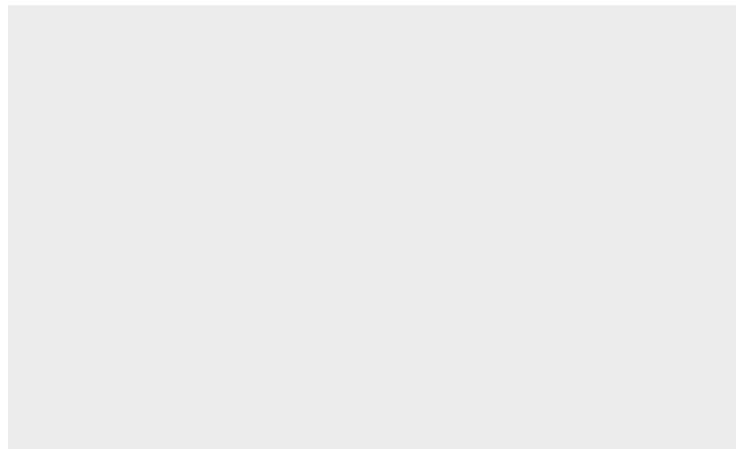
7.2 Objetos *ggplot*

A primeira etapa na criação de um gráfico **ggplot2** é definir um objeto **ggplot**. Fazemos isso com a função **ggplot**, que inicializa o gráfico. Se leremos a página de ajuda dessa função, veremos que o primeiro argumento é usado para especificar quais dados estão associados a este objeto:

```
ggplot(data = murders)
```

Também podemos usar o operador *pipe* para inserir os dados como o primeiro argumento. Portanto, linha de código abaixo é equivalente à anterior:

```
murders %>% ggplot()
```



O código acima renderiza um quadro vazio, pois nenhuma geometria foi definida. A única opção de estilo que vemos é um fundo cinza.

O que aconteceu é que o objeto foi criado, mas por não ter sido atribuído a nenhuma variável, ele foi automaticamente executado. Podemos atribuir nosso gráfico a um objeto, por exemplo, desta forma:

```
p <- ggplot(data = murders)
class(p)
#> [1] "gg"      "ggplot"
```

Para renderizar o gráfico associado a esse objeto, podemos simplesmente imprimir o objeto `p`. Cada uma das duas linhas de código a seguir produz o mesmo gráfico que vemos acima:

```
print(p)
p
```

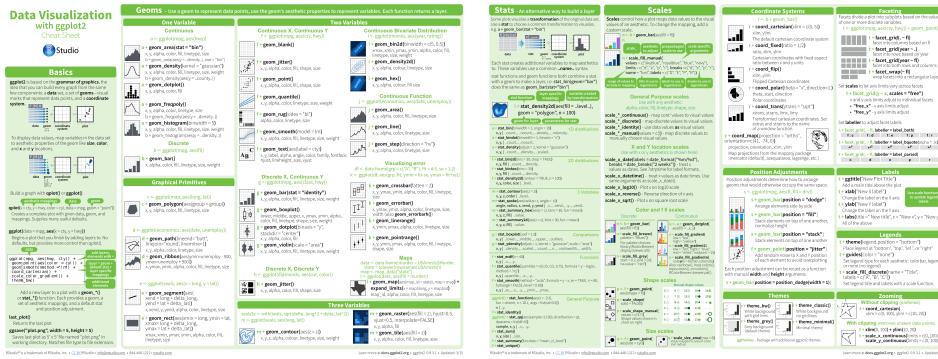
7.3 Geometrias

No `ggplot2` criamos gráficos adicionando *camadas* (*layers* em inglês). As camadas podem definir geometrias, calcular estatísticas resumidas, definir quais escalas usar ou, até mesmo, alterar estilos. Para adicionar camadas, usamos o símbolo `+`. Em geral, uma linha de código ficará assim:

DADOS %>% `ggplot()` + CAMADA 1 + CAMADA 2 + ... + CAMADA N

Geralmente, a primeira camada que adicionamos define a geometria. Por exemplo, se queremos fazer um diagrama de dispersão, qual geometria devemos usar?

Observando rapidamente a folha de referência, vemos que a função usada para criar gráficos com essa geometria é `geom_point`.



(Imagen cedida pelo RStudio³. Licença CC-BY-4.0.⁴)

Os nomes das funções de geometrias seguem o padrão: `geom_X`, onde X é o nome da geometria. Alguns exemplos incluem `geom_point`, `geom_bar` e `geom_histogram`.

Para que `geom_point` funcione bem, precisamos fornecer dados e uma correspondência. Nós já conectamos o objeto `p` com tabela de dados `murders`. Se adicionarmos a camada `geom_point`, o padrão é usar os dados que foram conectados. Para descobrir quais correspondências são esperadas, leia a seção **Aesthetics** (estética) da página de ajuda de `geom_point`:

```
> Aesthetics (Estética)
>
> _geom_point_ entende a seguinte estética (estéticas obrigatórias estão em negrito):
>
> __x__
```

³<https://github.com/rstudio/cheatsheets>

⁴<https://github.com/rstudio/cheatsheets/blob/master/LICENSE>

```
>
> __y__
>
> alpha
>
> colour
```

e, como esperado, vemos que pelo menos dois argumentos são necessários `x` e `y`.

7.4 Mapeamentos estéticos

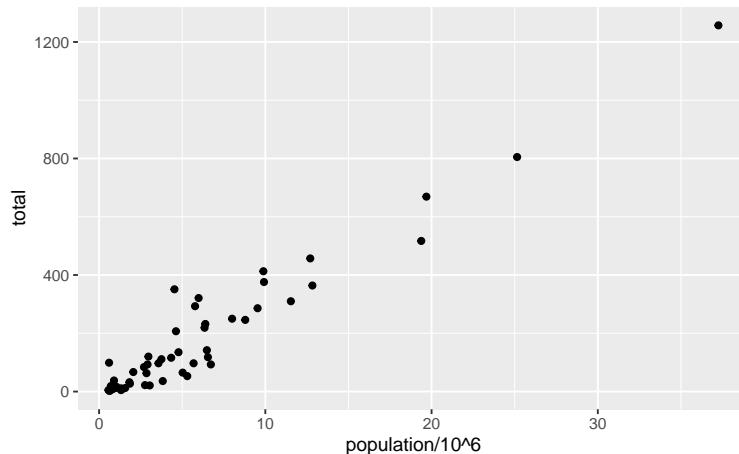
Os mapeamentos estéticos descrevem como as propriedades dos dados estão conectadas às características do gráfico, como a distância ao longo de um eixo, o tamanho ou a cor. A função `aes` conecta os dados com o que vemos no gráfico, definindo atribuições estéticas. Essa será uma das funções que mais serão usadas na representação gráfica. O resultado da função `aes` é frequentemente usado como argumento para uma função de geometria. Este exemplo produz um gráfico de dispersão do total de assassinatos versus população em milhões:

```
murders %>% ggplot() +
  geom_point(aes(x = population/10^6, y = total))
```

Podemos remover o `x =` e `y =` se quiséssemos, uma vez que esses são os dois argumentos esperados, respectivamente, como visto na página de ajuda.

Em vez de definir nosso gráfico do zero, também podemos adicionar uma camada ao objeto `p` que foi definido anteriormente como `p <- ggplot(data = murders)`:

```
p + geom_point(aes(population/10^6, total))
```



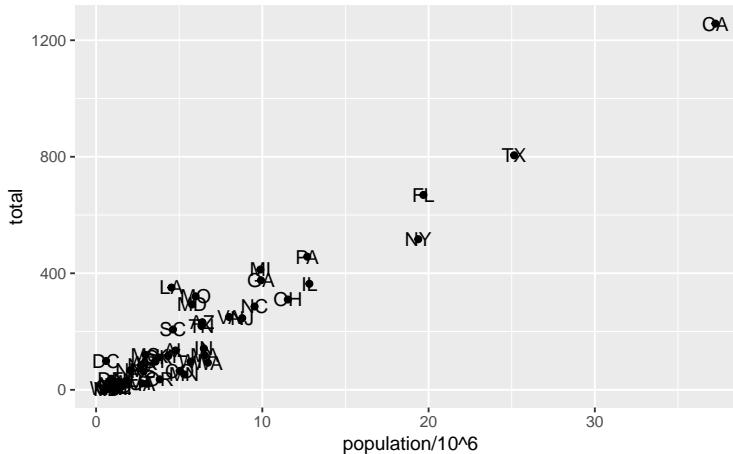
A escala e os rótulos são definidos por padrão ao adicionar essa camada. Assim como as funções `dplyr`, `aes` também usa os nomes de variáveis obtidos do componente do objeto: podemos usar `population` e `total` sem ter que chamá-los como `assassinatos$population` ou `murders$total`. O comportamento de reconhecer as variáveis de componentes de dados é bastante específico da função `aes`. Na maioria das funções, se você tentar acessar os valores de `population` ou `total` fora de `aes`, receberá uma mensagem de erro.

7.5 Camadas

Podemos criar uma segunda camada no gráfico para adicionar um rótulo a cada ponto para identificar o estado. As funções `geom_label` e `geom_text` nos permitem adicionar texto ao gráfico com ou sem um retângulo atrás do texto, respectivamente.

Como cada ponto (cada estado neste caso) possui um rótulo, precisamos de um mapeamento estético para fazer a conexão entre pontos e rótulos. Lendo a página de ajuda, aprendemos que o mapeamento entre o ponto e o rótulo é fornecido através do argumento `label` da função `aes`. Portanto, o código fica assim:

```
p + geom_point(aes(population/10^6, total)) +
  geom_text(aes(population/10^6, total, label = abb))
```



Adicionamos com sucesso uma segunda camada ao gráfico.

Como exemplo do comportamento único de `aes` mencionado acima, observe que esta chamada:

```
p_test <- p + geom_text(aes(population/10^6, total, label = abb))
```

funciona corretamente, enquanto esta chamada:

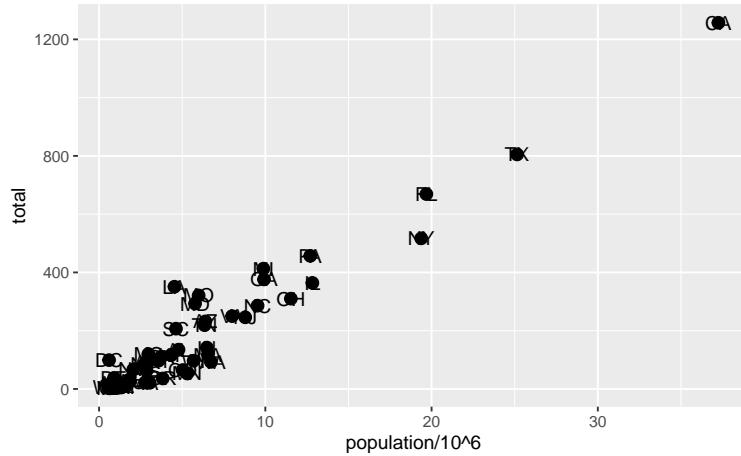
```
p_test <- p + geom_text(aes(population/10^6, total), label = abb)
```

dará um erro, uma vez que `abb` não foi encontrado por está fora da função `aes`. A camada `geom_text` não sabe onde encontrar `abb` porque é um nome de coluna e não uma variável global.

7.5.1 Trabalhando com argumentos

Cada função de geometria possui diversos outros argumentos além de `aes` e `dados`. Esses argumentos tendem a ser específicos para cada função. Por exemplo, no gráfico que desejamos fazer, os pontos são maiores que o tamanho padrão. No arquivo de ajuda, vemos que o argumento `size` (tamanho na tradução) é uma estética e podemos alterá-lo assim:

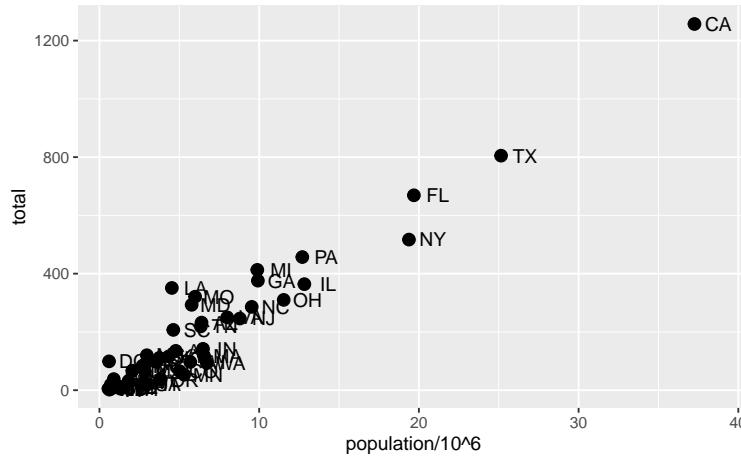
```
p + geom_point(aes(population/10^6, total), size = 3) +
  geom_text(aes(population/10^6, total, label = abb))
```



size não é um mapeamento: enquanto os mapeamentos usam dados de observações específicas e precisam estar dentro da função `aes()`, as operações que afetam todos os pontos da mesma maneira não precisam ser incluídas dentro `aes`.

Agora, como os pontos ficaram maiores, se tornou difícil ver os rótulos. Se leremos a página de ajuda para `geom_text`, vemos que o argumento `nudge_x` move o texto levemente para a direita ou para a esquerda:

```
p + geom_point(aes(population/10^6, total), size = 3) +  
  geom_text(aes(population/10^6, total, label = abb), nudge_x = 1.5)
```



Isso é preferível, pois facilita a leitura do texto. Na seção 7.11 aprenderemos uma maneira melhor de garantir que possamos ver os pontos e os rótulos.

7.6 Mapeamento estético global versus local

Na linha de código anterior, definimos o mapeamento `aes(population/10^6, total)` duas vezes, uma vez em cada geometria. Podemos evitar isso usando um mapeamento estético *global* no momento em que definimos o objeto `ggplot` em branco. Lembre-se de que a função `ggplot` contém um argumento que nos permite definir mapeamentos estéticos:

```
args(ggplot)
#> function (data = NULL, mapping = aes(), ..., environment = parent.frame())
#> NULL
```

Se definirmos um mapeamento em `ggplot`, todas as geometrias adicionadas como camadas serão atribuídas por padrão a esse mapeamento. Redefinimos `p`:

```
p <- murders %>% ggplot(aes(population/10^6, total, label = abb))
```

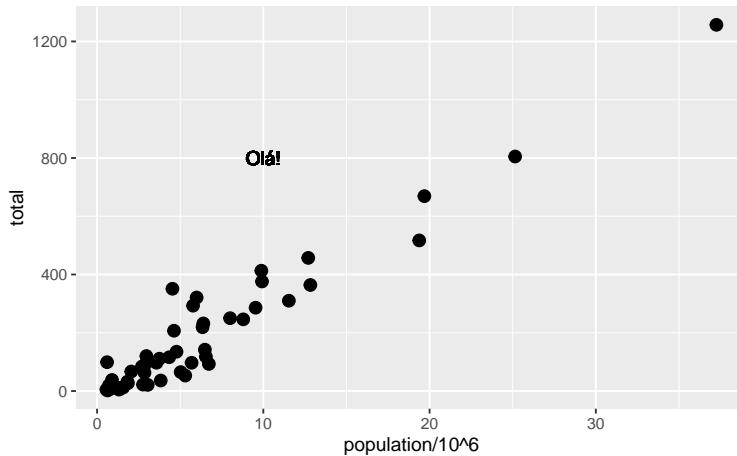
e então podemos simplesmente escrever o código a seguir para produzir o gráfico anterior:

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 1.5)
```

Mantemos os argumentos `size` e `nudge_x` em `geom_point` e `geom_text`, respectivamente, porque queremos aumentar apenas o tamanho dos pontos e ajustar a posição (`nudge` em inglês) dos rótulos. Se colocarmos esses argumentos em `aes`, eles serão aplicados aos dois gráficos. Observe também que a função `geom_point` não precisa do argumento `label` e, portanto, ignora essa estética.

Se necessário, podemos substituir o mapeamento global definindo um novo mapeamento dentro de cada camada. Essas definições a nível *local* substituem as definições de nível *global*. Aqui está um exemplo:

```
p + geom_point(size = 3) +
  geom_text(aes(x = 10, y = 800, label = "Olá!"))
```

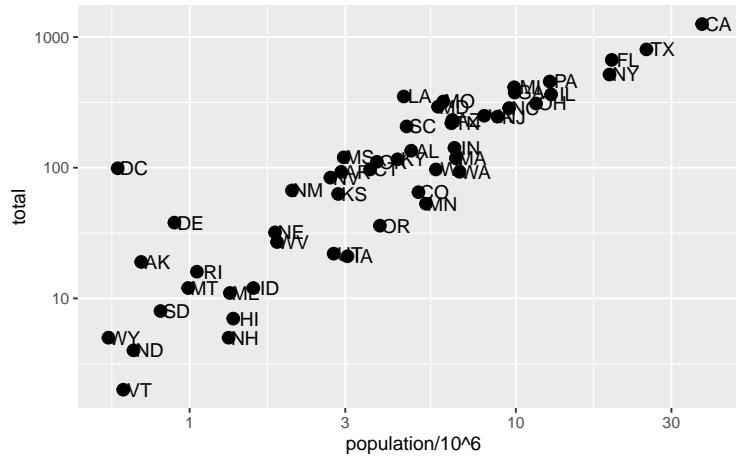


Claramente, a segunda chamada para `geom_text` não usa `population` e `total`.

7.7 Escalas

Primeiro, as escalas que queremos estão em uma escala logarítmica. Esse não é o padrão, portanto, essa alteração precisa ser adicionada por meio de uma camada `scale`. Uma rápida olhada na folha de referência revela que a função `scale_x_continuous` nos permite controlar o comportamento das escalas. Usamos assim:

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_continuous(trans = "log10") +
  scale_y_continuous(trans = "log10")
```



Como agora estamos usando a escala logarítmica, o ajuste na posição (*nudge*) deve ser menor.

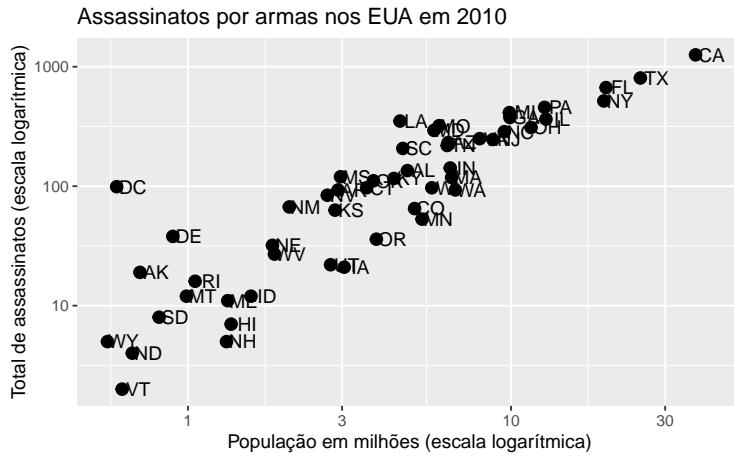
Essa transformação específica é tão comum que **ggplot2** oferece duas funções especializadas `scale_x_log10` e `scale_y_log10`, que podemos usar para reescrever o código assim:

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10()
```

7.8 Rótulos e títulos

Da mesma forma, a folha de referência nos mostra que para alterar os rótulos e adicionar um título usamos as seguintes funções:

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("População em milhões (escala logarítmica)") +
  ylab("Total de assassinatos (escala logarítmica)") +
  ggtitle("Assassinatos por armas nos EUA em 2010")
```



Estamos quase terminando! O que precisamos fazer agora é adicionar cor, legenda e realizar algumas alterações opcionais no estilo.

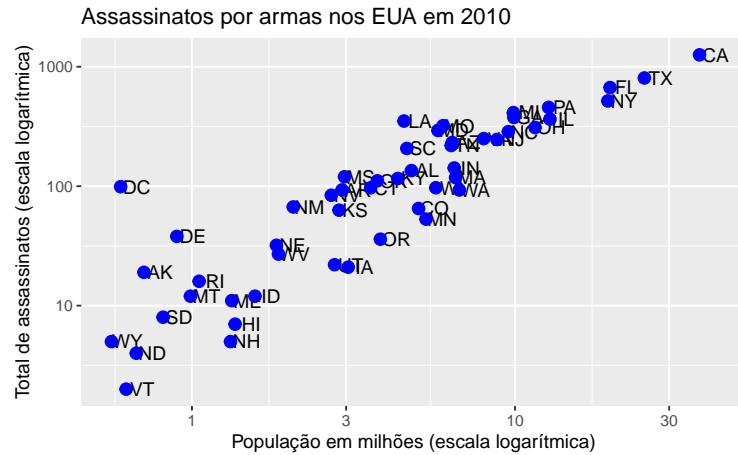
7.9 Usando categorias como cores

Podemos mudar a cor dos pontos usando o argumento `col` na função `geom_point`. Para facilitar a demonstração de novos recursos, redefiniremos `p` para ser tudo, exceto a camada de pontos:

```
p <- murders %>% ggplot(aes(population/10^6, total, label = abb)) +
  geom_text(nudge_x = 0.05) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("População em milhões (escala logarítmica)") +
  ylab("Total de assassinatos (escala logarítmica)") +
  ggtitle("Assassinatos por armas nos EUA em 2010")
```

a seguir, teste para ver o que acontece quando adicionamos chamadas diferentes a `geom_point`. Por exemplo, podemos deixar todos os pontos em azul adicionando o argumento `color`:

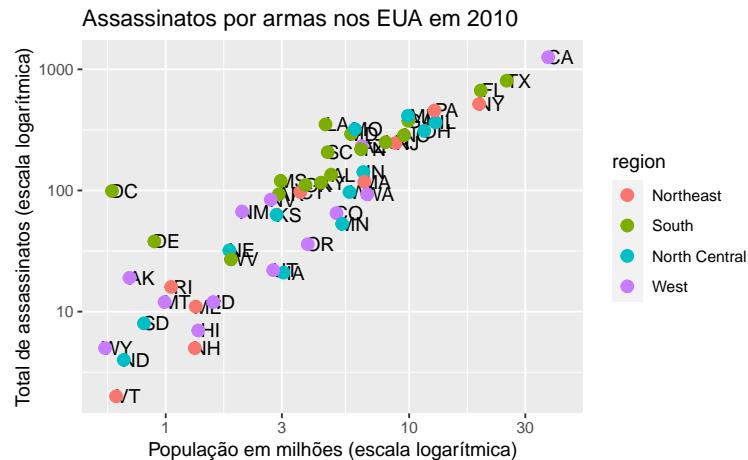
```
p + geom_point(size = 3, color = "blue")
```



Isso, é claro, não é o que queremos. Queremos atribuir cores de acordo com a região geográfica. Um bom comportamento padrão para **ggplot2** é que, se atribuirmos uma variável categórica à cor, ela atribuirá automaticamente uma cor diferente a cada categoria, além de adicionar uma legenda.

Como a escolha da cor é determinada por uma característica de cada observação, isso é considerado um mapeamento estético. Logo, para atribuir uma cor a cada ponto, precisamos usar **aes**. Usamos o seguinte código:

```
p + geom_point(aes(col=region), size = 3)
```



Os mapeamentos **x** e **y** são herdados daqueles já definidos em **p**, portanto não é necessário rededefinir-los. Nós também mudamos **aes** para o primeiro argumento, pois é aí que os mapeamentos são esperados nesta chamada de função.

Aqui está outro comportamento padrão útil: **ggplot2** adiciona automaticamente uma legenda que atribui a cor à região. Para evitar adicionar essa legenda, configuramos **geom_point** com o argumento **show.legend = FALSE**.

7.10 Anotações, formas e ajustes

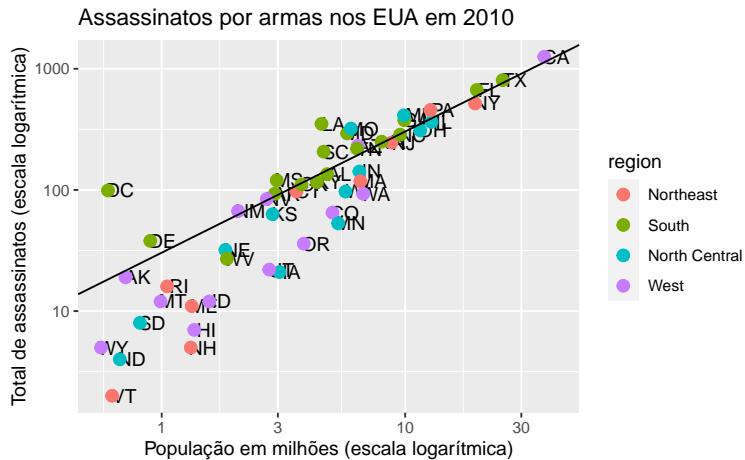
Muitas vezes, queremos adicionar formas ou anotações às figuras que não são derivadas diretamente do mapeamento estético; alguns exemplos incluem etiquetas, caixas, áreas sombreadas e linhas.

Aqui, queremos adicionar uma linha que represente a taxa média de homicídios em todo o país. Após determinar a taxa por milhão e armazená-la na variável r , a linha será definida pela fórmula: $y = rx$, sendo y e x nossos eixos: total de assassinatos e população em milhões, respectivamente. Na escala logarítmica, essa linha se torna: $\log(y) = \log(r) + \log(x)$. Então, no nosso gráfico é uma linha com a inclinação 1 e intercepta $\log(r)$. Para calcular esse valor, usamos nosso conhecimento de **dplyr**:

```
r <- murders %>%
  summarize(rate = sum(total)/sum(population) * 10^6) %>%
  pull(rate)
```

Para adicionar uma linha, usamos a função `geom_abline`. `ggplot2` usa `ab` no nome para nos lembrar de que estamos fornecendo a interceptação (`a`) e a inclinação (`b`). A linha padrão possui a inclinação 1 e intercepta 0, portanto, apenas precisamos definir a interceptação:

```
p + geom_point(aes(col=region), size = 3) +
  geom_abline(intercept = log10(r))
```



Aqui, `geom_abline` não usa nenhuma informação do objeto de dados.

Podemos alterar o tipo de linha e a cor das linhas usando argumentos. Além disso, devemos desenhar a linha primeiro para que não cubra nossos pontos.

```
p <- p + geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col=region), size = 3)
```

Observe que redefinimos `p` e vamos usar esse novo `p` abaixo e na próxima seção.

Os gráficos padrão criados por `ggplot2` já são muito úteis. No entanto, geralmente precisamos fazer pequenos ajustes no comportamento padrão. Embora nem sempre seja óbvio como fazer isso, mesmo com a folha de referência, `ggplot2` é muito flexível.

Por exemplo, podemos fazer alterações na legenda através da função `scale_color_discrete`. Em nosso gráfico original, a palavra `region` está iniciada com letra maiúscula. Podemos alterá-la assim:

```
p <- p + scale_color_discrete(name = "Region")
```

7.11 Pacotes complementares

O poder do `ggplot2` é aumentado ainda mais devido à disponibilidade de pacotes complementares (do inglês *add-on packages*). As alterações restantes necessárias para dar os retoques finais em nosso gráfico requerem os pacotes `ggthemes` e `ggrepel`.

O estilo de um gráfico `ggplot2` pode ser alterado usando as funções `theme`. Diversos temas estão incluídos como parte do pacote `ggplot2`. De fato, para a maioria dos gráficos deste livro, usamos uma função do pacote `dslabs` que define automaticamente um tema padrão:

```
ds_theme_set()
```

O pacote `ggthemes` permite adicionar muitos outros temas, incluindo o tema `theme_economist` que nós escolhemos. Após a instalação do pacote, você pode alterar o estilo adicionando uma camada da seguinte forma:

```
library(ggthemes)
p + theme_economist()
```

Você pode ver como são os outros temas simplesmente alterando a função. Por exemplo, você pode testar o tema `theme_fivethirtyeight()` no lugar do anterior.

A diferença final tem a ver com a posição dos rótulos. No nosso gráfico, alguns dos rótulos se sobrepõem. O pacote de plug-ins `ggrepel` inclui uma geometria que adiciona rótulos garantindo que eles não se sobreponham. Para usá-lo, simplesmente mudamos `geom_text` para `geom_text_repel`.

7.12 Como combinar tudo

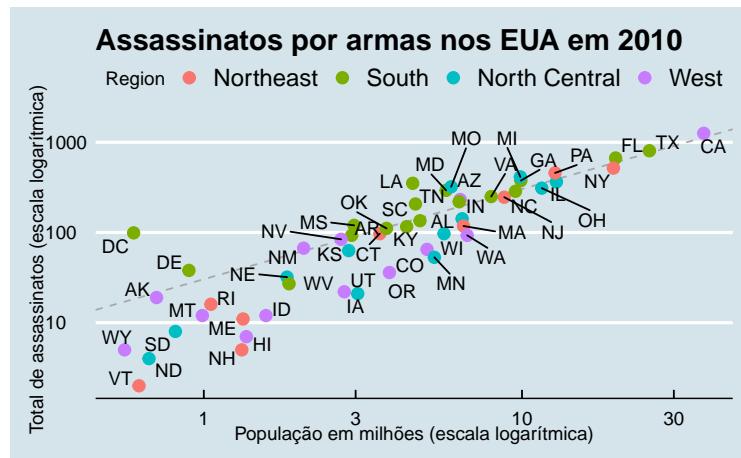
Agora que terminamos o teste, podemos escrever um código que produza o gráfico desejado do zero.

```
library(ggthemes)
library(ggrepel)

r <- murders %>%
  summarize(rate = sum(total) / sum(population) * 10^6) %>%
  pull(rate)

murders %>% ggplot(aes(population/10^6, total, label = abb)) +
  geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
  geom_point(aes(col=region), size = 3) +
```

```
geom_text_repel() +
scale_x_log10() +
scale_y_log10() +
xlab("População em milhões (escala logarítmica)") +
ylab("Total de assassinatos (escala logarítmica)") +
ggtitle("Assassinatos por armas nos EUA em 2010") +
scale_color_discrete(name = "Region") +
theme_economist()
```



7.13 Gráficos rápidos com qplot

Nós aprendemos poderosas técnicas para gerar visualizações com ggplot. No entanto, há casos em que precisamos apenas de um gráfico rápido, como por exemplo, um histograma dos valores em um vetor, um gráfico de dispersão dos valores em dois vetores ou um *boxplot* usando vetores categóricos e numéricos. Já demonstramos como gerar esses gráficos com *hist*, *plot* e *boxplot*. Entretanto, se queremos ser consistentes com o estilo ggplot, podemos usar a função *qplot*.

Se tivermos valores em dois vetores como:

```
data(murders)
x <- log10(murders$population)
y <- murders$total
```

e queremos fazer um diagrama de dispersão com ggplot, teríamos que escrever algo como:

```
data.frame(x = x, y = y) %>%
ggplot(aes(x, y)) +
geom_point()
```

Parece ser muito código para um gráfico tão simples. A função *qplot* sacrifica a flexibilidade oferecida pelo *ggplot*, mas permite gerar rapidamente um gráfico.

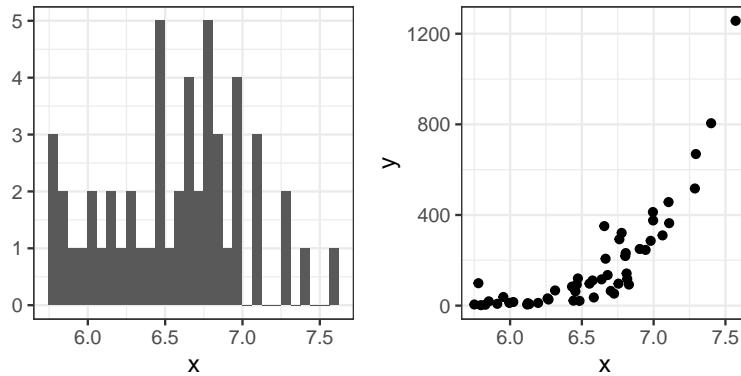
```
qplot(x, y)
```

Vamos aprender mais sobre `qplot` na seção 8.16

7.14 Grade de gráficos

Muitas vezes, temos que colocar gráficos próximos um do outro. O pacote `gridExtra` nos permite fazer isso:

```
library(gridExtra)
p1 <- qplot(x)
p2 <- qplot(x,y)
grid.arrange(p1, p2, ncol = 2)
```



7.15 Exercícios

Comece carregando os pacotes `dplyr` e `ggplot2`, bem como os dados `murders` e `heights`.

```
library(dplyr)
library(ggplot2)
library(dslabs)
data(heights)
data(murders)
```

1. Com `ggplot2`, os gráficos podem ser salvos como objetos. Por exemplo, podemos associar um conjunto de dados a um objeto gráfico da seguinte maneira:

```
p <- ggplot(data = murders)
```

Como `data` é o primeiro argumento, não precisamos declará-lo:

```
p <- ggplot(murders)
```

ou também podemos usar o *pipe*:

```
p <- murders %>% ggplot()
```

Qual é a classe do objeto `p`?

2. Lembre-se de que para imprimir um objeto, você pode usar o comando `print` ou apenas escrever o nome do objeto. Imprima o objeto `p` definido no exercício 1 e descreva o que você vê.

- a. Nada acontece.
- b. Um gráfico em branco.
- c. Um diagrama de dispersão.
- d. Um histograma.

3. Usando o `pipe %>%`, crie um objeto `p` desta vez associado ao conjunto de dados `heights` em vez do conjunto de dados `murders`.

4. Qual é a classe do objeto `p` que você acabou de criar?

5. Agora vamos adicionar uma camada e os mapeamentos estéticos correspondentes. Para os dados de assassinatos, plote o total de assassinatos versus o tamanho da população. Explore o conjunto de dados `murders` para lembrar quais são os nomes dessas duas variáveis e escolha a resposta correta. **Dica:** use `?murders`.

- a. `state` e `abb`
- b. `total_murders` e `population_size`
- c. `total` e `population`
- d. `murders` e `size`

6. Para criar um gráfico de dispersão, adicionamos uma camada com `geom_point`. O mapeamento estético requer que definamos as variáveis dos eixos `x` e `y`, respectivamente. Portanto, o código fica assim:

```
murders %>% ggplot(aes(x = , y = )) +
  geom_point()
```

exceto que temos que definir as duas variáveis `x` e `y`. Preencha o espaço com os nomes corretos das variáveis.

7. Note que, se não usarmos nomes de argumentos, podemos obter o mesmo gráfico se inserirmos os nomes das variáveis na ordem correta, assim:

```
murders %>% ggplot(aes(population, total)) +
  geom_point()
```

Refaça o gráfico, mas agora com total de assassinatos no eixo `x` e o tamanho da população no eixo `y`.

8. Se, em vez de pontos quisermos adicionar textos, podemos usar as geometrias `geom_text()` ou `geom_label()`. O código a seguir:

```
murders %>% ggplot(aes(population, total)) + geom_label()
```

nos dará a mensagem de erro: `Error: geom_label requires the following missing aesthetics: label` (ou na tradução `Erro: geom_label requer a seguinte estética ausente: label`)

Por que isso acontece?

- a. Precisamos mapear caracteres para cada ponto através do argumento `label` em `aes`.
- b. Precisamos deixar `geom_label` saber quais caracteres usar no gráfico.

- c. A geometria `geom_label` não requer valores dos eixos x e y.
d. `geom_label` não é um comando de `ggplot2`.
9. Reescreva o código acima para usar a abreviação como o rótulo através da função `aes`.
10. Mude a cor dos rótulos para azul. Como faremos isso?
- Adicionando uma coluna chamada `blue` em `murders`.
 - Como cada etiqueta precisa de uma cor diferente, mapeamos as cores através `aes`.
 - Usando o argumento `color` no `ggplot`.
 - Como queremos que todas as cores sejam azuis, não precisamos atribuir cores, basta usar o argumento de cores em `geom_label`.
11. Reescreva o código acima para que os rótulos fiquem azuis.
12. Agora, suponha que desejemos usar cores para representar as diferentes regiões. Nesse caso, qual das seguintes opções é a mais apropriada?
- Adicione uma coluna chamada `color` em `murders` com a cor que queremos usar.
 - Como cada tag precisa de uma cor diferente, mapeie as cores através do argumento de cores da função `aes`.
 - Use o argumento `color` do `ggplot`.
 - Como queremos que todas as cores sejam azuis, não precisamos atribuir cores, basta usar o argumento de cores em `geom_label`.
13. Reescreva o código acima para que a cor dos rótulos seja determinada pela região do estado.
14. Agora vamos mudar o eixo x para uma escala logarítmica para levar em conta o fato de que a distribuição da população é assimétrica. Vamos começar definindo um objeto `p` salvando o gráfico que criamos até agora:
- ```
p <- murders %>%
 ggplot(aes(population, total, label = abb, color = region)) +
 geom_label()
```
- Para alterar o eixo y para uma escala logarítmica, aprendemos sobre a função `scale_x_log10()`. Adicione esta camada ao objeto `p` para alterar a escala e criar o gráfico.
15. Repita o exercício anterior, mas agora mude os dois eixos para que estejam na escala logarítmica.
16. Agora edite o código acima para adicionar o título “Dados de assassinatos por armas” ao argumento. Dica: use a função `ggtitle`.



# 8

---

## *Visualizando distribuições de dados*

---

Os dados numéricos são frequentemente resumidos com o valor da média. Por exemplo, a qualidade de uma escola secundária às vezes é resumida em um único número: a pontuação média em um teste padronizado. Ocasionalmente, um segundo número é incluído: o desvio padrão. Por exemplo, você pode ler um relatório afirmando que as pontuações foram 680 mais ou menos 50 (o desvio padrão). O relatório resumiu um vetor completo de pontuações com apenas dois números. Isso é apropriado? Existe alguma informação importante que não estamos considerando ao exibir esse resumo em vez da lista completa?

O primeiro componente básico de visualização de dados que aprenderemos será como resumir listas de fatores ou vetores de números. Na maioria das vezes, a melhor maneira de compartilhar ou explorar esses resumos é através da visualização de dados. O resumo estatístico mais básico de uma lista de objetos ou números é sua distribuição. Depois que um vetor é resumido como uma distribuição, existem várias técnicas de visualização de dados para transmitir efetivamente essas informações.

Neste capítulo, discutiremos primeiro as propriedades de uma variedade de distribuições e como visualizá-las usando como estudo de caso uma base de dados com altura de alunos. Então, na Seção 8.16, discutiremos as geometrias **ggplot2** para essas visualizações.

---

### 8.1 Tipos de variáveis

Trabalharemos com dois tipos de variáveis: categórica e numérica. Cada uma pode ser dividida em dois outros grupos: as variáveis categóricas podem ser ordinais ou não, enquanto as numéricas podem ser discretas ou contínuas.

Quando cada entrada em um vetor vem de um pequeno número de grupos, nos referimos aos dados como *dados categóricos*. Dois exemplos simples são sexo (masculino ou feminino) e regiões (nordeste, sul, norte central, oeste). Alguns dados categóricos podem ser solicitados, mesmo que não sejam números, por exemplo, quão picante é um alimento (pouco, médio, muito). Nos livros de estatística, os dados categóricos ordenados são referidos como *dados ordinais*.

Exemplos de dados numéricos são tamanho da população, taxas de homicídios e altura. Alguns dados numéricos podem ser tratados como categóricos ordenados. Podemos dividir ainda mais os dados numéricos em contínuos e discretos. Variáveis contínuas são aquelas que podem assumir qualquer valor, como alturas, se forem medidas com precisão suficiente. Por exemplo, um par de botões de punho pode medir 68,12 e 68,11 polegadas, respectivamente. As contagens, como o tamanho da população, são discretas porque precisam ser números redondos.

Observe que dados numéricos discretos podem ser considerados ordinais. Embora isso seja

tecnicamente verdadeiro, geralmente reservamos o termo dados ordinais para variáveis que pertencem a um pequeno número de grupos diferentes, com cada grupo tendo muitos membros. Por outro lado, quando temos muitos grupos com poucos casos em cada grupo, geralmente nos referimos a eles como variáveis numéricas discretas. Assim, por exemplo, o número de maços de cigarro que uma pessoa fuma por dia, arredondado para o maço mais próximo, seria considerado ordinal, enquanto o número real de cigarros seria considerado uma variável numérica. No entanto, existem exemplos que podem ser considerados numéricos e ordinais quando se trata de visualizar dados.

## 8.2 Estudo de caso: descrevendo a altura dos alunos

Aqui, apresentamos um novo problema motivador. É um exemplo artificial, mas nos ajudará a ilustrar os conceitos necessários para entender as distribuições.

Imagine que precisamos descrever as alturas de nossos colegas de classe para um ET, um alienígena que nunca viu seres humanos. Como primeiro passo, precisamos coletar dados. Para fazer isso, pedimos aos alunos que indiquem suas alturas em polegadas. Pedimos ainda que forneçam informações sobre o seu sexo biológico, porque sabemos que existem duas distribuições diferentes por sexo. Coletamos os dados e salvamos no *data frame heights*:

```
library(tidyverse)
library(dslabs)
data(heights)
```

Uma maneira de passar as alturas para o ET é simplesmente enviar a ele esta lista de 1,050 alturas. Entretanto, existem maneiras muito mais eficazes de transmitir essas informações, e entender o conceito de uma distribuição ajudará. Para simplificar a explicação, primeiro nos concentraremos nas alturas masculinas. Examinaremos os dados de altura feminina na Seção 8.14.

## 8.3 Função de distribuição

Acontece que, em alguns casos, a média e o desvio padrão são praticamente tudo que precisamos para entender os dados. Aprenderemos técnicas de visualização de dados que nos ajudarão a determinar quando esse resumo de dois números é apropriado. Essas mesmas técnicas servirão como uma alternativa para quando esses dois números não forem suficientes.

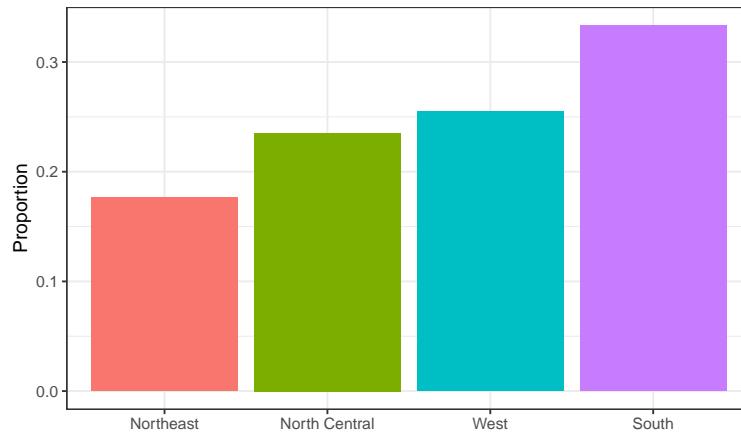
O resumo estatístico mais básico de uma lista de objetos ou números é sua distribuição. A maneira mais simples de pensar em uma distribuição é como uma descrição compacta de uma lista com muitas entradas. Este conceito não deve ser novo para os leitores deste livro. Por exemplo, com dados categóricos, a distribuição simplesmente descreve a proporção de cada categoria exclusiva. Para o sexo representado no conjunto de dados de altura seria:

```
#>
#> Female Male
#> 0.227 0.773
```

Essa tabela de frequência de duas categorias é a forma mais simples de uma distribuição. Nós

não precisamos realmente visualizar isso, pois um número descreve tudo o que precisamos saber: 23% são mulheres e os restantes são homens. Quando há mais categorias, um gráfico de barras simples pode descrever a distribuição. Aqui está um exemplo de distribuição dos estados dos EUA por região:

```
#> `summarise()` ungrouping output (override with ` .groups` argument)
```



Esse gráfico simplesmente mostra quatro números, um para cada categoria. Geralmente usamos gráficos de barras quando temos poucos números. Embora esse gráfico em particular não forneça muito mais informações do que uma tabela de frequência, ele é um excelente exemplo de como convertemos um vetor em um gráfico que resume sucintamente todas as informações. Quando os dados são numéricos, a tarefa de exibir distribuições é mais desafiadora.

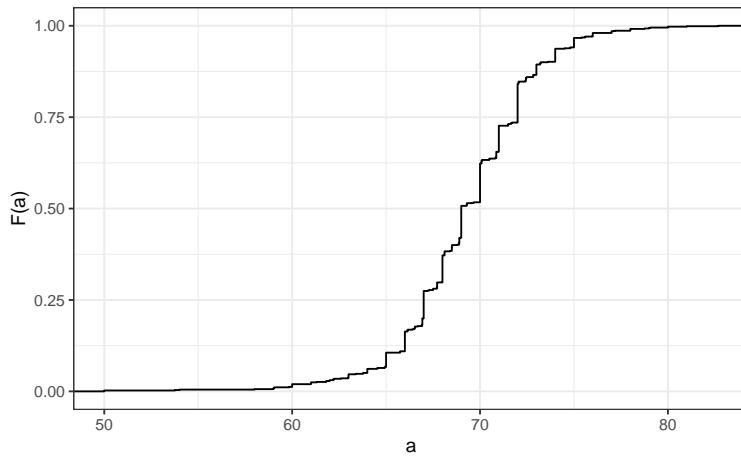
## 8.4 Funções de distribuição acumulada

Dados numéricos que não são categóricos também têm distribuições. Em geral, quando os dados não são categóricos, a indicação da frequência de cada entrada não é um resumo efetivo, pois a maioria das entradas são únicas. Por exemplo, em nosso estudo de caso, enquanto vários estudantes relataram uma altura de 68 polegadas, dois estudantes relataram alturas de 68.503937007874 e 68.8976377952756 polegadas. Acreditamos que eles tenham convertido suas alturas para polegadas a partir dos valores 174 e 175 centímetros, respectivamente.

Os livros de estatística nos ensinam que uma maneira mais útil de definir uma distribuição numérica de dados é definir uma função que indica a proporção dos dados abaixo de  $a$  para todos os valores possíveis de  $a$ . Essa função é chamada de função de distribuição acumulada (FDA). Em estatística, a seguinte notação é usada:

$$F(a) = \Pr(x \leq a)$$

Aqui vemos um gráfico de  $F$  para dados de altura de estudantes do sexo masculino:



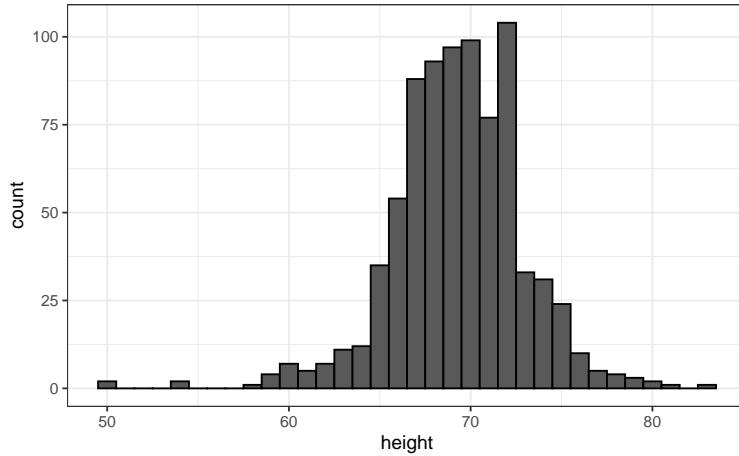
Semelhante ao que a tabela de frequência faz para dados categóricos, o FDA define a distribuição de dados numéricos. No gráfico, podemos ver que 16% dos valores são menores que 65, pois  $F(66) = 0.164$ , ou que 84% dos valores são menores que 72, pois  $F(72) = 0.841$  e assim por diante. De fato, podemos relatar a proporção de valores entre duas alturas, denominadas aqui como  $a$  e  $b$ , ao computar  $F(b) - F(a)$ . Isso significa que, se enviarmos esse diagrama para o ET, ele terá todas as informações necessárias para reconstruir a lista completa. Parafraseando a expressão “uma imagem vale mais que mil palavras”, neste caso, uma imagem é tão informativa quanto 812 números.

Uma observação final: como os FDAs podem ser matematicamente definidos, a palavra *empírico* é adicionada para distinguir quando os dados são usados. Portanto, usamos o termo FDA empírico (FDAE).

## 8.5 Histogramas

Embora o conceito de FDA seja amplamente discutido nos livros estatísticos, o gráfico não é muito popular na prática. O principal motivo é que ele não transmite facilmente características de interesse, como: em que valor a distribuição se concentra? A distribuição é simétrica? Quais intervalos contêm 95% dos valores? Os histogramas são preferidos porque facilitam responder a essas perguntas. Os histogramas sacrificam apenas um pouco de informação para produzir gráficos muito mais fáceis de interpretar.

A maneira mais fácil de criar um histograma é dividir a distribuição dos nossos dados em compartimentos de mesmo tamanho que não se sobreponham. Então, para cada compartimento, contamos o número de valores que estão nesse intervalo. O histograma representa graficamente essas contagens como barras em que a base é definida pelos intervalos. Aqui está o histograma para os dados de altura separados pelos intervalos de valores em polegadas: [49.5, 50.5], [51.5, 52.5], [53.5, 54.5], ..., [82.5, 83.5]



Como você pode ver na figura acima, um histograma é semelhante a um gráfico de barras, mas se diferencia pelo fato de o eixo x ser numérico e não categórico.

Se enviarmos esse gráfico para o ET, ele aprenderá imediatamente alguns detalhes importantes sobre nossos dados. Primeiro, o intervalo de dados varia de 50 a 84, com a maioria (acima de 95%) entre 63 e 75 polegadas. Em segundo lugar, as alturas são quase simétricas ficando em torno de 69 polegadas. Além disso, adicionando contagens, o ET poderia obter uma aproximação muito boa da proporção dos dados em qualquer intervalo. Portanto, o histograma acima não é apenas mais fácil de interpretar, mas também oferece quase todas as informações contidas na lista com 812 alturas em menos de 30 barras.

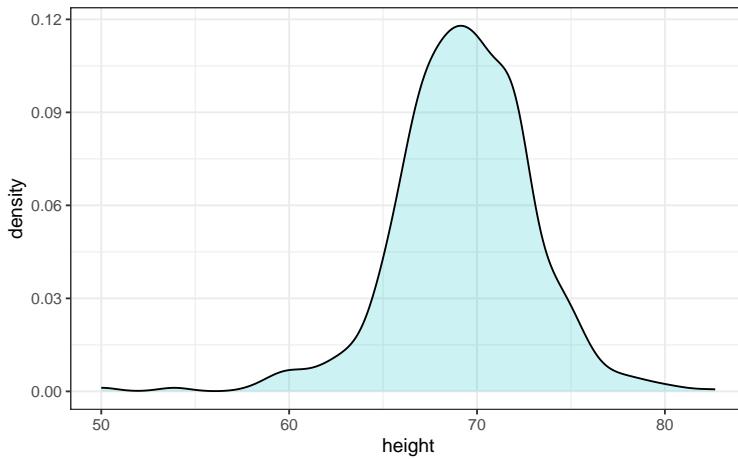
Que informações perdemos? Note que todos os valores em cada intervalo são tratados da mesma maneira ao calcular as alturas das barras. Por exemplo, o histograma não distingue entre 64, 64,1 e 64,2 polegadas. Como essas diferenças são quase imperceptíveis aos olhos, as implicações práticas são desprezíveis e conseguimos resumir os dados em apenas 23 números.

Discutimos como codificar histogramas na Seção 8.16.

---

## 8.6 Curvas de densidade

Os gráficos de densidade, também conhecidos como curvas de densidade suavizadas (*smooth density*), são esteticamente mais atraentes que os histogramas. Abaixo, vemos um gráfico de densidade para nossos dados de altura:

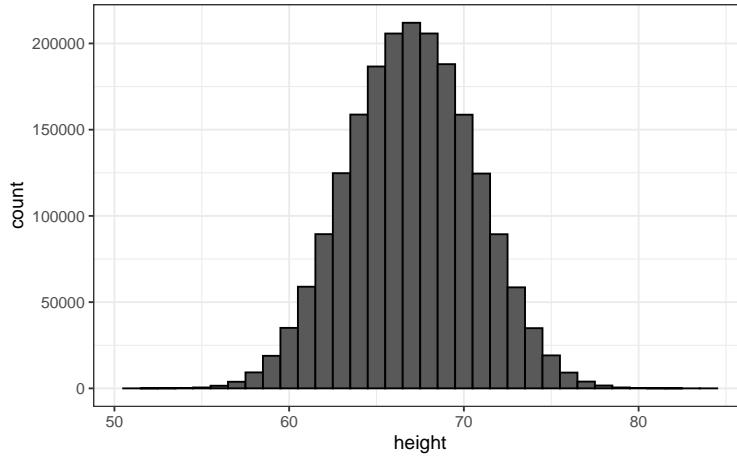


Nesse gráfico, não temos mais extremidades afiadas nos limites de intervalo e muitos dos picos locais foram removidos. Além disso, a escala do eixo y mudou de contagens para *densidade*.

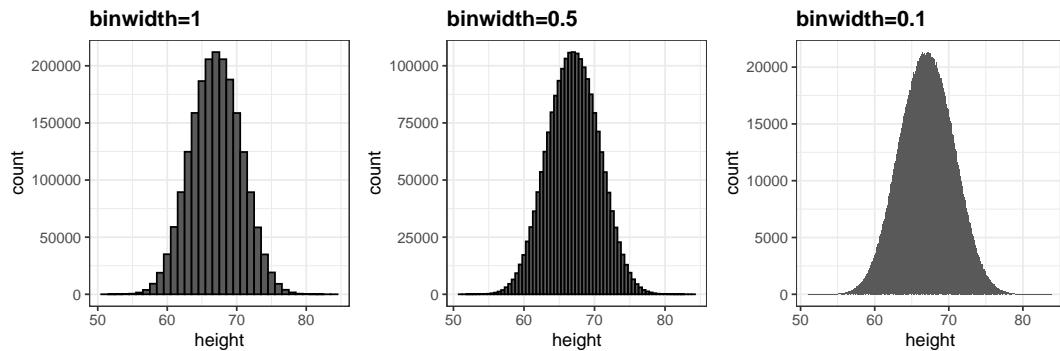
Para entender o gráfico de densidade, precisamos entender sobre *estimativas*, um tópico que não abordaremos até mais tarde. No entanto, ofereceremos aqui uma explicação heurística para ajudá-lo a entender o básico, e assim, você possa usar essa útil ferramenta na visualização de dados.

O principal novo conceito que você precisa entender é que assumimos que nossa lista de valores observados é um subconjunto de uma lista muito maior de valores não observados. No caso das alturas, você pode imaginar que nossa lista com 812 estudantes do sexo masculino se origina de uma lista hipotética que contém as alturas de todos os estudantes do mundo, medidas com grande precisão. Digamos que existam 1.000.000 dessas medidas. Essa lista de valores tem uma distribuição, como qualquer lista de valores. Essa distribuição considerável é o que queremos realmente reportar ao ET, pois é muito mais generalizada. Infelizmente, não conseguimos ver isso.

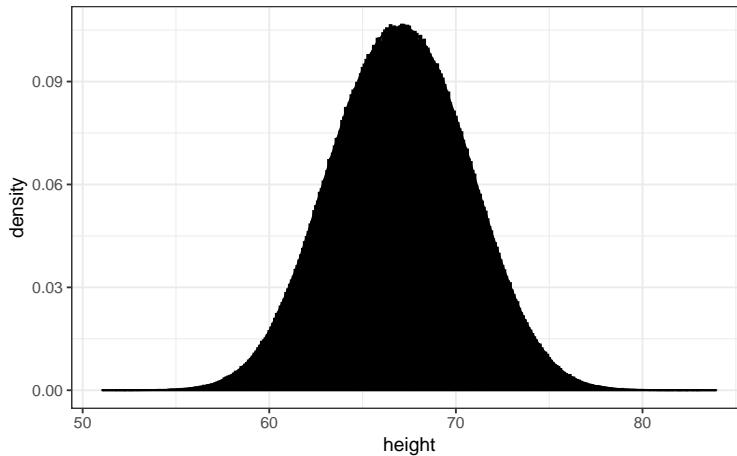
Entretanto, podemos fazer uma suposição que pode nos ajudar a obter um resultado aproximado disso. Se tivéssemos 1.000.000 valores, medidos com muita precisão, poderíamos fazer um histograma com barras de compartimentos muito, muito pequenas. A suposição é que, se mostrarmos isso, a altura das barras consecutivas será semelhante. Isso é o que queremos dizer com densidade suave (*smooth density*): não temos grandes saltos nas alturas de barras consecutivas. Aqui está um histograma hipotético com barras de tamanho 1:



Quanto menor o intervalo do compartimento das barras, mais suave o histograma se torna. Aqui estão os histogramas com larguras de compartimento de 1, 0,5 e 0,1:

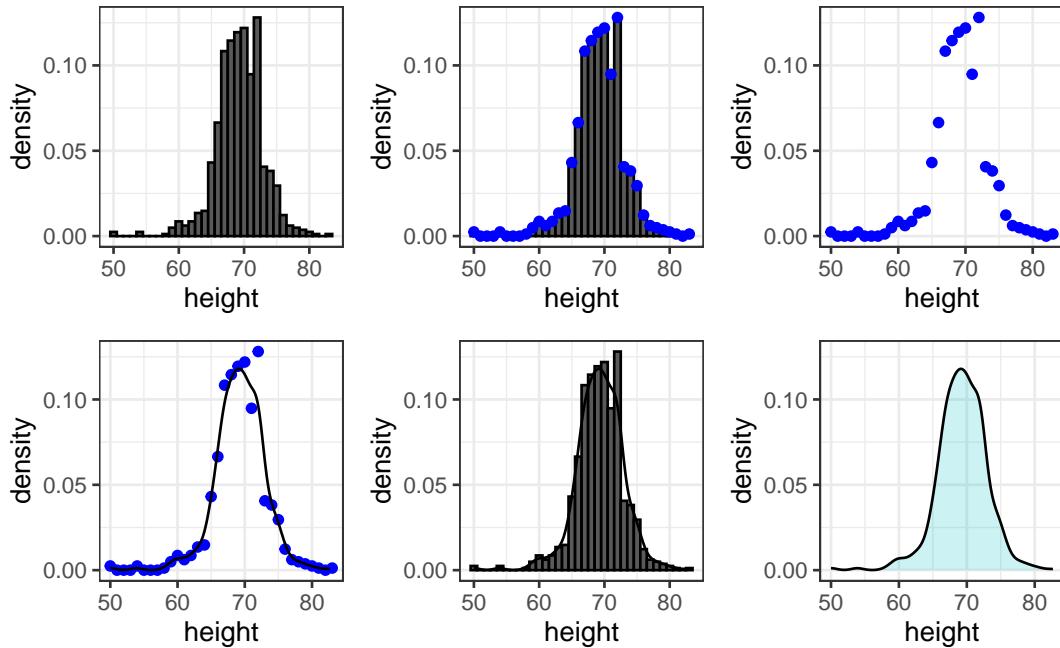


A densidade suave é basicamente a curva que atravessa a parte superior das barras do histograma quando os compartimentos são muito, muito pequenos. Para que a curva não dependa do tamanho hipotético da lista hipotética, calculamos a curva usando frequências em vez de contagens:

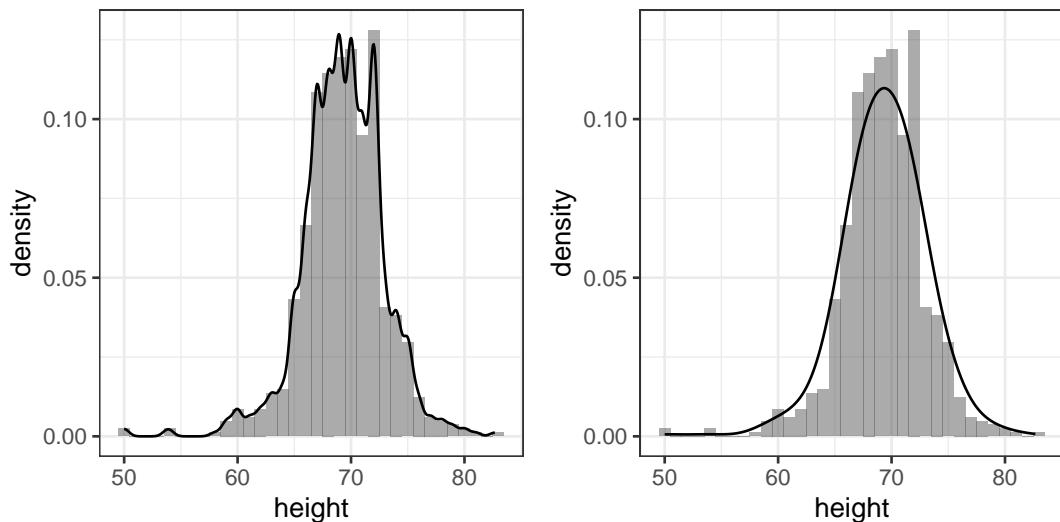


Agora, voltamos à realidade. Não temos milhões de medições. Em vez disso, temos 812 e não podemos fazer um histograma com compartimentos muito pequenos.

Portanto, fazemos um histograma usando tamanhos de compartimento apropriados para nossos dados e calculamos as frequências em vez de contagens. Além disso, desenhamos uma curva suave que passa pelo topo das barras do histograma. Os gráficos a seguir mostram as etapas que levam à construção das curvas de densidade suavizadas:



No entanto, lembre-se de que *suave* é um termo relativo. De fato, podemos controlar a suavidade da curva alterando o número de pontos nos compartimentos. Aqui estão dois exemplos que usam diferentes níveis de suavidade no mesmo histograma:



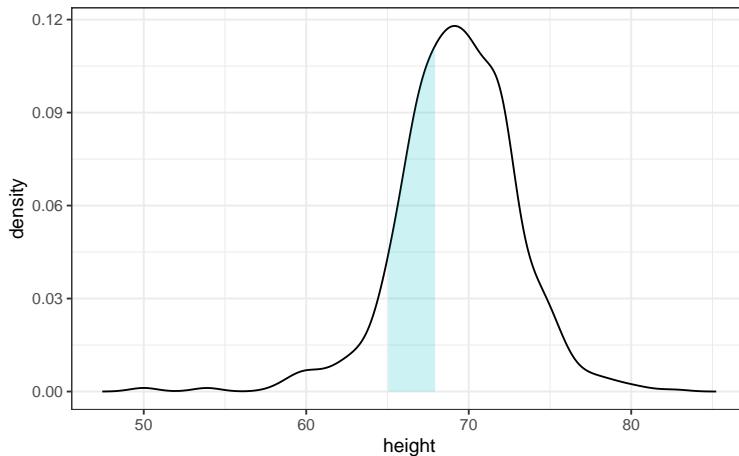
Precisamos tomar essa decisão com cuidado, pois as visualizações resultantes podem alterar nossa interpretação dos dados. Devemos selecionar um grau de suavidade que sejam representativos para os dados subjacentes. No caso da altura, realmente temos motivos para

acreditar que a proporção de pessoas com alturas semelhantes deve ser a mesma. Por exemplo, a proporção de estudantes com 72 polegadas de altura deve ser mais próxima à proporção de estudantes com 71 do que com 78 ou 65 polegadas. Isso implica que a curva deve ser razoavelmente suave; isto é, a curva deve se parecer mais com o exemplo à direita do que com a esquerda.

Embora o histograma seja um resumo sem suposições, a densidade suavizada é baseada em algumas suposições.

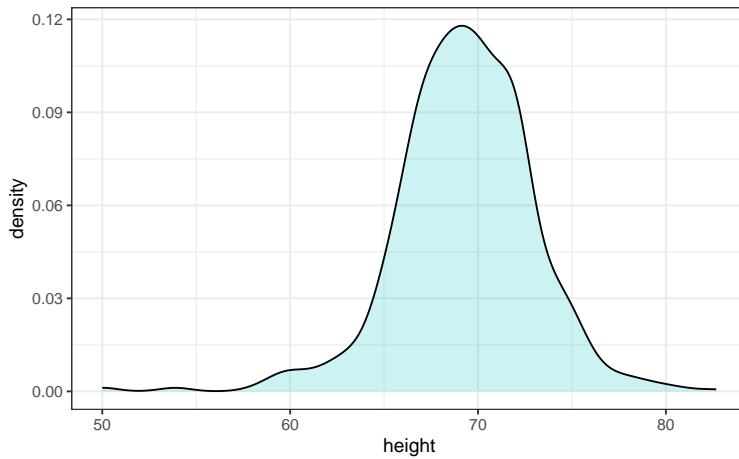
### 8.6.1 Interpretando o eixo y

Observe que a interpretação do eixo y de um gráfico de densidade suave não é óbvia. Ele é dimensionado para que a área sob a curva de densidade seja igual a 1. Se você imaginar que formamos um compartimento com uma base de 1 unidade de comprimento, o valor do eixo y indica a proporção de valores nesse compartimento. No entanto, isso é válido apenas para compartimentos de tamanho 1. Para intervalos de outro tamanho, a melhor maneira de determinar a proporção de dados nesse intervalo é calculando a proporção da área total contida no intervalo. Por exemplo, aqui vemos a proporção de valores entre 65 e 68:



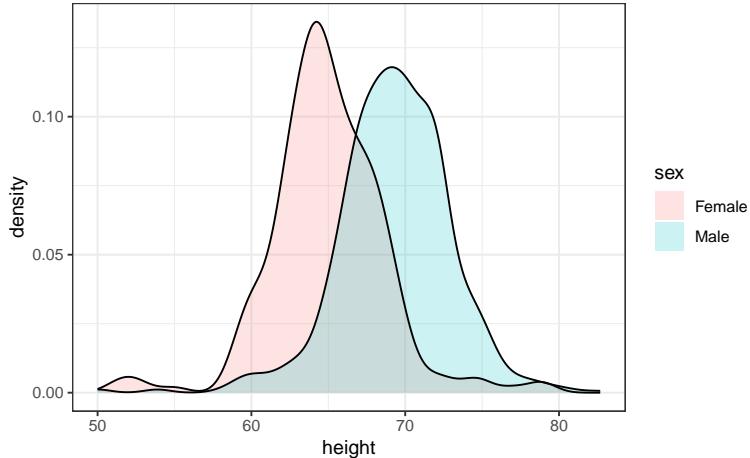
A proporção dessa área é de aproximadamente 0.3, o que significa que aproximadamente NaN% das alturas dos estudantes do sexo masculino estão entre 65 e 68 polegadas.

Ao entender isso, estamos prontos para usar as curvas de densidade como uma forma de sumarizar. Para esse conjunto de dados, nos sentimos confortáveis em assumir suavidade e, assim, compartilhar essa figura esteticamente agradável com o ET, que pode usá-la para entender nossos dados de altura masculina:



### 8.6.2 Densidades permitem a estratificação

Como nota final, observamos que uma vantagem das densidades suaves sobre os histogramas para propósitos de visualização é que as densidades facilitam a comparação entre duas distribuições. Isso se deve em grande parte às bordas irregulares do histograma, que geram confusão. Aqui está um exemplo comparando as alturas masculina e feminina:

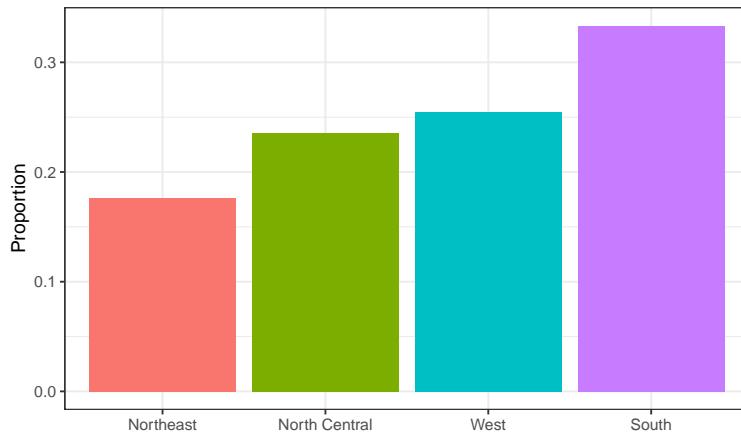


Com o argumento correto, `ggplot` sombreia automaticamente a região de interseção com uma cor diferente. Mostraremos exemplos de códigos para densidades do `ggplot2` na Seção 9 bem como na Seção 8.16.

## 8.7 Exercícios

1. No conjunto de dados `murders` (assassinatos), a região é uma variável categórica e sua distribuição é a seguinte:

```
#> `summarise()` ungrouping output (override with ` .groups` argument)
```

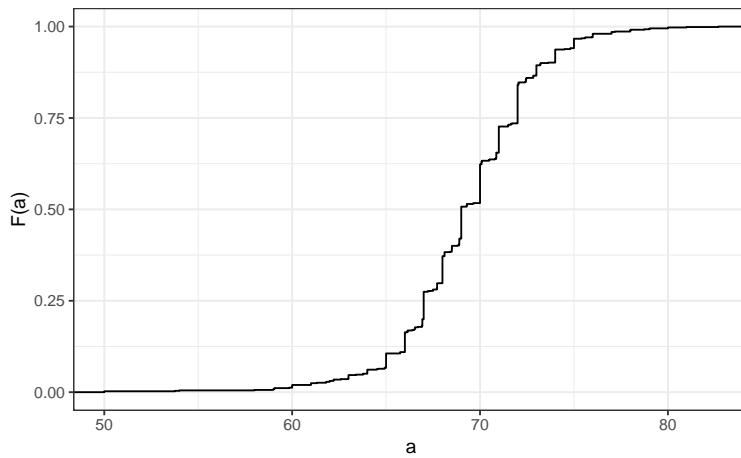


Arredondando para os 5% mais próximos, qual a proporção de estados na região “Centro-Norte” (*North Central*)?

2. Qual das seguintes opções é verdadeira?

- a. O gráfico acima é um histograma.
- b. O gráfico acima mostra apenas quatro números com um gráfico de barras.
- c. As categorias não são números, portanto, não faz sentido representar graficamente a distribuição.
- d. As cores, e não a altura das barras, descrevem a distribuição.

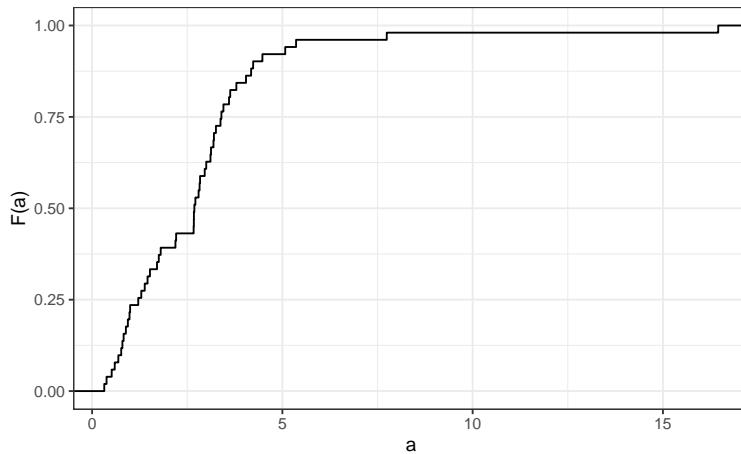
3. O gráfico a seguir mostra o FDAE para a altura dos homens:



De acordo com o gráfico, qual a porcentagem de homens com menos de 75 polegadas?

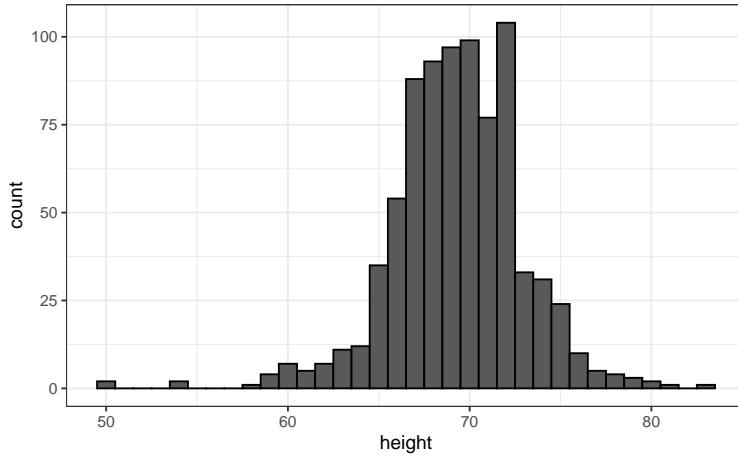
- a. 100%
  - b. 95%
  - c. 80%
  - d. 72 polegadas
4. Considerando o valor em polegadas mais próximo, qual a altura  $m$  que separa a metade dos estudantes do sexo masculino maiores da outra metade?

- a. 61 polegadas  
 b. 64 polegadas  
 c. 69 polegadas  
 d. 74 polegadas
5. Aqui está um FDAE das taxas de homicídio entre os estados:



Sabendo que existem 51 estados (contando DC) e com base nesse gráfico, quantos estados têm taxas de homicídios superiores a 10 por 100.000 pessoas?

- a. 1  
 b. 5  
 c. 10  
 d. 50
6. De acordo com o FDAE anterior, qual das seguintes afirmações é verdadeira?
- a. Cerca de metade dos estados têm taxas de homicídio acima de 7 por 100.000 e a outra metade abaixo.  
 b. A maioria dos estados tem taxas de homicídio inferiores a 2 por 100.000.  
 c. Todos os estados têm taxas de homicídio superiores a 2 por 100.000.  
 d. Com exceção de 4 estados, as taxas de homicídio são inferiores a 5 por 100.000.
7. Abaixo é apresentado o histograma das alturas masculinas do conjunto de dados `heights`:



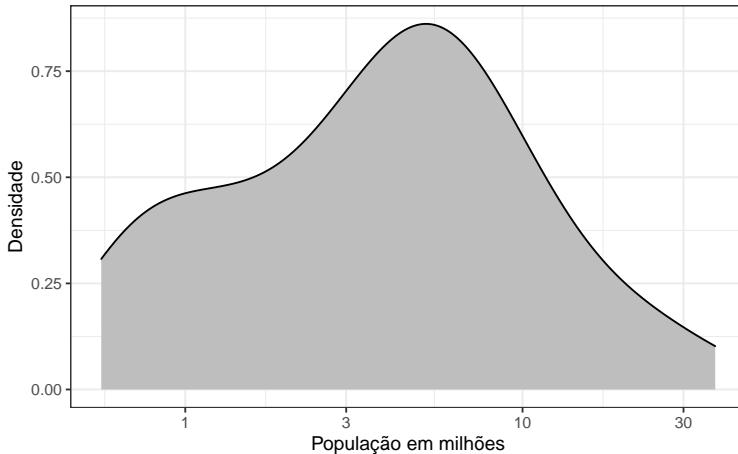
De acordo com esse gráfico, quantos homens possuem entre 63,5 e 65,5 polegadas de altura?

- a. 10
- b. 24
- c. 34
- d. 100

8. Aproximadamente, qual **porcentagem** tem menos de 60 polegadas?

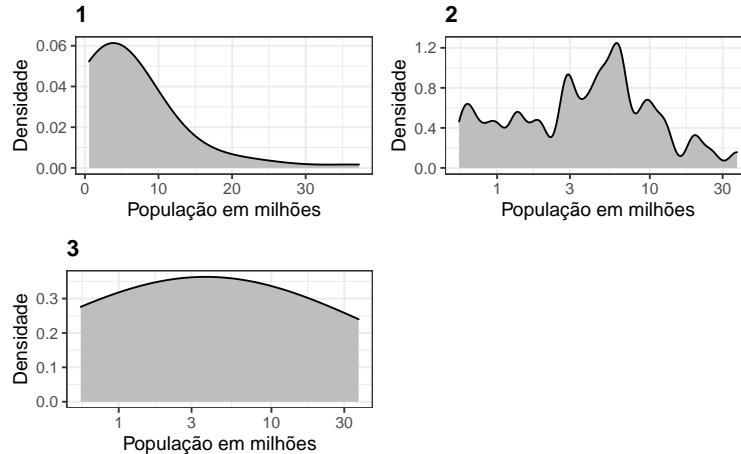
- a. 1%
- b. 10%
- c. 25%
- d. 50%

9. Com base no gráfico de densidade abaixo, qual a proporção aproximada de estados dos EUA tem populações com mais de 10 milhões de habitantes?



- a. 0,02
- b. 0,15
- c. 0,50
- d. 0,55

10. Abaixo estão três gráficos de densidade. É possível que eles sejam do mesmo conjunto de dados?



Qual das seguintes afirmações é verdadeira?

- a. É impossível que eles sejam do mesmo conjunto de dados.
- b. Eles são do mesmo conjunto de dados, mas os gráficos são diferentes devido a erros no código.
- c. Eles são do mesmo conjunto de dados, mas o primeiro e o segundo gráficos aplicam uma baixa suavização, enquanto o terceiro gráfico suaviza demais.
- d. Eles são do mesmo conjunto de dados, mas o primeiro não está na escala logarítmica, o segundo suaviza pouco e o terceiro suaviza demais.

## 8.8 Distribuição normal

Histogramas e gráficos de densidade fornecem excelentes resumos de uma distribuição. Mas podemos resumi-los melhor? Muitas vezes vemos a média e o desvio padrão usados como um resumo estatístico: um resumo de dois números! Para entender o que são esses resumos e por que são tão usados, precisamos entender a distribuição normal.

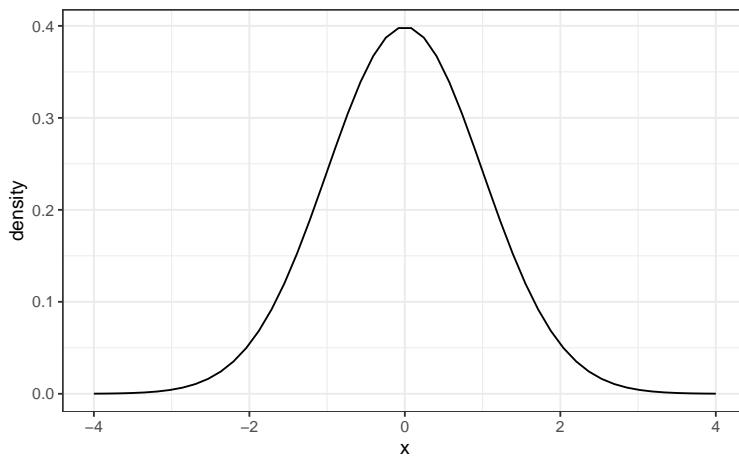
A distribuição normal, também conhecida como curva de sino ou como distribuição Gaussiana, é um dos conceitos matemáticos mais famosos da história. Uma razão para isso é que distribuições aproximadamente normais ocorrem em muitas situações, incluindo em jogos de azar, alturas, pesos, pressão arterial, resultados de testes padronizados e erros de medição experimental. Existem explicações para isso e as descreveremos posteriormente. Aqui, focamos em como a distribuição normal nos ajuda a resumir os dados.

Em vez de usar dados, a distribuição normal é definida com uma fórmula matemática. Para qualquer intervalo  $(a, b)$ , a proporção de valores nesse intervalo pode ser calculada usando esta fórmula:

$$\Pr(a < x < b) = \int_a^b \frac{1}{\sqrt{2\pi}s} e^{-\frac{1}{2}\left(\frac{x-m}{s}\right)^2} dx$$

Você não precisa memorizar ou entender os detalhes da fórmula. Entretanto, note que essa fórmula é completamente definida por apenas dois parâmetros:  $m$  e  $s$ . O restante dos símbolos na fórmula representam os extremos do intervalo que determinamos,  $a$  e  $b$ , além das constantes matemáticas conhecidas  $\pi$  e  $e$ . Esses dois parâmetros,  $m$  e  $s$ , são conhecidos como média (do inglês *average* ou *mean*) e desvio padrão (do inglês *standard deviation* ou SD) da distribuição, respectivamente.

A distribuição é simétrica, centrada na média, e a maioria dos valores (cerca de 95%) está dentro de 2 desvios padrão (DP) da média. É assim que a distribuição normal se parece quando a média é 0 e o DP é 1:



O fato de a distribuição ser definida por apenas dois parâmetros implica que, se a distribuição de um conjunto de dados puder ser aproximada por uma distribuição normal, todas as informações necessárias para descrever a distribuição poderão ser codificadas em apenas dois números: a média e o desvio padrão. Agora vamos definir esses valores para uma lista arbitrária de números.

Para uma lista de números contidos em um vetor  $x$ , a média é definida como:

```
m <- sum(x)/ length(x)
```

e o DP é definido como:

```
s <- sqrt(sum((x-mu)^2)/ length(x))
```

que pode ser interpretado como a distância média entre os valores e sua média.

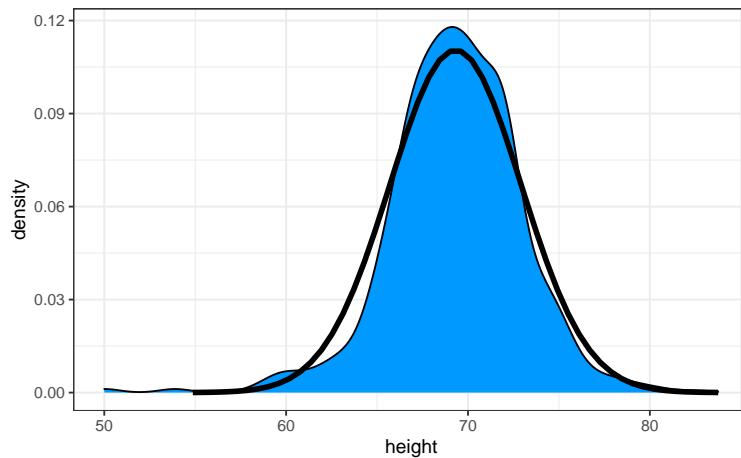
Vamos calcular os valores para alturas de homens que armazenaremos no objeto  $x$ :

```
index <- heights$sex == "Male"
x <- heights$height[index]
```

As funções predefinidas `mean` e `sd` podem ser usadas aqui (note que, por razões que serão explicadas na Seção ??, `sd` é calculado dividindo por `length(x)-1` em vez de `length(x)`):

```
m <- mean(x)
s <- sd(x)
c(average = m, sd = s)
#> average sd
#> 69.31 3.61
```

Aqui está um gráfico da densidade suave. A distribuição normal com média = 69.3 e DP = 3.6 foi plotada como uma linha preta com a curva de densidade suave da altura dos alunos em azul:



A distribuição normal parece ser uma boa aproximação aqui. Veremos agora como essa aproximação funciona para prever a proporção de valores dentro dos intervalos.

## 8.9 Unidades padrão

Para dados que estão aproximadamente com distribuição normal, é conveniente pensar em termos de unidades padrão. A unidade padrão de um valor nos diz a quantos desvios padrão da média ele está. Especificamente, para um valor  $x$  de um vetor  $X$ , definimos o valor de  $x$  em unidades padrão como  $z = (x - \bar{m})/s$ , sendo  $\bar{m}$  e  $s$  a média e desvio padrão de  $X$ , respectivamente. Por que é conveniente fazer isso?

Primeiro, olhe novamente a fórmula da distribuição normal e note que o que está sendo exposto é  $-z^2/2$  com  $z$  equivalente a  $x$  em unidades padrão. Uma vez que o máximo de  $e^{-z^2/2}$  é alcançado quando  $z = 0$ , isso explica por que o máximo da distribuição ocorre na média. Também explica a simetria, pois  $-z^2/2$  é simétrico em torno de 0. Além disso, observe que, se convertermos dados normalmente distribuídos em unidades padrão, poderemos saber rapidamente, por exemplo, se uma pessoa está na média ( $z = 0$ ), entre os maiores ( $z \approx 2$ ), entre os menores ( $z \approx -2$ ) ou apresenta uma ocorrência extremamente rara ( $z > 3$  ou  $z < -3$ ). Lembre-se de que, independentemente das unidades originais, essas regras se aplicam a todos os dados que são aproximadamente normais.

Em R, podemos obter as unidades padrão usando a função `scale`:

```
z <- scale(x)
```

Agora, para ver quantos homens estão dentro de 2 DP da média, simplesmente digitamos:

```
mean(abs(z) < 2)
#> [1] 0.95
```

A proporção é de aproximadamente 95%, que é o que a distribuição normal prevê! Para

ter ainda mais confirmação de que a aproximação é precisa, podemos usar gráficos QQ (*quantile-quantile plots*).

## 8.10 Gráficos QQ

Uma maneira sistemática de avaliar quão bem a distribuição normal se ajusta aos dados é verificar se as proporções observadas e previstas correspondem. Em geral, essa é a abordagem do gráfico quantil-quantil (Gráfico QQ ou QQ-plot).

Primeiro, vamos definir os quantis teóricos para a distribuição normal. Nos livros de estatística, usamos o símbolo  $\Phi(x)$  para definir a função que nos dá a probabilidade de que uma distribuição normal padrão seja menor que  $x$ . Por exemplo,  $\Phi(-1.96) = 0.025$  e  $\Phi(1.96) = 0.975$ . Em R, podemos avaliar  $\Phi$  usando a função `pnorm`:

```
pnorm(-1.96)
#> [1] 0.025
```

A função inversa  $\Phi^{-1}(x)$  nos fornece os quantis teóricos da distribuição normal. Por exemplo,  $\Phi^{-1}(0.975) = 1.96$ . Em R, podemos avaliar o inverso de  $\Phi$  usando a função `qnorm`.

```
qnorm(0.975)
#> [1] 1.96
```

Note que esses cálculos são para a distribuição normal padrão (média = 0, desvio padrão = 1), mas também podemos defini-los para qualquer distribuição normal. Podemos fazer isso usando os argumentos `mean` e `sd` nas funções `pnorm` e `qnorm`. Por exemplo, podemos usar `qnorm` para determinar quantis de uma distribuição com média e desvio padrão específicos:

```
qnorm(0.975, mean = 5, sd = 2)
#> [1] 8.92
```

Para a distribuição normal, todos os cálculos relacionados aos quantis são executados sem dados, por isso recebem o nome de *quantis teóricos*. Mas quantis podem ser definidos para qualquer distribuição, mesmo empíricas. Então, se tivermos dados em um vetor  $x$ , podemos definir o quantil associado a qualquer proporção  $p$ , assim como o  $q$  para os quais a proporção de valores abaixo de  $q$  é  $p$ . Usando o código R, podemos definir  $q$  como o valor pelo qual `mean(x <= q) = p`. Note que nem todo  $p$  tem um  $q$  para o qual a proporção é exatamente  $p$ . Existem várias maneiras de definir os melhores  $q$  conforme discutido na página de ajuda da função `quantile`.

Como um exemplo rápido, para dados de altura masculina, vemos que:

```
mean(x <= 69.5)
#> [1] 0.515
```

Portanto, cerca de 50% são menores ou iguais a 69 polegadas. Isso implica que se  $p = 0.50$ , então  $q = 69.5$ .

A ideia de um gráfico QQ é que, se seus dados forem bem aproximados pela distribuição normal, os quantis de seus dados deverão ser semelhantes aos quantis de uma distribuição normal. Para criar um gráfico QQ, fazemos o seguinte:

1. Definimos um vetor de  $m$  dimensões  $p_1, p_2, \dots, p_m$ .

2. Definimos um vetor de quantis  $q_1, \dots, q_m$  para as proporções  $p_1, \dots, p_m$  usando seus dados. Nos referimos a eles como *quantis de amostra*.
3. Definimos um vetor de quantis teóricos para as proporções  $p_1, \dots, p_m$  para uma distribuição normal com a mesma média e desvio padrão dos dados.
4. Plotamos os quantis da amostra versus os quantis teóricos.

Vamos construir um diagrama QQ usando um código em R. Comece definindo o vetor de proporções.

```
p <- seq(0.05, 0.95, 0.05)
```

Para obter os quantis dos dados, podemos usar a função `quantile` assim:

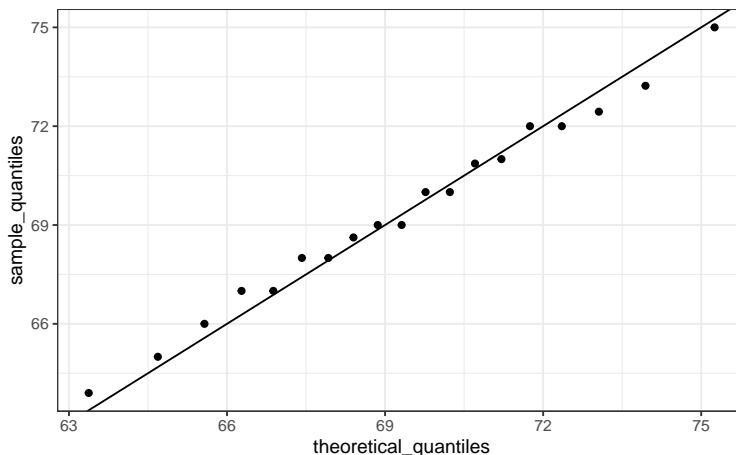
```
sample_quantiles <- quantile(x, p)
```

Para obter os quantis teóricos da distribuição normal com a média e o desvio padrão correspondentes, usamos a função `qnorm`:

```
theoretical_quantiles <- qnorm(p, mean = mean(x), sd = sd(x))
```

Para ver se eles correspondem ou não, nós os plotamos um contra o outro e desenhamos a linha de identidade:

```
qplot(theoretical_quantiles, sample_quantiles) + geom_abline()
```



Observe que esse código é muito mais limpo se usarmos unidades padrão:

```
sample_quantiles <- quantile(z, p)
theoretical_quantiles <- qnorm(p)
qplot(theoretical_quantiles, sample_quantiles) + geom_abline()
```

O código acima foi apresentado para ajudar a melhor descrever o conceito dos gráficos QQ. No entanto, na prática, é mais fácil usar as funções do `ggplot2`, descritos na Seção 8.16:

```
heights %>% filter(sex == "Male") %>%
 ggplot(aes(sample = scale(height))) +
 geom_qq() +
 geom_abline()
```

Enquanto na ilustração anterior usamos 20 quantis, a função `geom_qq` usa, por padrão, a mesma quantidade de quantis que dados.

## 8.11 Percentis

Antes de continuar, vamos definir alguns termos que são comumente usados na análise exploratória de dados.

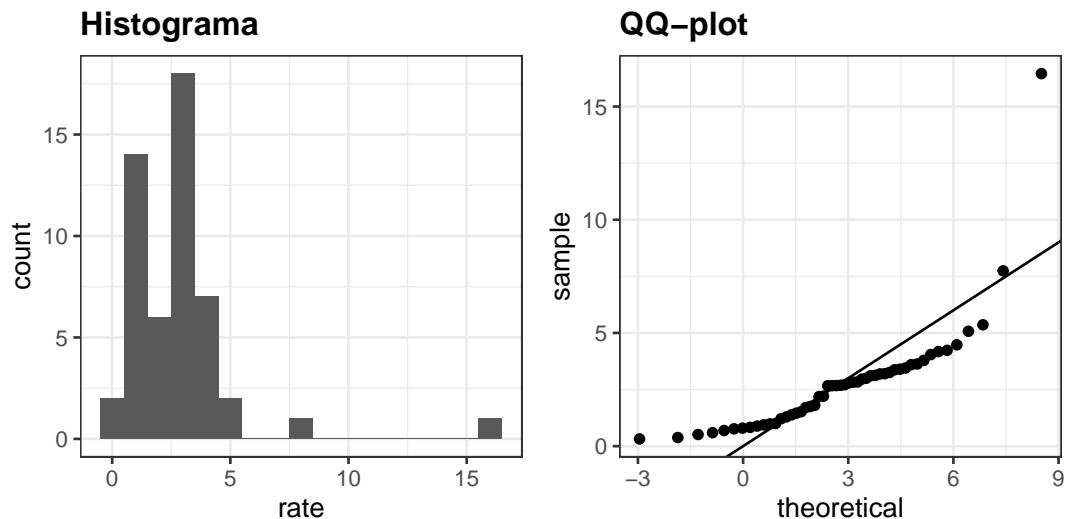
Percentis (*percentiles*) são formas de quantis comumente usadas. Percentis são quantis que podem ser obtidos ao definir  $p$  como  $0.01, 0.02, \dots, 0.99$ . Por exemplo,  $p = 0.25$  é denominado como 25º percentil, pois fornece um número para o qual 25% dos dados estão abaixo. O percentil mais famoso é 50º, também conhecido como *mediana*.

Para a distribuição normal, a mediana e a média são as mesmas, mas geralmente não é o caso.

Outro caso especial que recebe um nome são os *quartis*, obtidos através da configuração  $p = 0.25, 0.50$  e  $0.75$ .

## 8.12 Diagramas de caixas (*boxplots*)

Para apresentar os *boxplots* (diagramas de caixas), usaremos mais uma vez os dados de assassinatos nos EUA. Vamos supor que queremos sumarizar a distribuição da taxa de homicídios. Usando as técnicas de visualização de dados que aprendemos anteriormente, podemos ver que a aproximação normal não se aplica aqui:

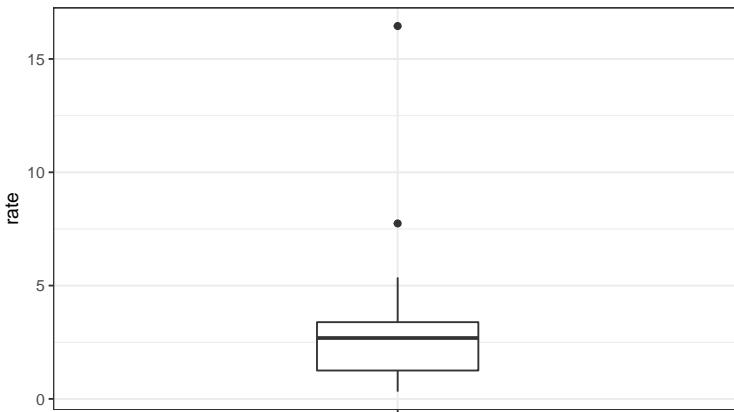


Nesse caso, o histograma acima ou um gráfico de densidade suave serviria como um resumo relativamente sucinto.

Agora, suponha que aqueles que estão acostumados a receber apenas dois números como resumos solicitem uma summarização numérica mais compacta.

Aqui, Tukey ofereceu alguns conselhos. Primeiramente, ele sugeriu fornecer um resumo de cinco números compostos pelo intervalo, juntamente com os quartis (percentis 25, 50 e 75).

Além disso, Tukey sugeriu ignorar *outliers* ao calcular o intervalo, sendo preferível plotá-los como pontos independentes. Ofereceremos uma explicação detalhada dos outliers mais adiante. Por fim, ele recomendou que representássemos graficamente esses números como uma “caixa” com “fios de bigode”, assim:



com a caixa definida pelos percentis 25% e 75% e os bigodes mostrando o intervalo. A distância entre esses dois é chamada de intervalo *interquartil*. Os dois pontos são representam valores discrepantes (*outliers*), conforme definido por Tukey. A mediana é mostrada com uma linha horizontal. Hoje, chamamos esse gráfico de *boxplot*.

A partir desse simples gráfico, sabemos que a mediana é de aproximadamente 2,5, que a distribuição não é simétrica e que o intervalo varia de 0 a 5 para a grande maioria dos estados, com duas exceções.

Discutimos como criar *boxplots* na Seção 8.16.

### 8.13 Estratificação

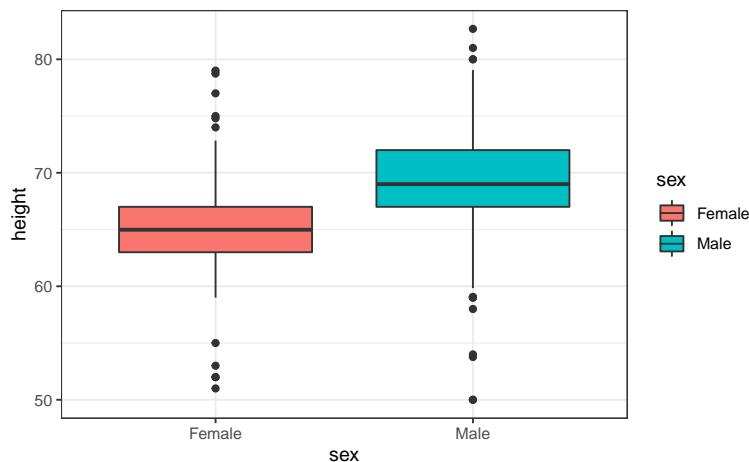
Na análise de dados, geralmente dividimos as observações em grupos com base nos valores de uma ou mais variáveis associadas a essas observações. Por exemplo, na próxima seção, dividiremos os valores de altura em grupos de acordo com uma variável de gênero: mulheres e homens. Chamamos esse procedimento de *estratificação* e nos referimos aos grupos resultantes como estratos (*strata*).

A estratificação é comum na visualização de dados, pois geralmente estamos interessados em saber como a distribuição de variáveis difere entre diferentes subgrupos. Veremos vários exemplos ao longo desta parte do livro. Além disso, revisaremos o conceito de estratificação quando aprendermos sobre regressão no Capítulo ?? e na parte de aprendizagem da máquina do livro.

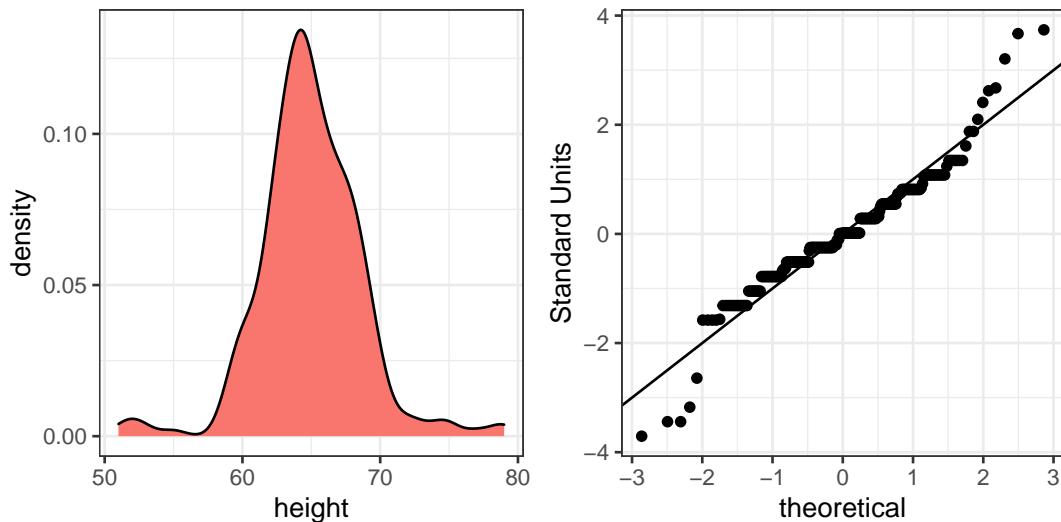
### 8.14 Estudo de caso: descrevendo as alturas dos alunos (continuação)

Usando histogramas, gráficos de densidade e gráficos QQ, ficamos convencidos de que os dados de altura masculina se aproximam muito de uma distribuição normal. Nesse caso, damos ao ET um resumo muito sucinto: as alturas masculinas seguem uma distribuição normal com uma média de 69.3 polegadas e um desvio padrão de 3.6 polegadas. Com essas informações, o ET terá uma boa ideia do que esperar quando conhecer nossos alunos do sexo masculino. No entanto, para fornecer uma visão geral, também devemos fornecer um resumo das alturas femininas.

Aprendemos que os boxplots são úteis quando queremos comparar rapidamente duas ou mais distribuições. Aqui vemos as alturas para homens e mulheres:



O diagrama mostra imediatamente que os homens são, em média, mais altos que as mulheres. Além disso, os desvios padrão parecem ser semelhantes. Entretanto, será que a aproximação normal também funciona para os dados de altura feminina coletados pela pesquisa? Esperamos que eles sigam uma distribuição normal, assim como os meninos. No entanto, gráficos exploratórios revelam que a aproximação não é tão útil:



Vemos algo que não vemos nos meninos: o gráfico de densidade tem uma segunda “protuberância”. Além disso, o gráfico QQ mostra que os pontos mais altos tendem a ser maiores do que o esperado pela distribuição normal. Também vemos cinco pontos no gráfico QQ que sugerem alturas inferiores às esperadas para uma distribuição normal. Ao reportar novamente ao ET, podemos precisar fornecer um histograma das alturas femininas, em vez de apenas a média e o desvio padrão.

No entanto, ao reler a citação de Tukey percebemos algo que não esperávamos ver. Se olharmos para outros conjuntos de dados de distribuição de alturas femininas, podemos descobrir que elas são bem aproximadas com uma distribuição normal. Então, por que nossas alunas são diferentes? Seria nossa classe composta por um time de basquete feminino? Será que há uma parte das mulheres dizendo ser mais altas do que realmente são? Uma outra explicação mais provável pode estar na forma em que os alunos entraram em suas alturas. FEMALE (feminino) era a opção de gênero padrão, logo alguns homens, ao inserirem suas alturas, podem ter esquecido de mudar essa opção. De qualquer forma, a visualização de dados ajudou a descobrir uma possível falha em nossos dados.

Em relação aos cinco menores valores, note que esses valores são:

```
heights %>% filter(sex == "Female") %>%
 top_n(5, desc(height)) %>%
 pull(height)
#> [1] 51 53 55 52 52
```

Como essas alturas são inseridas pelos próprios alunos, há uma possibilidade que eles desejavam dizer na verdade 5'1", 5'2", 5'3" ou 5'5" (medida em pés e polegadas).

## 8.15 Exercícios

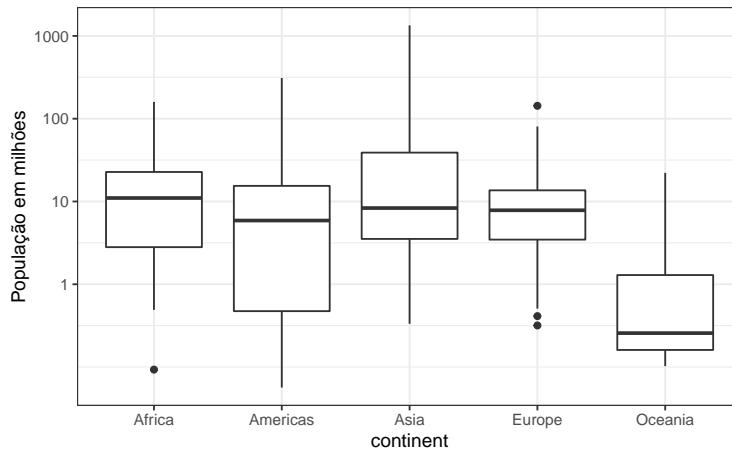
- Defina variáveis contendo alturas de homens e mulheres desta maneira:

```
library(dslabs)
data(heights)
male <- heights$height[heights$sex == "Male"]
female <- heights$height[heights$sex == "Female"]
```

Quantas medições temos para cada uma delas?

2. Suponha que não possamos fazer um gráfico e queremos comparar as distribuições lado a lado. Não podemos simplesmente listar todos os números. Em vez disso, veremos os percentis. Crie uma tabela com cinco linhas mostrando os percentis 10, 30, 50, 70 e 90 para cada um dos sexos. Denomine as tabelas como `female_percentiles` para mulheres e `male_percentiles` para homens. Em seguida, crie um *data frame* inserindo cada tabela como uma coluna.

3. Analise os seguintes *boxplots* mostrando os tamanhos de população por país:



Qual continente tem o país com o maior tamanho populacional?

4. Qual continente tem o maior tamanho médio de população?
  5. Qual é o tamanho médio da população da África (arredonde para o valor mais próximo)?
  6. Qual a proporção de países na Europa com populações inferiores a 14 milhões?
    - a. 0,99
    - b. 0,75
    - c. 0,50
    - d. 0,25
  7. Se usarmos uma transformação logarítmica, qual dos continentes apresentados anteriormente apresenta o maior intervalo interquartil?
  8. Carregue o conjunto de dados de alturas (`heights`) e crie um vetor `x` com apenas as alturas masculinas:
- ```
library(dslabs)
data(heights)
x <- heights$height[heights$sex=="Male"]
```

Qual a proporção dos dados entre 69 e 72 polegadas (maior que 69, mas menor ou igual a 72)? Dica: use um operador lógico e a função `mean`.

9. Suponha que a única coisa que você saiba sobre os dados seja a média e o desvio padrão. Use a aproximação normal para estimar a proporção que você acabou de calcular. Dica: comece calculando a média e o desvio padrão. Então, use a função `pnorm` prever as proporções.
10. Observe que a aproximação calculada na pergunta nove está muito próxima do cálculo exato realizado na primeira questão. Agora faça a mesma tarefa para valores mais extremos. Compare o cálculo exato e a aproximação normal do intervalo [79,81]. Quantas vezes maior é a proporção real do que a aproximação?
11. Considere a distribuição de homens adultos no mundo como uma distribuição normal, com uma altura média de 69 polegadas e um desvio padrão de 3 polegadas. Usando essa aproximação, encontre a proporção de homens adultos com 7 pés de altura (equivalente a 84 polegadas ou 2,13 metros) ou mais, conhecidos como “seven-footers”. Dica: use a função `pnorm`.
12. Existem cerca de um bilhão de homens entre 18 e 40 anos no mundo. Use sua resposta à pergunta anterior para estimar quantos desses homens (18 a 40 anos) têm sete pés de altura (84” ou 2,13 m) ou mais no mundo.
13. Existem cerca de 10 jogadores da liga americana de basquete (NBA - *National Basketball Association*) com 7 pés de altura ou mais. Usando a resposta para as duas perguntas anteriores, qual a proporção de *seven-footers* do mundo, com idades entre 18 e 40 anos, na NBA?
14. Repita os cálculos feitos na pergunta anterior para a altura do jogador de basquete LeBron James: 6 pés e 8 polegadas (equivalente a 80” ou 2,03m). Existem cerca de 150 jogadores que são pelo menos tão altos.
15. Ao responder às perguntas anteriores, descobrimos que não é raro que alguém com de sete pés de altura se torne jogador da NBA. Assim, qual das opções a seguir representa uma crítica justa aos nossos cálculos?

- a. Prática e talento são o que fazem um ótimo jogador de basquete, não a altura.
- b. A aproximação normal não é apropriada para alturas.
- c. Conforme observado na questão 10, a aproximação normal tende a subestimar valores discrepantes. É possível que existam mais *seven-footers* do que previmos.
- d. Conforme observado na questão 10, a aproximação normal tende a superestimar valores discrepantes. É possível que existam menos *seven-footers* do que previmos.

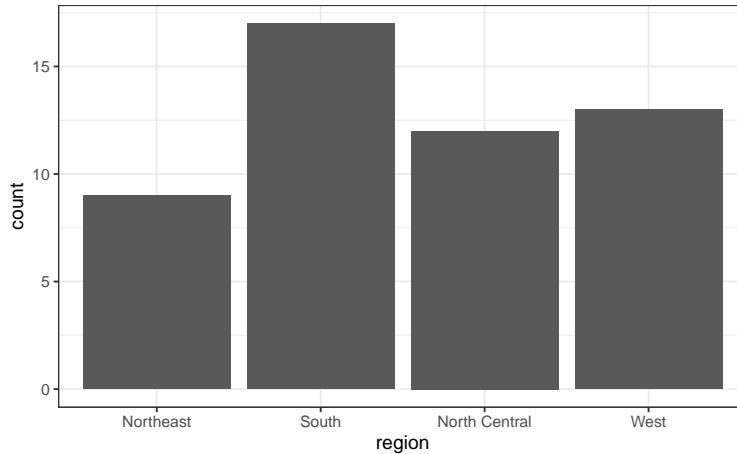
8.16 Geometrias ggplot2

No capítulo 7 apresentamos o pacote **ggplot2** para visualização de dados. Aqui, demonstramos como gerar gráficos relacionados a distribuições, especificamente os gráficos mostrados anteriormente neste capítulo.

8.16.1 Gráficos de barra

Para gerar um gráfico de barras (*barplot* em inglês), podemos usar a geometria `geom_bar`. Por padrão, R conta o número de ocorrências em cada categoria e desenha uma barra. Aqui vemos o gráfico de barras para as regiões dos Estados Unidos.

```
murders %>% ggplot(aes(region)) + geom_bar()
```

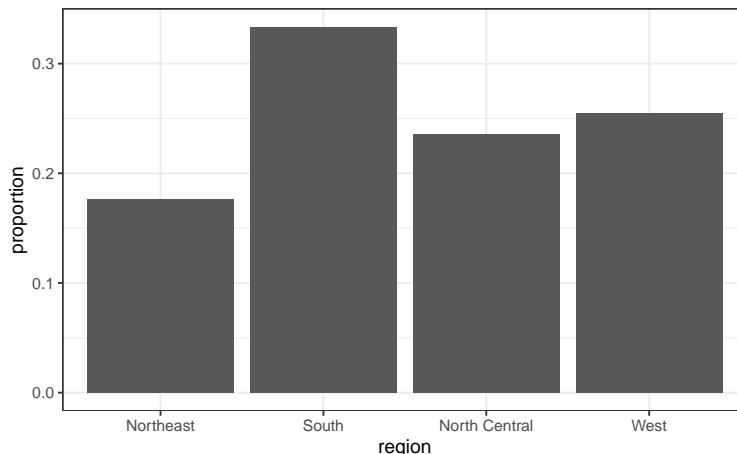


Muitas vezes, já temos uma tabela com a distribuição que queremos apresentar como um gráfico de barras. Como por exemplo:

```
data(murders)
tab <- murders %>%
  count(region) %>%
  mutate(proportion = n/sum(n))
tab
#> #>   region   n proportion
#> #> 1   Northeast 9  0.176
#> #> 2   South    17 0.333
#> #> 3   North Central 12 0.235
#> #> 4   West     13 0.255
```

Não queremos mais que `geom_bar` realize as contagens, mas que apenas represente graficamente uma barra para a altura fornecida pela variável `proportion`. Para isso, precisamos fornecer as categorias (`x`) e os valores (`y`), além de usar a opção `stat="identity"`.

```
tab %>% ggplot(aes(region, proportion)) + geom_bar(stat = "identity")
```



8.16.2 Histogramas

Para gerar histogramas, podemos utilizar a função `geom_histogram`. Ao analisar a página de ajuda dessa função, vemos que o único argumento necessário é `x`, a variável para a qual construiremos um histograma. Neste caso, ele não será necessário declará-lo, pois sabemos que `x` é o primeiro argumento usado. O código fica assim:

```
heights %>%
filter(sex == "Female") %>%
ggplot(aes(height)) +
geom_histogram()
```

Se executarmos o código acima, ele retornará uma mensagem:

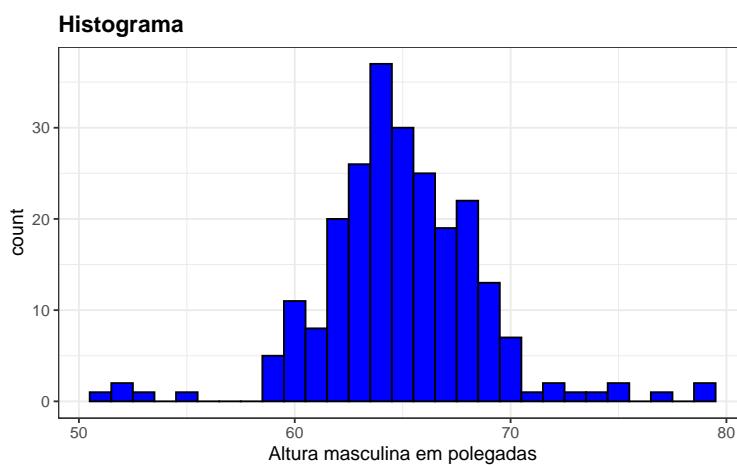
`stat_bin()` usando `bins = 30`. Escolha um valor melhor usando `binwidth`.

Anteriormente, usamos um tamanho de compartimento de 1 polegada; portanto, o código ficará assim:

```
heights %>%
filter(sex == "Female") %>%
ggplot(aes(height)) +
geom_histogram(binwidth = 1)
```

Finalmente, se por razões estéticas quisermos adicionar cores, podemos usar os argumentos descritos na página de ajuda. Podemos ainda adicionar rótulos e um título:

```
heights %>%
filter(sex == "Female") %>%
ggplot(aes(height)) +
geom_histogram(binwidth = 1, fill = "blue", col = "black") +
xlab("Altura masculina em polegadas") +
ggtitle("Histograma")
```



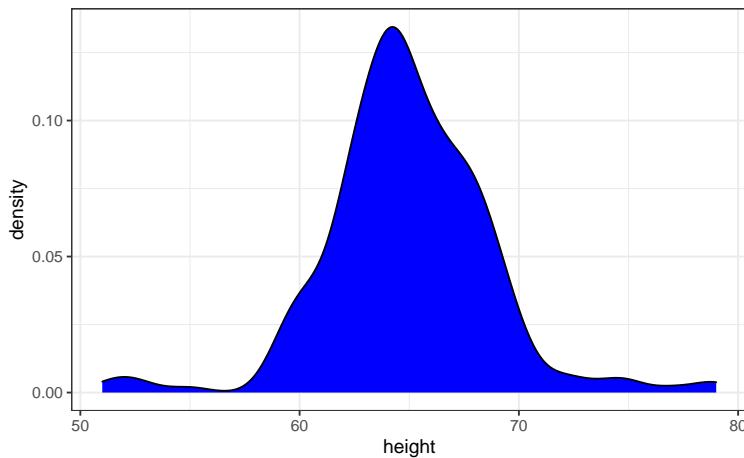
8.16.3 Gráficos de densidade

Para criar um gráfico de densidade suave, usamos `geom_density`. Assim, para fazer um gráfico de densidade suave com os dados que visualizamos anteriormente como um histograma, podemos usar este código:

```
heights %>%
filter(sex == "Female") %>%
ggplot(aes(height)) +
geom_density()
```

Para preencher com cores, podemos usar o argumento `fill`.

```
heights %>%
filter(sex == "Female") %>%
ggplot(aes(height)) +
geom_density(fill="blue")
```

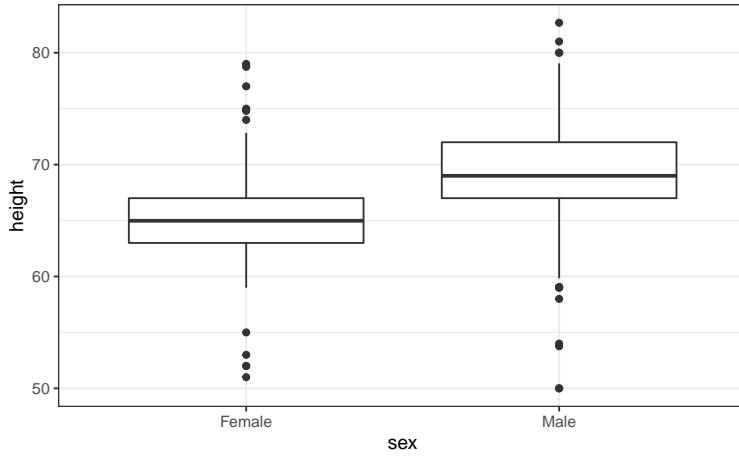


Para alterar a suavidade da densidade, usamos o argumento `adjust` para multiplicar o valor padrão pelo ajuste. Por exemplo, se queremos que o parâmetro de suavização seja duas vezes maior, usamos:

```
heights %>%
filter(sex == "Female") +
geom_density(fill="blue", adjust = 2)
```

8.16.4 Boxplots

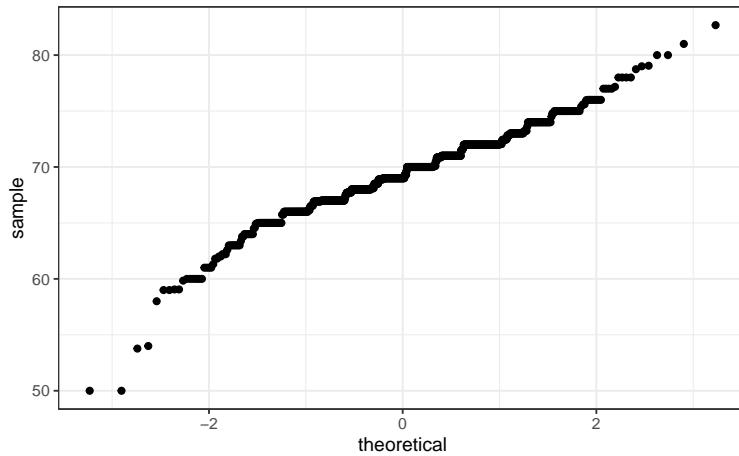
A geometria para a criação de gráficos de caixas é a `geom_boxplot`. Como já discutimos, os *boxplots* são úteis para comparar distribuições. Por exemplo, abaixo vemos as alturas mostradas anteriormente para as mulheres comparadas às alturas dos homens. Para essa geometria, precisamos dos argumentos `x` com as categorias e `y` com os valores:



8.16.5 Gráficos QQ

Para gráficos QQ, usamos a geometria `geom_qq`. Na página de ajuda, aprendemos que precisamos especificar o `sample` (aprenderemos sobre amostras em um capítulo posterior). Aqui está o gráfico QQ para as alturas masculinas:

```
heights %>% filter(sex=="Male") %>%
ggplot(aes(sample = height)) +
geom_qq()
```



Por padrão, a variável de amostra é comparada a uma distribuição normal com uma média de 0 e um desvio padrão de 1. Para alterar isso, usamos o argumento `dparams` de acordo com a página de ajuda. Para adicionar uma linha de identidade, basta atribuir uma outra camada. Para linhas retas, usamos a função `geom_abline`. Por padrão, a linha de identidade tem inclinação = 1 (`slope = 1`) e interceptação = 0 (`intercept = 0`).

```
params <- heights %>% filter(sex=="Male") %>%
summarize(mean = mean(height), sd = sd(height))

heights %>% filter(sex=="Male") %>%
ggplot(aes(sample = height)) +
```

```
geom_qq(dparams = params) +
geom_abline()
```

Outra opção aqui é dimensionar os dados primeiro e, em seguida, plotar o *QQplot* contra a distribuição normal padrão.

```
heights %>%
filter(sex=="Male") %>%
ggplot(aes(sample = scale(height))) +
geom_qq() +
geom_abline()
```

8.16.6 Imagens

Não tivemos que usar imagens para os conceitos descritos neste capítulo, mas as usaremos na Seção 10.14. Por isso, apresentaremos a seguir as duas geometrias usadas para criar imagens: **geom_tile** e **geom_raster**. Elas se comportam de maneira semelhante (para ver o que as difere, consulte a página de ajuda). Para criar uma imagem em **ggplot2**, precisamos de um *data frame* com as coordenadas x e y, além dos valores associados a cada uma delas. Aqui temos um *data frame*:

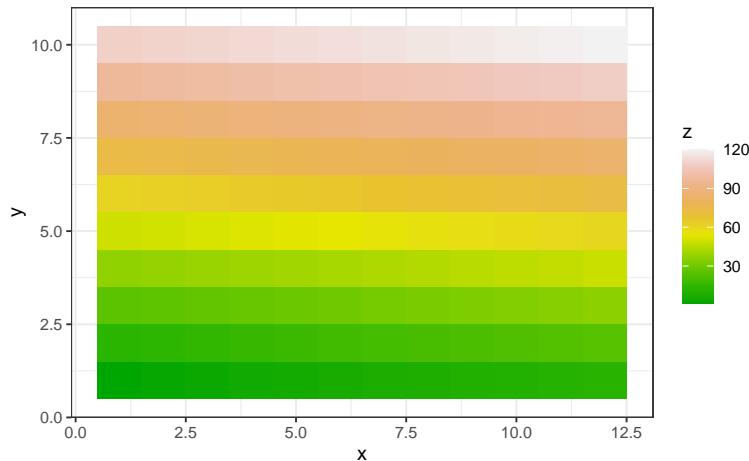
```
x <- expand.grid(x = 1:12, y = 1:10) %>%
mutate(z = 1:120)
```

Note que esta é a versão organizada de uma matriz, `matrix(1:120, 12, 10)`. Para representar graficamente a imagem, usamos o seguinte código:

```
x %>% ggplot(aes(x, y, fill = z)) +
geom_raster()
```

Para essas imagens, geralmente é necessário alterar a escala de cores. Isso pode ser feito através da camada `scale_fill_gradientn`.

```
x %>% ggplot(aes(x, y, fill = z)) +
geom_raster() +
scale_fill_gradientn(colors = terrain.colors(10))
```



8.16.7 Gráficos rápidos

Na seção 7.13, introduzimos `qplot` como uma função útil quando precisamos fazer um rápido diagrama de dispersão. Também podemos usar `qplot` para criar histogramas, gráficos de densidade, *boxplots*, gráficos QQ e muito mais. Embora não forneça o nível de controle do `ggplot`, `qplot` é definitivamente útil, pois permite criar um gráfico com um pequeno trecho de código.

Suponha que tenhamos as alturas femininas em um objeto `x`:

```
x <- heights %>%
  filter(sex=="Male") %>%
  pull(height)
```

Para fazer um histograma rápido, podemos usar:

```
qplot(x)
```

A função supõe que queremos criar um histograma porque fornecemos apenas uma variável. Na seção 7.13 vimos que, se fornecermos duas variáveis para `qplot`, ele cria automaticamente um diagrama de dispersão.

Para fazer um gráfico QQ rápido, você pode usar o argumento `sample`. Note que podemos adicionar camadas, como fazemos com `ggplot`.

```
qplot(sample = scale(x)) + geom_abline()
```

Se fornecermos um *factor* e um vetor numérico, obteremos um gráfico como o que veremos a seguir. Observe que no código estamos usando o argumento `data`. Como o *data frame* não é o primeiro argumento em `qplot`, temos que usar o operador de ponto.

```
heights %>% qplot(sex, height, data = .)
```

Também podemos selecionar uma geometria específica usando o argumento `geom`. Portanto, para converter o gráfico anterior em um *boxplot*, usamos o seguinte código:

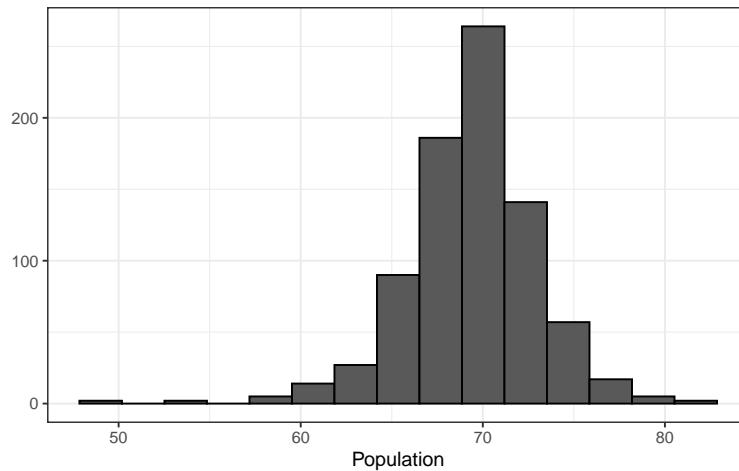
```
heights %>% qplot(sex, height, data = ., geom = "boxplot")
```

Também podemos usar o argumento `geom` para gerar um gráfico de densidade em vez de um histograma:

```
qplot(x, geom = "density")
```

Embora não tanto quanto com `ggplot`, temos alguma flexibilidade para melhorar os resultados de `qplot`. Olhando a página de ajuda, vemos várias maneiras pelas quais podemos melhorar a aparência do histograma anterior. Por exemplo:

```
qplot(x, bins=15, color = I("black"), xlab = "Population")
```



Nota técnica: o motivo pelo qual usamos `I("black")` é porque nós queremos que `qplot` trate "black" como caracteres ao invés de um *factor*. Esse é o comportamento padrão dentro do `aes`, que é chamado internamente aqui. Em geral, a função `I` é usada no R para dizer "mantenha como está".

8.17 Exercícios

1. Agora vamos usar a função `geom_histogram` para fazer um histograma das alturas no conjunto de dados `height`. Lendo a documentação para essa função, vemos que ela requer apenas um mapeamento: os valores a serem usados para o histograma. Faça um histograma de todos os gráficos.

Qual é a variável que contém as alturas?

- a. `sex`
- b. `heights`
- c. `height`
- d. `heights$height`

2. Agora crie um objeto `ggplot`. Use o *pipe* para atribuir os dados de altura ao objeto `ggplot`. Atribua `height` para os valores de `x` através da função `aes`.
3. Agora, estamos prontos para adicionar uma camada para fazer o histograma. Use o objeto criado no exercício anterior e a função `geom_histogram` para fazer o histograma.
4. Quando executamos o código no exercício anterior, recebemos o aviso: `stat_bin()` usando `bins = 30`. Escolha um valor melhor com `binwidth`.

Use o argumento `binwidth` para alterar o histograma criado no exercício anterior para usar compartimentos de 1 polegada.

5. Em vez de um histograma, vamos fazer um gráfico de densidade suave. Neste caso, não criaremos um objeto. Em vez disso, criaremos e exibiremos o gráfico com uma linha de código. Altere a geometria no código usado anteriormente para criar uma densidade suave em vez de um histograma.

6. Agora vamos fazer um gráfico de densidade para homens e mulheres separadamente. Podemos fazer isso usando o argumento `group`. Atribuímos grupos por meio de mapeamento estético, pois cada ponto precisa de um grupo antes de fazer os cálculos necessários para estimar uma densidade.

7. Também podemos atribuir grupos através do argumento `color`. Isso tem o benefício adicional de usar cores para distinguir grupos. Mude o código acima para usar cores.

8. Além disso, podemos atribuir grupos através do argumento `fill`. Esse argumento também permite usar cores para distinguir grupos, desta forma:

```
heights %>%
ggplot(aes(height, fill = sex)) +
geom_density()
```

No entanto, aqui a segunda densidade é plotada em cima da primeira. Podemos tornar as curvas mais visíveis usando *alpha blending* para adicionar transparência. Defina o parâmetro `alpha` como 0,2 na função `geom_density` para fazer essa alteração.

9

Visualização de dados na prática

Neste capítulo, demonstraremos como códigos relativamente simples usando **ggplot2** podem criar gráficos esclarecedores e esteticamente agradáveis. Como motivação, criaremos gráficos que nos ajudarão a entender melhor as tendências da saúde e da economia global. Vamos implementar o que aprendemos nos capítulos 7 e 8.¹⁶ e aprenderemos a expandir os códigos para aperfeiçoar os gráficos criados. À medida que prosseguimos em nosso estudo de caso, descreveremos os princípios gerais mais relevantes para a visualização de dados e aprenderemos conceitos como *faceting*, *gráficos de séries temporais*, *transformações* e *gráficos de ridge*.

9.1 Estudo de caso: novas perspectivas sobre pobreza

Hans Rosling¹ foi co-fundador da Fundação Gapminder², uma organização dedicada a educar o público através de dados para dissipar mitos comuns sobre o chamado mundo em desenvolvimento. A organização usa dados para mostrar como as tendências atuais nos campos da saúde e da economia contradizem as narrativas emanadas da cobertura sensacionalista da mídia sobre catástrofes, tragédias e outros eventos desafortunados. Conforme declarado no site da Fundação Gapminder:

Jornalistas e lobistas contam histórias dramáticas. Esse é o trabalho deles. Eles contam histórias sobre eventos extraordinários e pessoas incomuns. As pilhas de histórias dramáticas se acumulam na mente das pessoas gerando visões de mundo excessivamente dramáticas, estresse e fortes sentimentos negativos: “O mundo está ficando pior!”, “Somos nós contra eles!”, “Outras pessoas são estranhas!”, “A população continua crescendo!” e “Ninguém liga!”

Hans Rosling decidiu, de maneira dramática, educar o público sobre tendências reais orientadas a dados usando visualizações eficazes. Esta seção é baseada em duas palestras que exemplificam essa perspectiva educacional: *Novas Perspectivas Sobre a Pobreza*³ e *As Melhores Estatísticas Jamais Vistas*⁴. Especificamente, nesta seção, usamos dados para tentar responder às duas perguntas a seguir:

1. É uma caracterização justa dizer que, nos dias de hoje, o mundo está dividido em nações ocidentais ricas e nações em desenvolvimento na África, Ásia e América Latina?

¹https://pt.wikipedia.org/wiki/Hans_Rosling

²<http://www.gapminder.org/>

³https://www.ted.com/talks/hans_rosling_reveals_new_insights_on_poverty?language=pt

⁴https://www.ted.com/talks/hans_rosling_shows_the_best_stats_you_ve_ever_seen?language=pt

2. A desigualdade de renda piorou em todos os países nos últimos 40 anos?

Para responder a essas perguntas, usaremos a base de dados `gapminder` fornecida pelo `dslabs`. Esse *dataset* foi criado usando diversas planilhas disponibilizadas pela Fundação Gapminder. Você pode acessar essa tabela desta maneira:

```
library(tidyverse)
library(dslabs)
data(gapminder)
gapminder %>% as_tibble()
#> # A tibble: 10,545 x 9
#>   country    year infant_mortality life_expectancy fertility population
#>   <fct>     <int>            <dbl>           <dbl>        <dbl>       <dbl>
#> 1 Albania    1960            115.            62.9        6.19      1636054
#> 2 Algeria    1960            148.            47.5        7.65      11124892
#> 3 Angola     1960            208             36.0        7.32      5270844
#> 4 Antigua~   1960            NA              63.0        4.43      54681
#> 5 Argentin~  1960            59.9            65.4        3.11      20619075
#> # ... with 10,540 more rows, and 3 more variables: gdp <dbl>,
#> #   continent <fct>, region <fct>
```

9.1.1 Teste de Hans Rosling

Assim como na apresentação em vídeo *Novas Perspectivas Sobre a Pobreza*, começamos testando nossos conhecimentos sobre as diferenças na mortalidade infantil em diferentes países. Para cada um dos cinco pares de países abaixo, quais deles você imagina que tiveram as maiores taxas de mortalidade infantil em 2015? Quais pares você acha que são mais parecidos?

1. Sri Lanka ou Turquia
2. Polônia ou Coréia do Sul
3. Malásia ou Rússia
4. Paquistão ou Vietnã
5. Tailândia ou África do Sul

Ao responder a essas perguntas sem dados, os países não europeus geralmente são escolhidos como os que apresentam as mais altas taxas de mortalidade infantil: Sri Lanka sobre a Turquia, Coreia do Sul sobre Polônia e Malásia sobre Rússia. Também é comum supor que os países considerados parte do mundo em desenvolvimento - Paquistão, Vietnã, Tailândia e África do Sul - têm taxas de mortalidade igualmente altas.

Para responder a essas perguntas **com dados**, podemos usar `dplyr`. Por exemplo, para a primeira comparação, vemos que:

```
gapminder %>%
filter(year == 2015 & country %in% c("Sri Lanka", "Turkey")) %>%
select(country, infant_mortality)
#>   country infant_mortality
#> 1 Sri Lanka            8.4
#> 2 Turkey               11.6
```

A Turquia tem a maior taxa de mortalidade infantil.

Podemos usar esse código em todas as comparações e descobrir o seguinte:

```
#> New names:
#> * country -> country...1
#> * infant_mortality -> infant_mortality...2
#> * country -> country...3
#> * infant_mortality -> infant_mortality...4
```

country	infant mortality	country	infant mortality
Sri Lanka	8.4	Turkey	11.6
Poland	4.5	South Korea	2.9
Malaysia	6.0	Russia	8.2
Pakistan	65.8	Vietnam	17.3
Thailand	10.5	South Africa	33.6

Vemos que os países europeus desta lista têm taxas de mortalidade infantil mais altas: a Polônia tem uma taxa mais alta do que a Coreia do Sul e a Rússia tem uma taxa mais alta que a Malásia. Também vemos que o Paquistão tem uma taxa muito mais alta do que o Vietnã e a África do Sul tem uma taxa muito mais alta do que a Tailândia. Acontece que, quando Hans Rosling deu esse teste para grupos de pessoas instruídas, a pontuação média foi inferior a 2,5 em 5, pior do que eles teriam obtido se tivessem apenas chutado as respostas. Isso implica que, mais do que ignorantes, estamos mal informados. Neste capítulo, vemos como a visualização dos dados ajuda a nos informar.

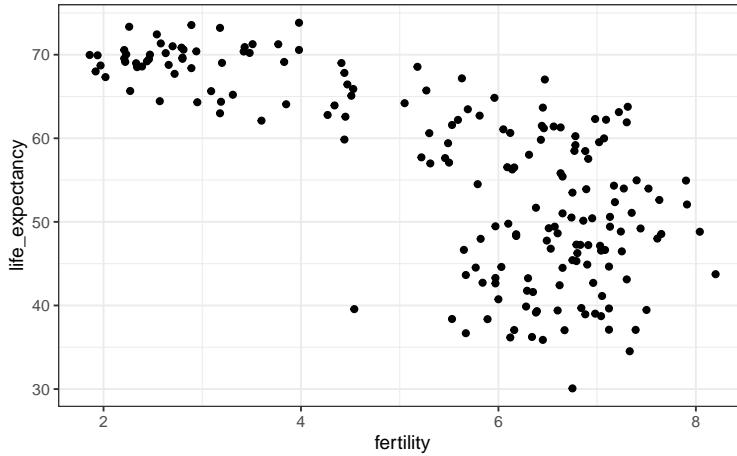
9.2 Diagramas de dispersão (*scatterplots*)

A razão disso decorre da noção preconcebida de que o mundo está dividido em dois grupos: o mundo ocidental (Europa Ocidental e América do Norte), caracterizado por uma longa expectativa de vida e famílias pequenas, versus o mundo em desenvolvimento (África, Ásia e América Latina), caracterizada por uma curta expectativa de vida e famílias numerosas. Mas os dados suportam essa visão dicotômica?

Os dados necessários para responder a essa pergunta também estão disponíveis em nossa tabela `gapminder`. Usando nossas habilidades de visualização de dados recém-aprendidas, podemos enfrentar esse desafio.

Para analisar essa visão de mundo, nosso primeiro plote é um gráfico de dispersão da expectativa de vida versus taxas de fertilidade (número médio de filhos por mulher). Começamos analisando os dados de cerca de 50 anos atrás, quando talvez essa visão tenha sido consolidada em nossas mentes.

```
filter(gapminder, year == 1962) %>%
ggplot(aes(fertility, life_expectancy)) +
geom_point()
```

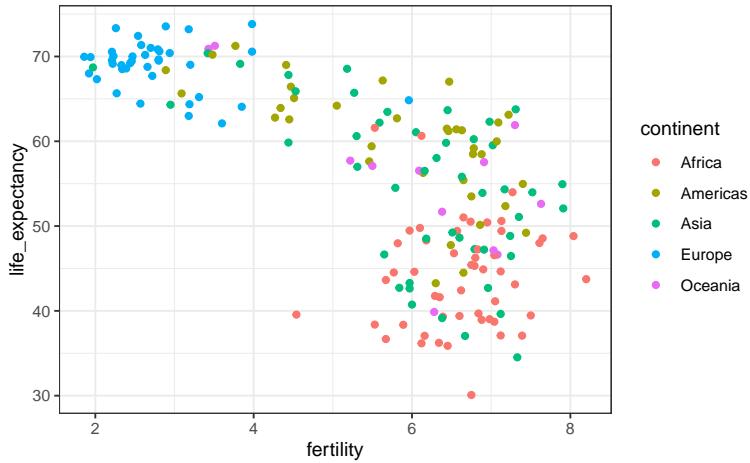


A maioria dos pontos se enquadra em duas categorias diferentes:

1. Expectativa de vida em torno de 70 anos e 3 ou menos filhos por família.
2. Expectativa de vida menor que 65 anos e mais de 5 filhos por família.

Para confirmar que esses países são das regiões que esperamos, podemos usar cores para representar os continentes.

```
filter(gapminder, year == 1962) %>%
ggplot( aes(fertility, life_expectancy, color = continent)) +
geom_point()
```



Em 1962, a visão do “Ocidente versus mundo em desenvolvimento” era baseada em uma certa realidade. Será que ainda é assim 50 anos depois?

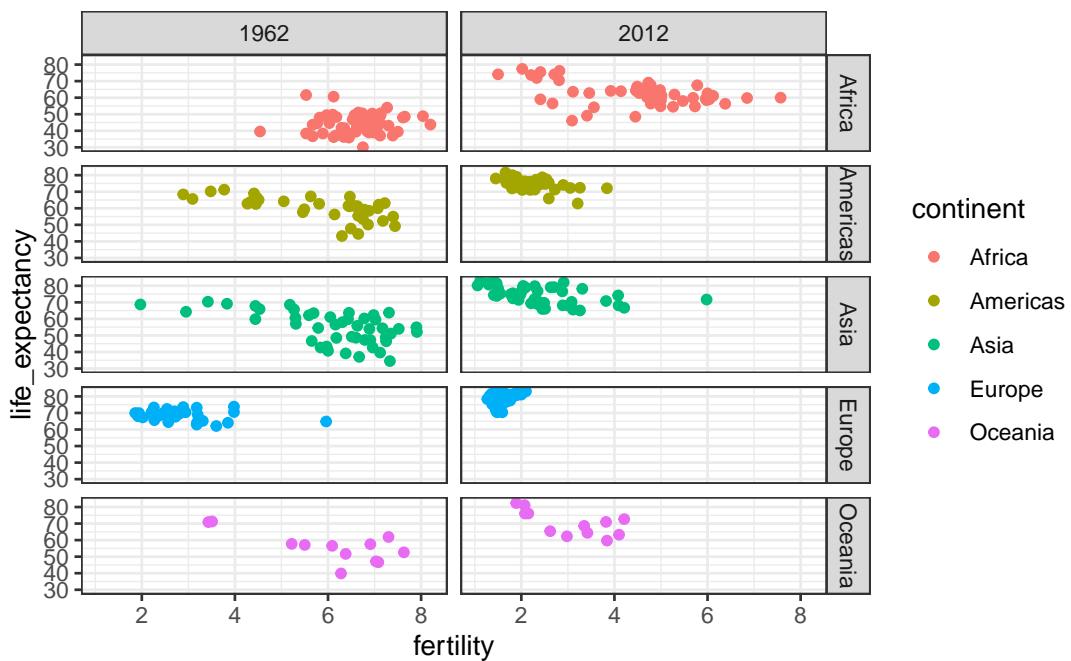
9.3 Separe em facetas

Podemos facilmente representar graficamente os dados de 2012 da mesma maneira que fizemos em 1962. No entanto, para comparação, é preferível representar graficamente lado

a lado. No **ggplot2**, conseguimos isso separando as variáveis em *facets*: estratificamos os dados por alguma variável e fazemos o mesmo gráfico para cada estrato.

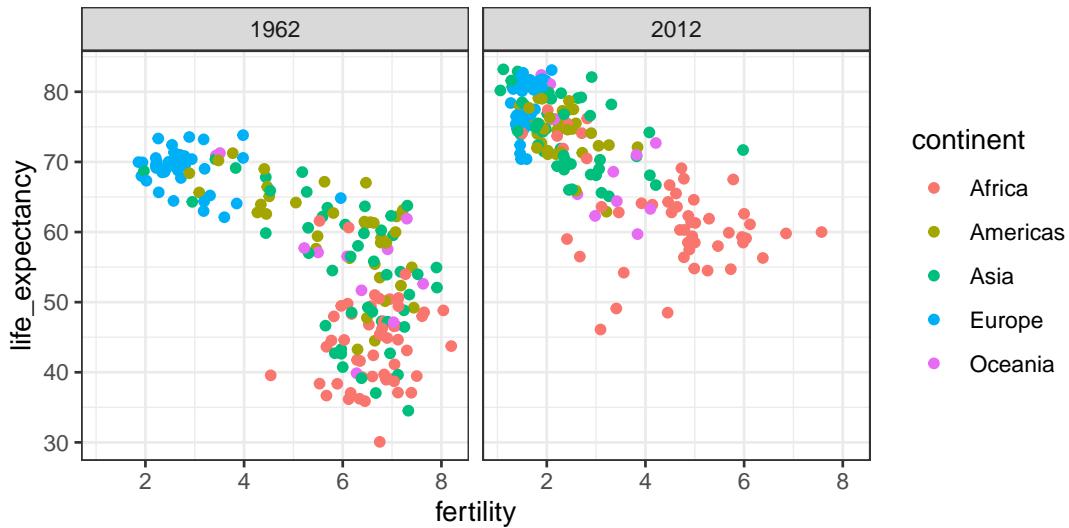
Para separar em facetas, adicionamos uma camada com a função `facet_grid`, que separa automaticamente os gráficos. Essa função permite separar até duas variáveis em facetas usando colunas para representar uma variável e linhas para representar a outra. A função espera que as variáveis de linha e coluna sejam separadas por um `~`. Aqui vemos um exemplo de um diagrama de dispersão em que adicionamos `facet_grid` como a última camada:

```
filter(gapminder, year%in%c(1962, 2012)) %>%
  ggplot(aes(fertility, life_expectancy, col = continent)) +
  geom_point() +
  facet_grid(continent ~ year)
```



Acima, vemos um gráfico para cada combinação de continente/ano. No entanto, este exemplo apresenta mais do que queremos, que é simplesmente comparar dois anos: 1962 e 2012. Nesse caso, existe apenas uma variável. Podemos usar `.` para que `facet_grid` saiba que queremos usar todas as variáveis:

```
filter(gapminder, year%in%c(1962, 2012)) %>%
  ggplot(aes(fertility, life_expectancy, col = continent)) +
  geom_point() +
  facet_grid(. ~ year)
```

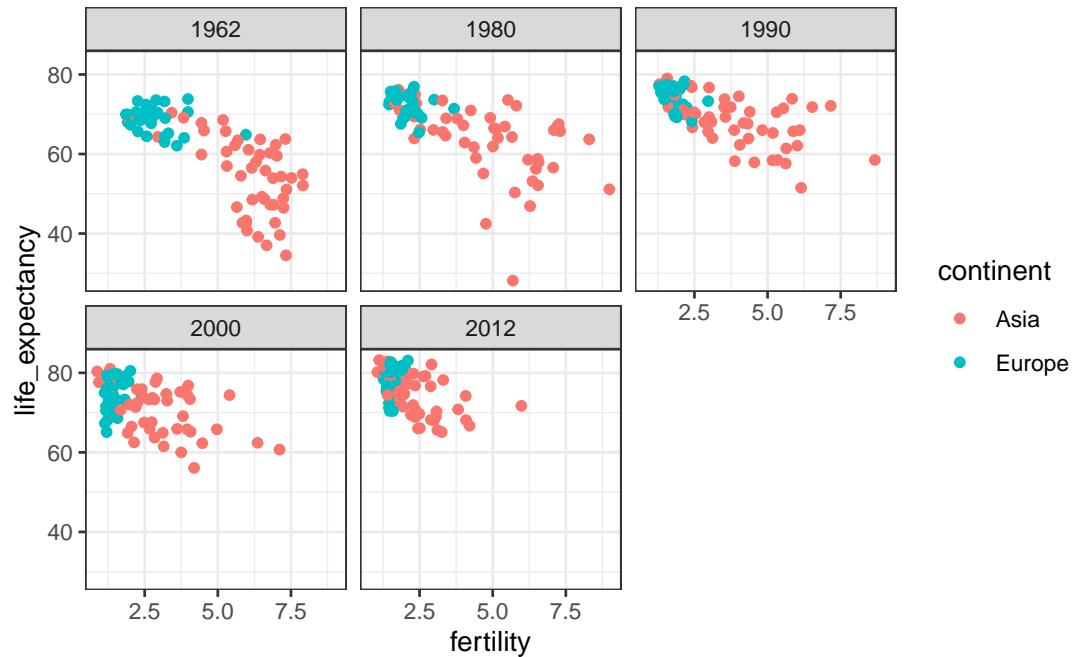


Este gráfico mostra claramente que a maioria dos países mudou do cluster de *países em desenvolvimento* para o cluster de *países ocidentais*. Em 2012, a visão do mundo ocidental versus o mundo em desenvolvimento não faz mais sentido. Isso é particularmente evidente ao comparar a Europa com a Ásia, este último agora inclui vários países que apresentaram grandes melhorias.

9.3.1 facet_wrap

Para explorar como essa transformação ocorreu ao longo dos anos, podemos fazer o gráfico para vários anos. Por exemplo, podemos adicionar os anos 1970, 1980, 1990 e 2000. No entanto, se fizermos isso, não queremos todos os gráficos na mesma linha, que é o que faz `facet_grid` por padrão, pois eles parecerão muito estreitos para exibir os dados. Em vez disso, queremos usar várias linhas e colunas. A função `facet_wrap` nos permite fazer isso automaticamente, acomodando a série de gráficos para que cada imagem tenha dimensões visíveis:

```
years <- c(1962, 1980, 1990, 2000, 2012)
continents <- c("Europe", "Asia")
gapminder %>%
filter(year %in% years & continent %in% continents) %>%
ggplot( aes(fertility, life_expectancy, col = continent)) +
geom_point() +
facet_wrap(~year)
```

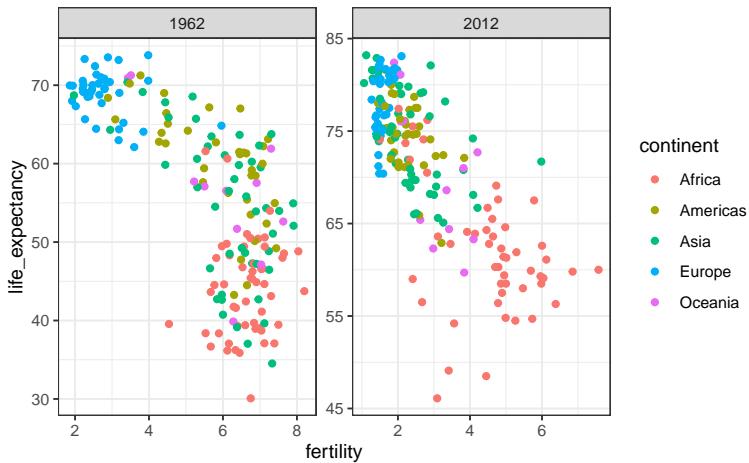


Este gráfico mostra claramente como a maioria dos países asiáticos melhorou a uma taxa muito mais rápida que os países europeus.

9.3.2 Escalas fixas para melhores comparações

A escolha padrão da faixa de eixos é importante. Quando não estiver usando `facet`, esse intervalo é determinado pelos dados mostrados no gráfico. Usando `facet`, esse intervalo é determinado pelos dados exibidos em todos os gráficos e, portanto, permanece fixo em todas as plots. Isso facilita muito nas comparações entre gráficos. Por exemplo, no gráfico acima, podemos ver que a expectativa de vida aumentou e a fertilidade diminuiu na maioria dos países. Vemos isso porque a nuvem de pontos está se movendo. Este não é o caso se ajustarmos as escalas:

```
filter(gapminder, year%in%c(1962, 2012)) %>%
ggplot(aes(fertility, life_expectancy, col = continent)) +
geom_point() +
facet_wrap(. ~ year, scales = "free")
```



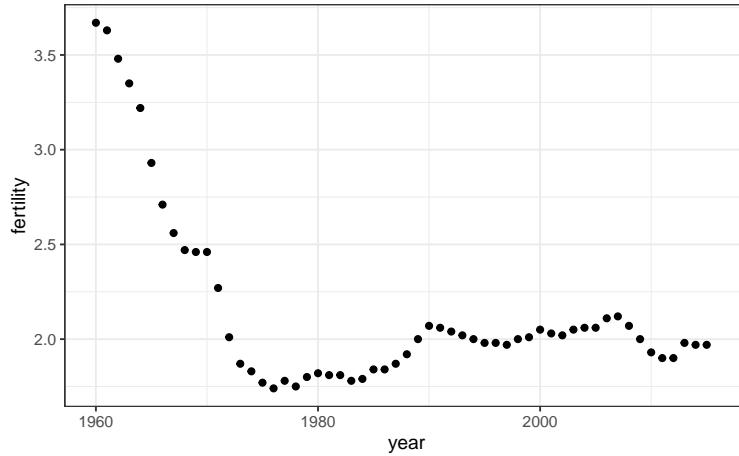
No gráfico acima, devemos prestar atenção especial ao intervalo para observar que o gráfico à direita tem uma expectativa de vida mais longa.

9.4 Gráficos de séries temporais

As visualizações acima ilustram efetivamente que os dados não são mais consistentes com a visão de mundo ocidental versus o mundo em desenvolvimento. Vendo esses gráficos, novas questões surgem. Por exemplo, quais países estão melhorando mais e quais menos? A melhoria foi constante nos últimos 50 anos ou acelerou mais em determinados períodos? Para uma análise mais detalhada que possa ajudar a responder a essas perguntas, apresentamos gráficos de séries temporais.

Os gráficos de séries temporais apresentam o tempo no eixo x e um resultado ou medida de interesse no eixo y. Por exemplo, aqui está um gráfico da tendência nas taxas de fertilidade nos Estados Unidos:

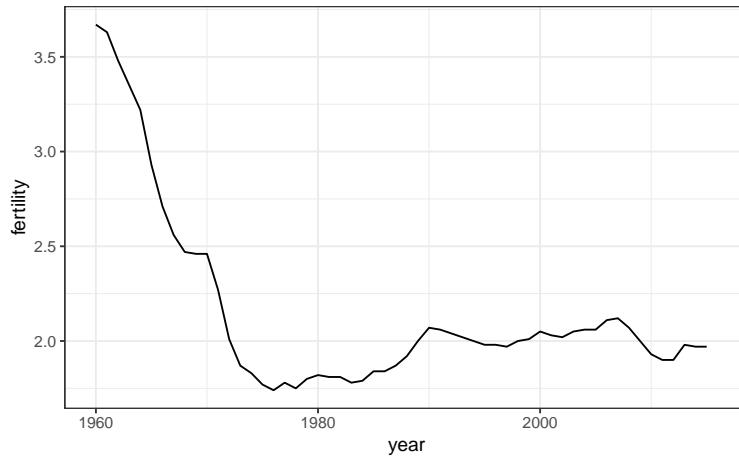
```
gapminder %>%
filter(country == "United States") %>%
ggplot(aes(year, fertility)) +
geom_point()
```



Observamos que a tendência não é linear. Em vez disso, há uma queda acentuada durante as décadas de 1960 e 1970 para menos de dois filhos em média por mulher. Em seguida, a tendência retorna a dois e se estabiliza nos anos 1990.

Quando os pontos são regularmente e densamente espaçados, como vemos acima, criamos uma curva que conecta os pontos às linhas, para transmitir a ideia de que esses dados são provenientes de uma única série, aqui representando um país. Para fazer isso, usamos a função `geom_line` em vez de `geom_point`.

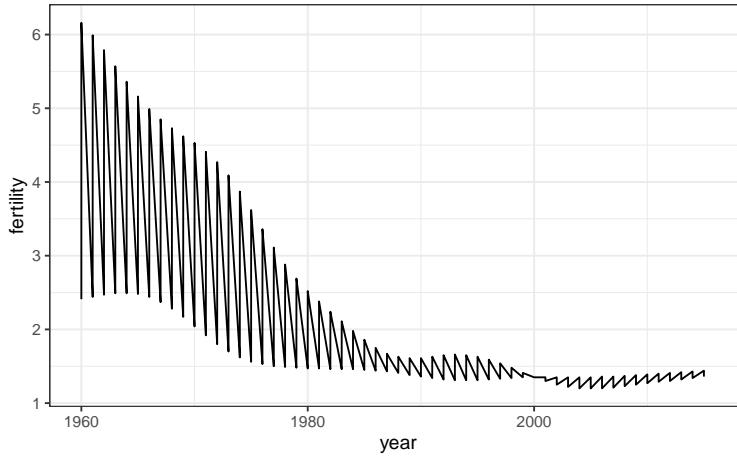
```
gapminder %>%
filter(country == "United States") %>%
ggplot(aes(year, fertility)) +
geom_line()
```



Isso é particularmente útil ao comparar dois países. Se criarmos um subconjunto dos dados para incluir dois países, um da Europa e outro da Ásia, podemos adaptar o código acima:

```
countries <- c("South Korea", "Germany")

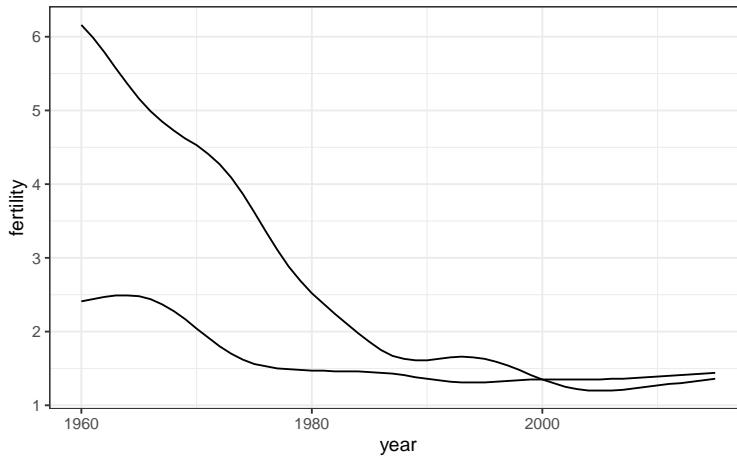
gapminder %>% filter(country %in% countries) %>%
ggplot(aes(year,fertility)) +
geom_line()
```



Claramente, esse **não** é o gráfico que queremos. Em vez de uma linha para cada país, os pontos dos dois países foram unidos porque não informamos a `ggplot` que queremos duas linhas independentes. Para que `ggplot` entenda que existem duas curvas que devem ser feitas separadamente, atribuímos cada ponto a um `group`, um para cada país:

```
countries <- c("South Korea", "Germany")

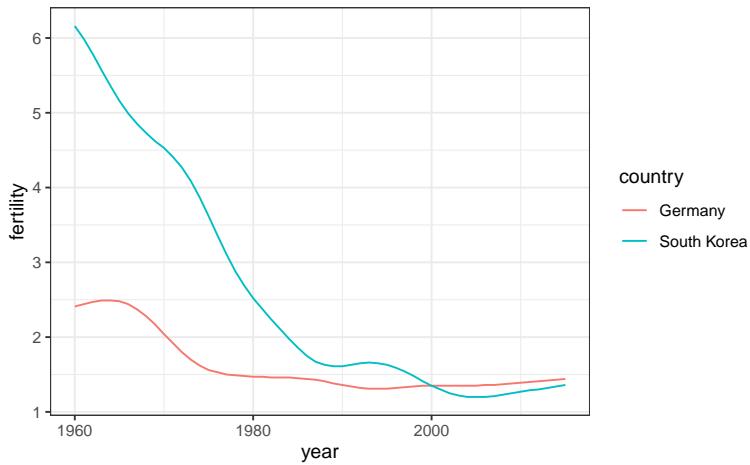
gapminder %>% filter(country %in% countries & !is.na(fertility)) %>%
  ggplot(aes(year, fertility, group = country)) +
  geom_line()
```



Mas qual linha representa cada país? Podemos atribuir cores para fazer essa distinção. Uma vantagem colateral do uso do argumento `color` para atribuir cores diferentes a diferentes países é que os dados são agrupados automaticamente:

```
countries <- c("South Korea", "Germany")

gapminder %>% filter(country %in% countries & !is.na(fertility)) %>%
  ggplot(aes(year, fertility, color = country)) +
  geom_line()
```



O gráfico mostra claramente como a taxa de fertilidade da Coreia do Sul caiu drasticamente durante as décadas de 1960 e 1970, e por volta de 1990 chegou a uma taxa semelhante à da Alemanha.

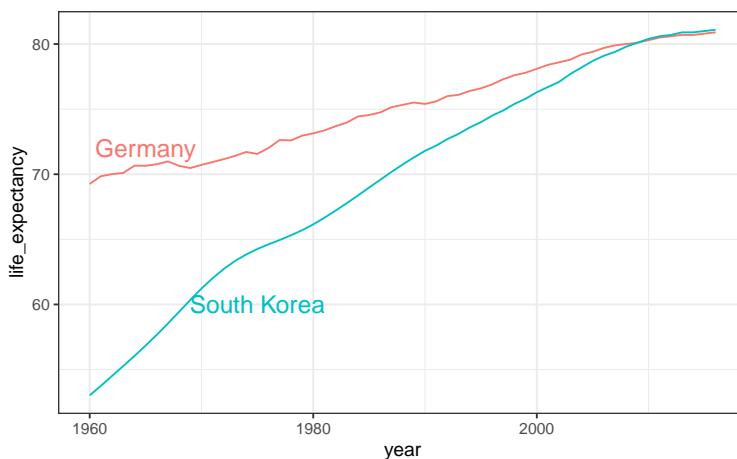
9.4.1 Rótulos em vez de legendas

Para gráficos de tendências, recomendamos rotular as linhas em vez de usar legendas. Assim, o espectador pode ver rapidamente qual linha representa qual país. Essa sugestão se aplica à maioria dos gráficos: rótulos geralmente são preferidos a legendas.

Demonstramos como fazer isso usando dados de expectativa de vida. Definimos uma tabela de dados com os locais dos rótulos e, em seguida, usamos uma segunda atribuição apenas para esses rótulos:

```
labels <- data.frame(country = countries, x = c(1975,1965), y = c(60,72))

gapminder %>%
filter(country %in% countries) %>%
ggplot(aes(year, life_expectancy, col = country)) +
geom_line() +
geom_text(data = labels, aes(x, y, label = country), size = 5) +
theme(legend.position = "none")
```



O gráfico mostra claramente como uma melhoria na expectativa de vida ocorreu em paralelo a quedas nas taxas de fertilidade. Em 1960, alemães viviam 15 anos a mais que sul-coreanos. Contudo, por volta de 2010, já não havia diferença. Isso exemplifica a melhoria que muitos países não ocidentais obtiveram nos últimos 40 anos.

9.5 Transformações de dados

Agora voltaremos nossa atenção para a segunda questão relacionada à ideia comum de que a distribuição da riqueza em todo o mundo piorou nas últimas décadas. Quando se pergunta ao público em geral se os países pobres se tornaram mais pobres e os países ricos se tornaram mais ricos, a maioria responde que sim. Usando estratificações, histogramas, gráficos de densidades e boxplots, podemos ver se isso é verdade. Primeiro, aprenderemos como as transformações às vezes podem ajudar a fornecer resumos e gráficos mais informativos.

A tabela de dados `gapminder` inclui uma coluna com o produto interno bruto (PIB) dos países. O PIB mede o valor de mercado de bens e serviços produzidos por um país em um ano. O PIB por pessoa é frequentemente usado como um resumo aproximado da riqueza de um país. Aqui, dividimos esse valor por 365 para obter uma medida mais interpretável: *dólares por dia*. Usando o dólar atual como unidade, uma pessoa que sobrevive com uma renda inferior a US\$2 por dia é definida como vivendo em *pobreza absoluta*. Vamos adicionar essa variável à tabela de dados:

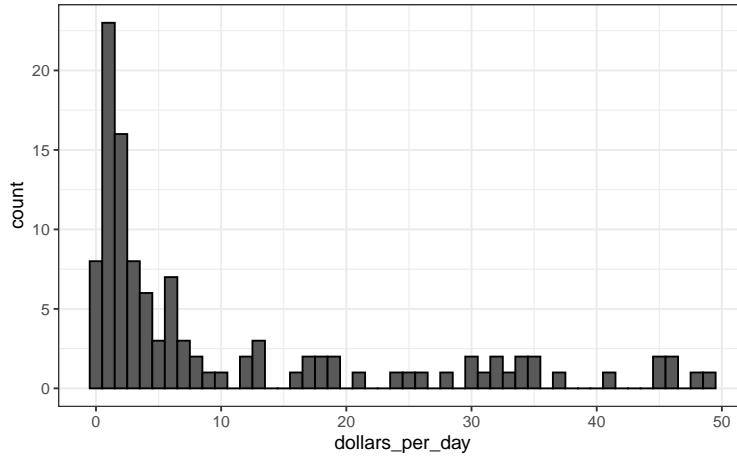
```
gapminder <- gapminder %>% mutate(dollars_per_day = gdp/population/365)
```

Os valores do PIB são ajustados pela inflação e representam o dólar atual, portanto esses valores devem ser comparáveis ao longo dos anos. Obviamente, essas são as médias dos países e dentro de cada país há muita variabilidade. Todos os gráficos e percepções descritos abaixo se referem às médias dos países e não aos indivíduos dentro deles.

9.5.1 Transformação logarítmica

Abaixo está um histograma de renda por dia em 1970:

```
past_year <- 1970
gapminder %>%
filter(year == past_year & !is.na(gdp)) %>%
ggplot(aes(dollars_per_day)) +
geom_histogram(binwidth = 1, color = "black")
```



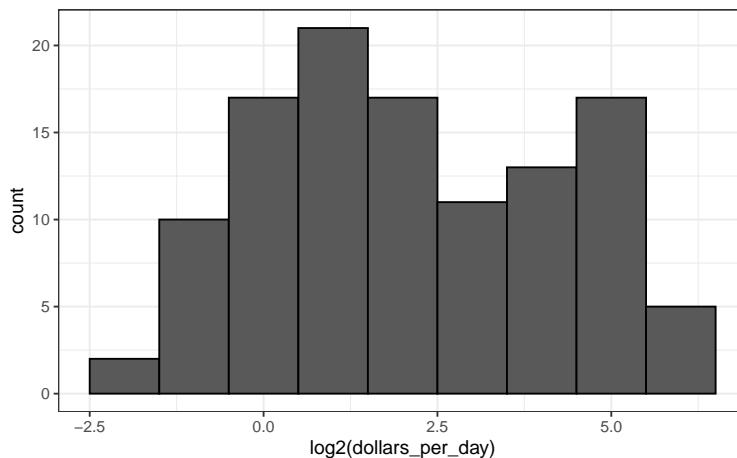
Usamos o argumento `color = "black"` para inserir bordas, e assim, distinguir claramente as barras.

Neste gráfico, vemos que, para a maioria dos países, as médias estão abaixo de \$10 por dia. Entretanto, a maioria do eixo x está dedicado aos 35 países com médias abaixo de \$10. Esse gráfico não é muito informativo em relação a esses países.

Seria mais informativo ver rapidamente quantos países têm renda média diária de cerca de \$1 (extremadamente pobre), \$2 (muito pobre), \$4 (pobre), \$8 (médio), \$16 (bem de vida), \$32 (rico), \$64 (muito rico) por dia. Como essas mudanças são multiplicativas, as transformações logarítmicas permitem converter as alterações multiplicativas em aditivas: quando a base 2 é usada, dobrar um valor se torna um aumento de 1.

Aqui está a distribuição se aplicarmos uma transformação logarítmica de base 2:

```
gapminder %>%
filter(year == past_year & !is.na(gdp)) %>%
ggplot(aes(log2(dollars_per_day))) +
geom_histogram(binwidth = 1, color = "black")
```



É assim que observamos mais de perto os países de renda média e baixa.

9.5.2 Qual base?

No caso anterior, usamos a base 2 nas transformações logarítmicas. Outras opções comuns são a base e (o logaritmo natural) e a base 10.

Em geral, não recomendamos o uso do logaritmo natural para exploração e visualização de dados. A razão disso é porque enquanto $2^2, 2^3, 2^4, \dots$ ou $10^2, 10^3, \dots$ são fáceis de calcular em nossas mentes, o mesmo não é verdade para e^2, e^3, \dots . Essa escala não é intuitiva nem fácil de interpretar.

No exemplo da renda em dólares por dia, usamos a base 2 em vez da base 10 porque o intervalo resultante é mais fácil de interpretar. O intervalo dos valores plotados é 0.327, 48.885.

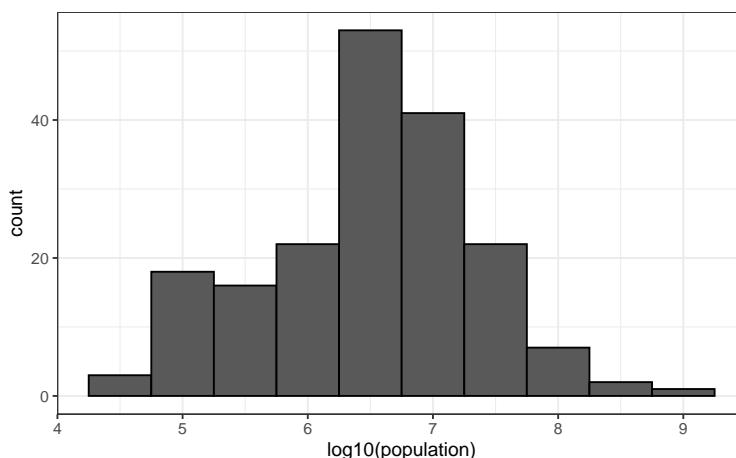
Na base 10, isso se torna um intervalo que inclui muito poucos números inteiros: apenas 0 e 1. Com a base dois, nosso intervalo inclui -2, -1, 0, 1, 2, 3, 4 e 5. É mais fácil de calcular 2^x e 10^x quando x é um número inteiro e está entre -10 e 10. Portanto, preferimos ter números inteiros menores na escala. Outra consequência de um intervalo limitado é que escolher a largura das barras (*binwidth* em inglês) é mais difícil. Com o log de base 2, sabemos que uma largura de barra igual a 1 será traduzida em uma barra com alcance de x a $2x$.

Para um exemplo em que a base 10 faz mais sentido, considere os tamanhos de populações. Um logaritmo de base 10 é preferível uma vez que o intervalo para esse caso é:

```
filter(gapminder, year == past_year) %>%
summarize(min = min(population), max = max(population))
#>      min      max
#> 1 46075 8.09e+08
```

Abaixo está o histograma dos valores transformados:

```
gapminder %>%
filter(year == past_year) %>%
ggplot(aes(log10(population))) +
geom_histogram(binwidth = 0.5, color = "black")
```



No gráfico acima, vemos rapidamente que as populações dos países variam entre dez mil e dez bilhões.

9.5.3 Transformar os valores ou a escala?

Existem duas maneiras de usar transformações logarítmicas em gráficos. Podemos pegar o logaritmo dos valores antes de representá-los graficamente ou usar escalas logarítmicas nos eixos. Ambas as abordagens são úteis e têm diferentes vantagens. Se transformarmos os dados em logaritmo, podemos interpretar mais facilmente os valores intermediários na escala. Por exemplo, se virmos:

----1----x----2-----3----

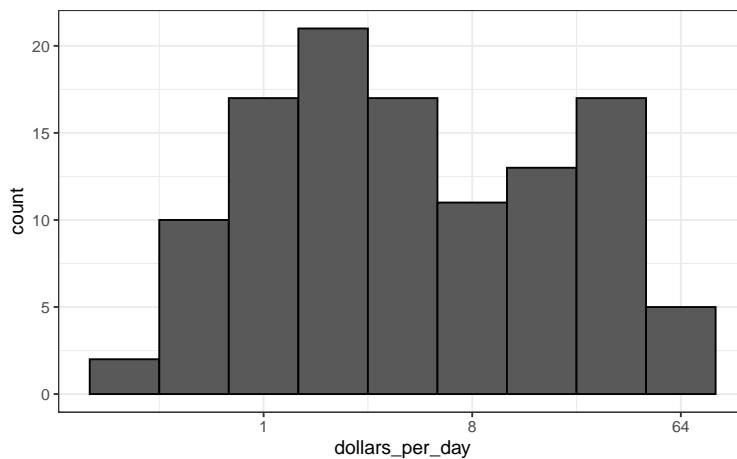
para dados transformados com o logaritmo, sabemos que o valor de x é aproximadamente 1,5. Se usarmos escalas logarítmicas:

----1----x----10-----100---

então, para determinar x precisamos calcular $10^{1.5}$, o que não é fácil de fazer mentalmente. A vantagem de usar escalas logarítmicas é que vemos os valores originais nos eixos. No entanto, a vantagem de apenas exibir essas escalas é que os valores originais são mostrados no gráfico e são mais fáceis de interpretar. Por exemplo, veríamos “32 dólares por dia” em vez de “5 log base 2 dólares por dia”.

Como aprendemos anteriormente, se quisermos dimensionar o eixo com logaritmos, podemos usar a função `scale_x_continuous`. Em vez de primeiro transformar os valores em logaritmo, aplicamos esta camada:

```
gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1, color = "black") +
  scale_x_continuous(trans = "log2")
```



Observe que a transformação logarítmica da base 10 tem sua própria função: `scale_x_log10()`, mas atualmente a base 2 não, embora possamos definir facilmente a nossa própria.

Outras transformações estão disponíveis através do argumento `trans`. Como aprenderemos mais adiante, a transformação da raiz quadrada (`sqrt`) é útil ao considerar contagens. A transformação logística (`logit`) é útil ao plotar proporções entre 0 e 1. E a transformação `reverse` é útil quando queremos que os menores valores estejam à direita ou acima.

9.6 Como visualizar distribuições multimodais

No histograma acima, vemos dois picos: um aos 4 e outro aos 32. Estatisticamente, esses picos às vezes são chamados de *modas*. A moda de uma distribuição é o valor com a frequência mais alta. A moda de uma distribuição normal é a média. Quando uma distribuição, como aquela apresentada acima, não diminui monotonicamente na moda, chamamos os locais onde ela sobe e desce novamente de *moda local* e dizemos que a distribuição tem *modas múltiplas*.

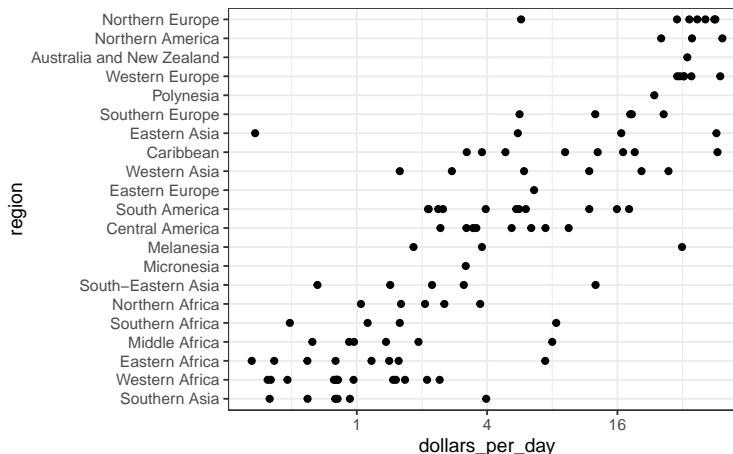
O histograma acima sugere que a distribuição de renda dos países em 1970 possui duas modas: uma de aproximadamente 2 dólares por dia (1 na escala log 2) e outro de aproximadamente 32 dólares por dia (5 na escala log 2). Essa *bimodalidade* é consistente com um mundo dicotômico composto por países com renda média inferior a \$8 (3 na escala log 2) por dia e países acima disso.

9.7 Como comparar múltiplas distribuições com *boxplots* e gráficos *ridge*

De acordo com o histograma, os valores da distribuição de renda para 1970 mostram uma dicotomia. No entanto, o histograma não nos mostra se os dois grupos de países são ocidentais ou parte do mundo em desenvolvimento.

Vamos começar analisando rapidamente os dados por região. Reorganizamos as regiões pela mediana e usamos uma escala logarítmica.

```
gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  mutate(region = reorder(region, dollars_per_day, FUN = median)) %>%
  ggplot(aes(dollars_per_day, region)) +
  geom_point() +
  scale_x_continuous(trans = "log2")
```



Já podemos ver que existe de fato uma dicotomia “ocidente versus o resto”: existem dois

grupos claros, com o rico grupo constituído pela América do Norte, Europa do Norte e Ocidental, Nova Zelândia e Austrália. Definimos grupos com base nesta observação:

```
gapminder <- gapminder %>%
  mutate(group = case_when(
    region %in% c("Western Europe", "Northern Europe", "Southern Europe",
      "Northern America",
      "Australia and New Zealand") ~ "West",
    region %in% c("Eastern Asia", "South-Eastern Asia") ~ "East Asia",
    region %in% c("Caribbean", "Central America",
      "South America") ~ "Latin America",
    continent == "Africa" &
    region != "Northern Africa" ~ "Sub-Saharan",
    TRUE ~ "Others"))
```

Convertemos essa variável `group` em um *factor* para controlar a ordem dos níveis:

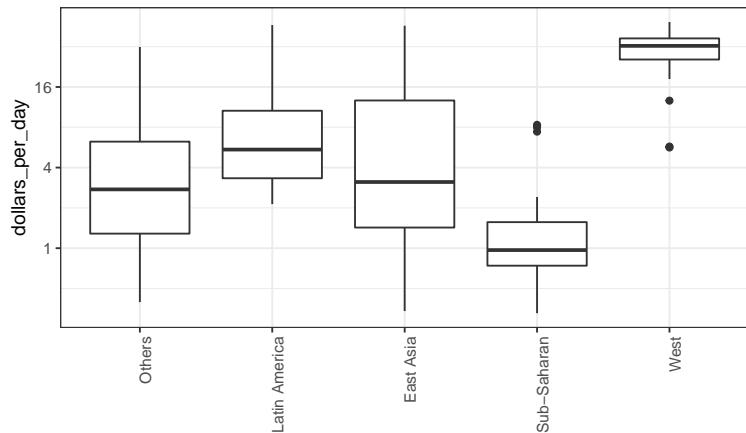
```
gapminder <- gapminder %>%
  mutate(group = factor(group, levels = c("Others", "Latin America",
    "East Asia", "Sub-Saharan",
    "West")))
```

Na próxima seção, mostraremos como visualizar e comparar distribuições entre grupos.

9.7.1 Boxplots (diagramas de caixa)

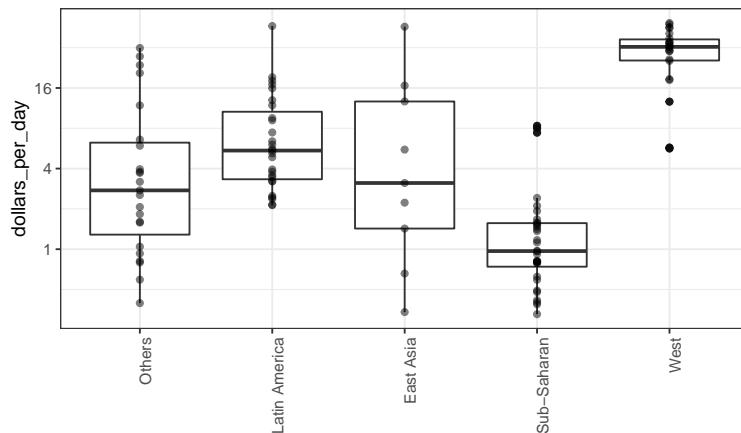
A análise exploratória dos dados acima revelou duas características sobre a distribuição de renda média em 1970. Usando um histograma, encontramos uma distribuição bimodal com modas relacionadas a países ricos e pobres. Agora, queremos comparar a distribuição entre esses cinco grupos para confirmar a dicotomia “oeste versus o resto”. O número de pontos em cada categoria é grande o suficiente para que um gráfico de resumo possa ser útil. Poderíamos gerar cinco histogramas ou cinco gráficos de densidade, mas pode ser mais prático ter todos os resumos visuais em um único gráfico. Portanto, vamos começar então colocando os boxplots lado a lado. Observe que adicionamos a camada `theme(axis.text.x = element_text(angle = 90, hjust = 1))` para colocar os rótulos dos grupos na vertical, uma vez que, se os mostrarmos horizontalmente, eles não irão caber no eixo. Assim, podemos ganhar espaço.

```
p <- gapminder %>%
  filter(year == past_year & !is.na(gdp)) %>%
  ggplot(aes(group, dollars_per_day)) +
  geom_boxplot() +
  scale_y_continuous(trans = "log2") +
  xlab("") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



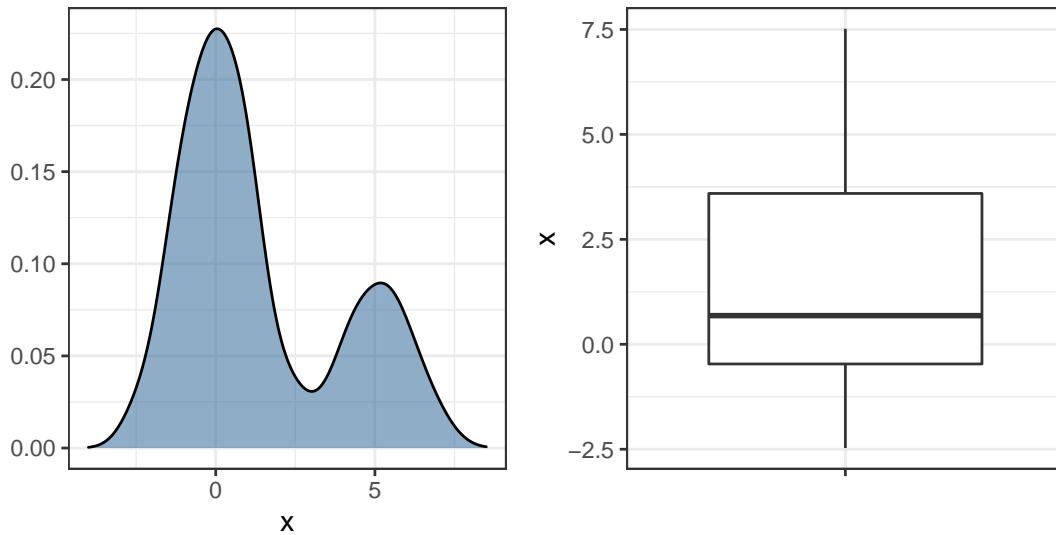
Os *boxplots* têm a limitação de que, resumindo os dados em cinco números, podemos perder importantes características dos dados. Uma maneira de evitar isso é exibindo os dados como pontos.

```
p + geom_point(alpha = 0.5)
```



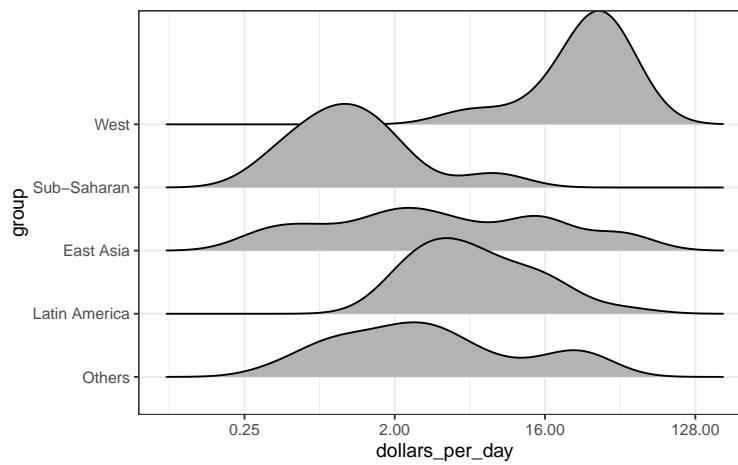
9.7.2 Gráficos *ridge*

Mostrar cada ponto individual nem sempre revela características importantes da distribuição. Embora esse não seja o caso aqui, quando o número de pontos representando dados é tão grande que há sobreposição, mostrar os dados pode ser contraproducente. Os *boxplots* ajudam nisso, fornecendo um resumo de cinco números, mas isso também tem suas limitações. Por exemplo, *boxplots* não revelam distribuições bimodais. Para ver isso, observe os dois gráficos abaixo que resumem o mesmo conjunto de dados:



Nos casos em que estamos preocupados que o resumo do *boxplot* seja muito simplista, podemos exibir densidades suaves ou histogramas empilhados usando gráficos *ridge*. Como estamos acostumados a visualizar densidades com valores no eixo x, as empilhamos verticalmente. Além disso, como precisamos de mais espaço nessa abordagem, é conveniente permitir sobreposições. O pacote **ggridges** inclui uma função conveniente para fazer isso. Abaixo estão os dados da receita, mostrados acima com gráficos de caixa, mas agora exibidos com uma crista gráfica _.

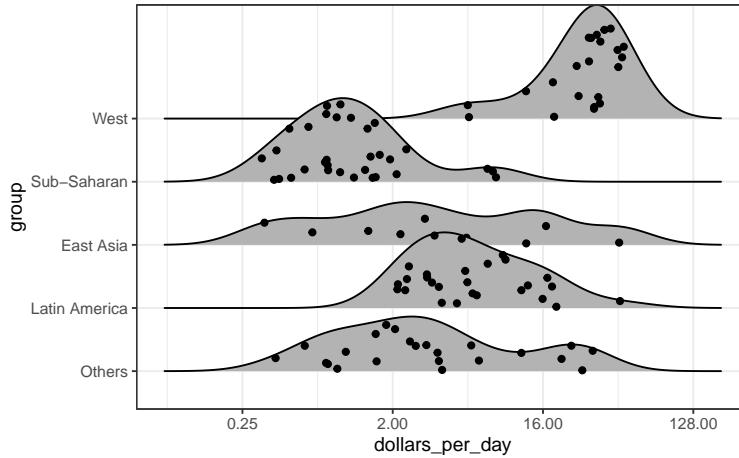
```
library(ggridges)
p <- gapminder %>%
  filter(year == past_year & !is.na(dollars_per_day)) %>%
  ggplot(aes(dollars_per_day, group)) +
  scale_x_continuous(trans = "log2")
p + geom_density_ridges()
```



Note de que precisamos inverter os valores de x e y que foram usados para o *boxplot*. Um parâmetro útil para `geom_density_ridges` é `scale`, que permite determinar quanto os gráficos irão se sobrepor. Por exemplo, `scale = 1` significa que não há sobreposição. Valores maiores que 1 resultam em maior sobreposição.

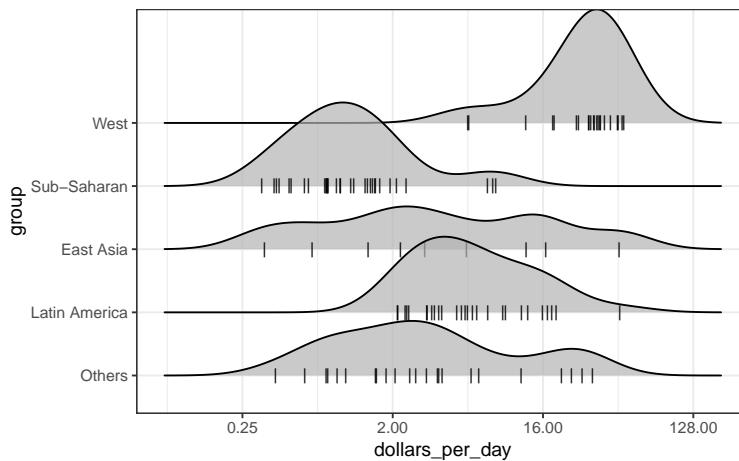
Se o número de pontos de dados for pequeno o suficiente, podemos adicioná-los ao gráfico *ridge* usando o seguinte código:

```
p + geom_density_ridges(jittered_points = TRUE)
```



Por padrão, a altura dos pontos é instável (*jittered*) e não deve ser interpretada de forma alguma. Para exibir pontos de dados, mas sem usar *jitter*, podemos usar o código a seguir para adicionar o que é conhecido como uma representação *rug* dos dados.

```
p + geom_density_ridges(jittered_points = TRUE,
position = position_points_jitter(height = 0),
point_shape = '|', point_size = 3,
point_alpha = 1, alpha = 0.7)
```

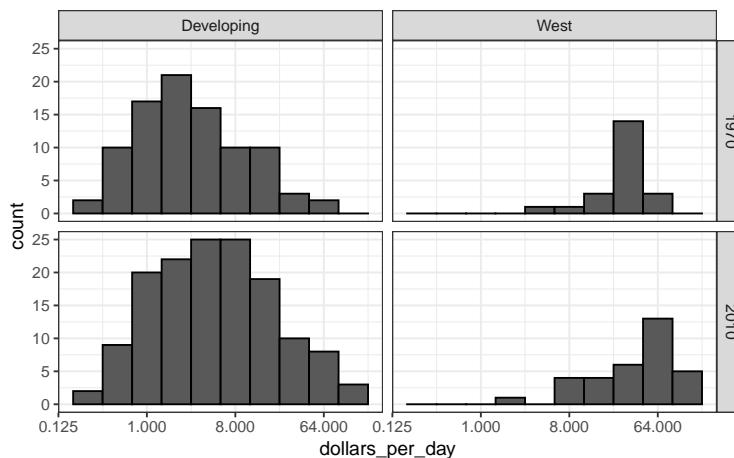


9.7.3 Exemplo: distribuições de renda 1970 versus 2010

A exploração dos dados mostra claramente que em 1970 havia uma dicotomia “ocidente versus o resto”. Mas essa dicotomia persiste? Vamos usar `facet_grid` para ver como as distribuições mudaram. Para começar, focamos em dois grupos: o ocidente e o resto. Vamos fazer quatro histogramas:

```
past_year <- 1970
present_year <- 2010
```

```
years <- c(past_year, present_year)
gapminder %>%
  filter(year %in% years & !is.na(gdp)) %>%
  mutate(west = ifelse(group == "West", "West", "Developing")) %>%
  ggplot(aes(dollars_per_day)) +
  geom_histogram(binwidth = 1, color = "black") +
  scale_x_continuous(trans = "log2") +
  facet_grid(year ~ west)
```



Antes de interpretar as conclusões deste gráfico, notamos que há mais países representados nos histogramas de 2010 do que em 1970: a contagem total é maior. Uma razão para isso é que vários países foram fundados após 1970. Por exemplo, a União Soviética foi dividida em diferentes países durante os anos 1990. Uma outra razão é que mais dados estão disponíveis para mais países em 2010.

Refizemos os gráficos usando apenas países com dados disponíveis para ambos os anos. Na seção *data wrangling* deste livro, aprenderemos a usar as ferramentas **tidyverse** que nos permitirão escrever códigos eficiente para isso. Entretanto, aqui iremos usar um simples código usando a função `intersect`:

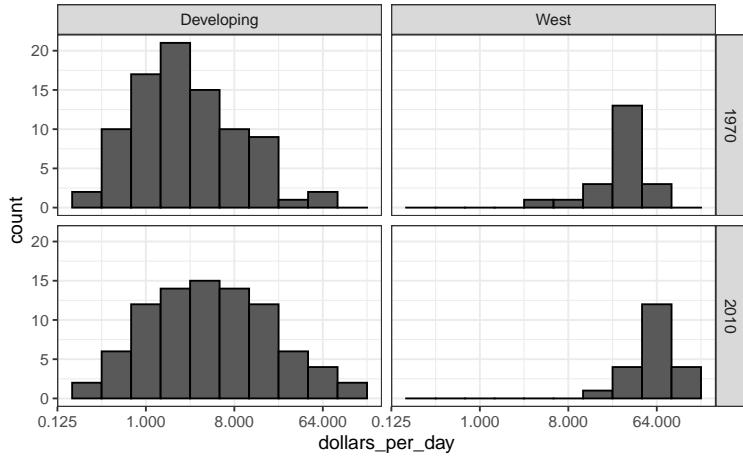
```
country_list_1 <- gapminder %>%
  filter(year == past_year & !is.na(dollars_per_day)) %>%
  pull(country)

country_list_2 <- gapminder %>%
  filter(year == present_year & !is.na(dollars_per_day)) %>%
  pull(country)

country_list <- intersect(country_list_1, country_list_2)
```

Estes 108 representam 86% da população mundial, portanto esse subconjunto deve ser representativo.

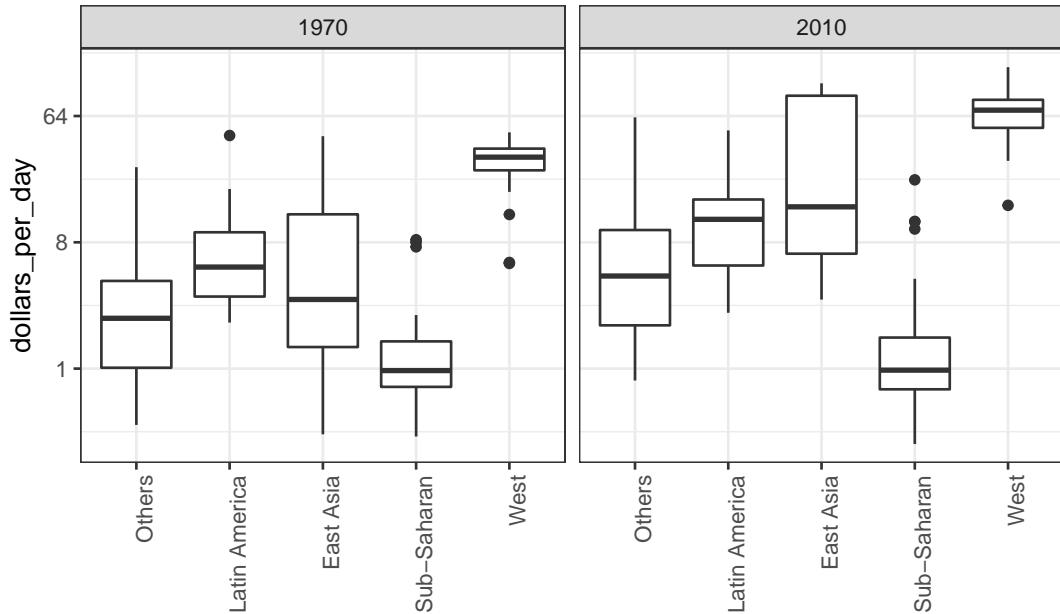
Vamos refazer o gráfico apenas para esse subconjunto, simplesmente adicionando `country %in% country_list` para a função `filter`:



Agora vemos que os países ricos ficaram um pouco mais ricos, mas em termos de porcentagem, os países pobres parecem ter melhorado mais. Em particular, vemos que a proporção de países em desenvolvimento que ganha mais de 16 dólares por dia aumentou substancialmente.

Para ver quais regiões específicas melhoraram mais, podemos refazer os *boxplots* construídos anteriormente, mas agora adicionamos o ano de 2010, e então, usamos *facet* para comparar os dois anos.

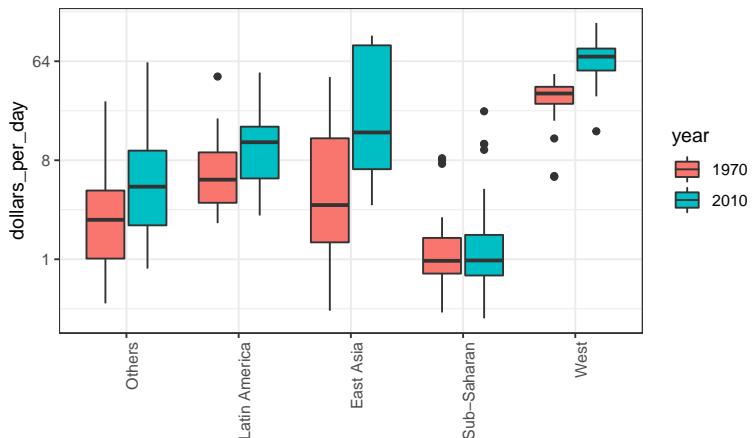
```
gapminder %>%
filter(year %in% years & country %in% country_list) %>%
ggplot(aes(group, dollars_per_day)) +
geom_boxplot() +
theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
scale_y_continuous(trans = "log2") +
xlab("") +
facet_grid(. ~ year)
```



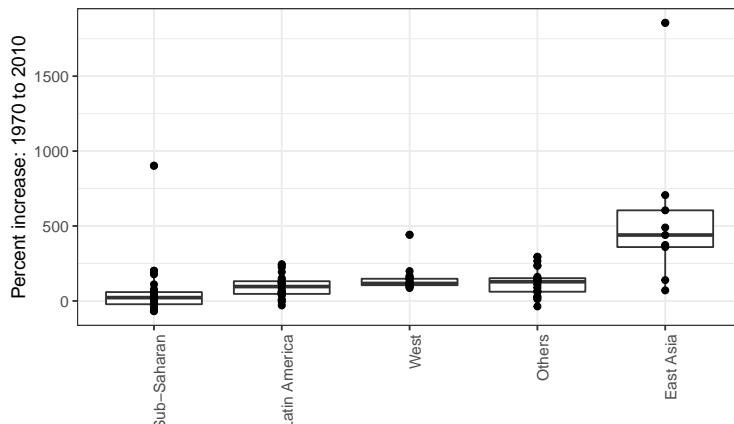
Agora faremos uma pausa para introduzir outro recurso importante do **ggplot2**. Como queremos comparar cada região antes e depois, seria conveniente ter o *boxplot* de 1970 próximo ao de 2010 para cada região. Em geral, as comparações são mais fáceis quando os dados são plotados um ao lado do outro.

Portanto, em vez de separarmos em facetas, manteremos os dados de cada ano juntos e iremos colori-los dependendo do ano. Observe que os grupos são separados automaticamente por ano e cada par de *boxplot* é desenhado lado a lado. Como o ano é um número, nós o converteremos em um *factor*, uma vez que **ggplot2** atribui automaticamente uma cor a cada categoria de um *factor*. Lembre-se que temos que converter as colunas *year* de *numeric* para *factor*.

```
gapminder %>%
  filter(year %in% years & country %in% country_list) %>%
  mutate(year = factor(year)) %>%
  ggplot(aes(group, dollars_per_day, fill = year)) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  scale_y_continuous(trans = "log2") +
  xlab("")
```



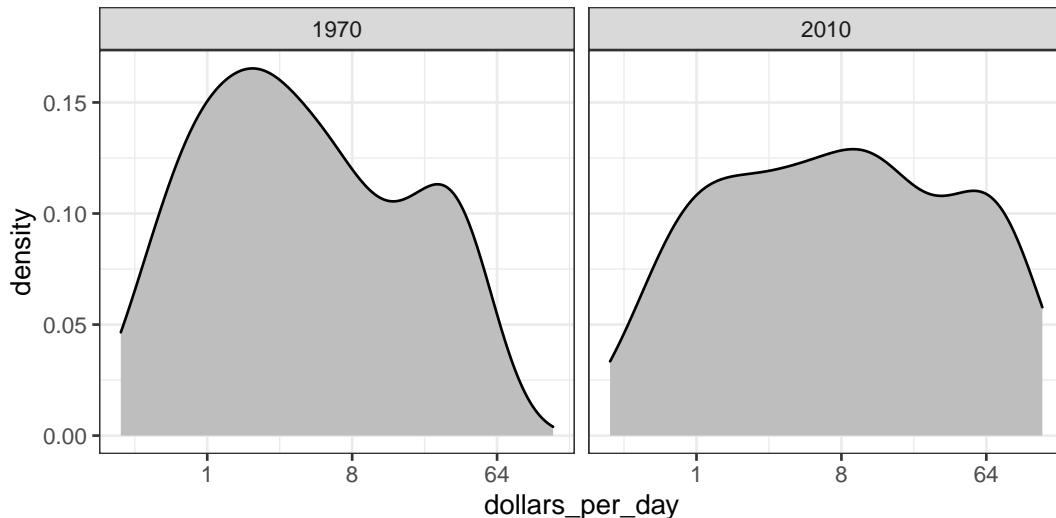
Finalmente, destacamos que, se estivermos mais interessados em comparar os valores antes e depois, pode fazer mais sentido traçar aumentos percentuais. Ainda não estamos prontos para aprender como codificar isso, mas é assim que o gráfico seria:



A exploração de dados anterior sugere que a diferença de renda entre países ricos e pobres diminuiu consideravelmente nos últimos 40 anos. Usamos uma série de histogramas e *boxplots* para ver isso. Sugerimos uma maneira sucinta de transmitir essa mensagem com apenas um gráfico.

Vamos começar observando que os gráficos de densidade para a distribuição de renda em 1970 e 2010 transmitem a mensagem de que a diferença está diminuindo:

```
gapminder %>%
filter(year %in% years & country %in% country_list) %>%
ggplot(aes(dollars_per_day)) +
geom_density(fill = "grey") +
scale_x_continuous(trans = "log2") +
facet_grid(. ~ year)
```



No gráfico de 1970 vemos duas tendências claras: países pobres e ricos. No 2010 alguns dos países pobres parecem ter mudado para a direita, diminuindo a diferença.

A próxima mensagem que devemos transmitir é que a razão para essa mudança na distribuição é que vários países pobres ficaram mais ricos do que alguns países ricos ficaram mais pobres. Para fazer isso, podemos atribuir uma cor aos grupos que identificamos durante a exploração de dados.

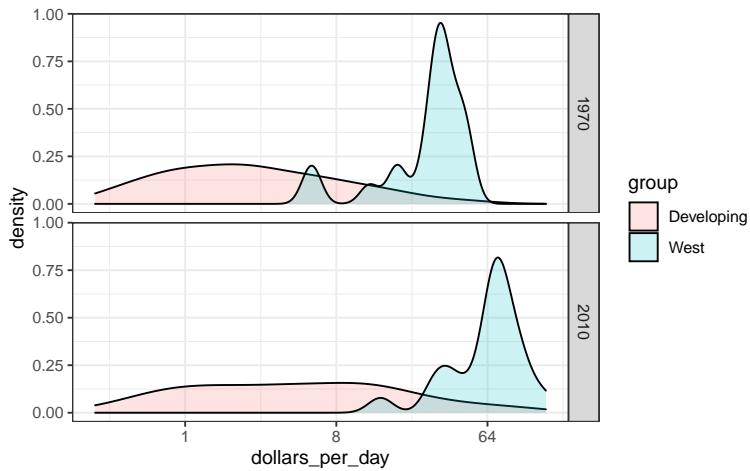
No entanto, precisamos primeiro aprender a suavizar essas densidades de maneira a preservar informações sobre o número de países em cada grupo. Para entender por que precisamos disso, lembre-se da discrepância no tamanho de cada grupo:

```
#> `summarise()` ungrouping output (override with ` .groups` argument)
```

Developing	West
87	21

Porém, quando sobreponemos duas densidades, o comportamento padrão é que a área representada por cada distribuição some 1, independentemente do tamanho de cada grupo:

```
gapminder %>%
  filter(year %in% years & country %in% country_list) %>%
  mutate(group = ifelse(group == "West", "West", "Developing")) %>%
  ggplot(aes(dollars_per_day, fill = group)) +
  scale_x_continuous(trans = "log2") +
  geom_density(alpha = 0.2) +
  facet_grid(year ~ .)
```



O gráfico acima mostra que há o mesmo número de países em cada grupo. Para mudar isso, precisaremos aprender como acessar as variáveis calculadas com a função `geom_density`.

9.7.4 Como acessar variáveis calculadas

Para tornar as áreas dessas densidades proporcionais ao tamanho do grupo, simplesmente multiplicamos os valores do eixo y pelo tamanho do grupo. No arquivo de ajuda de `geom_density`, vemos que as funções calculam uma variável chamada `count` que faz exatamente isso. Queremos que essa variável, e não a densidade, esteja no eixo y.

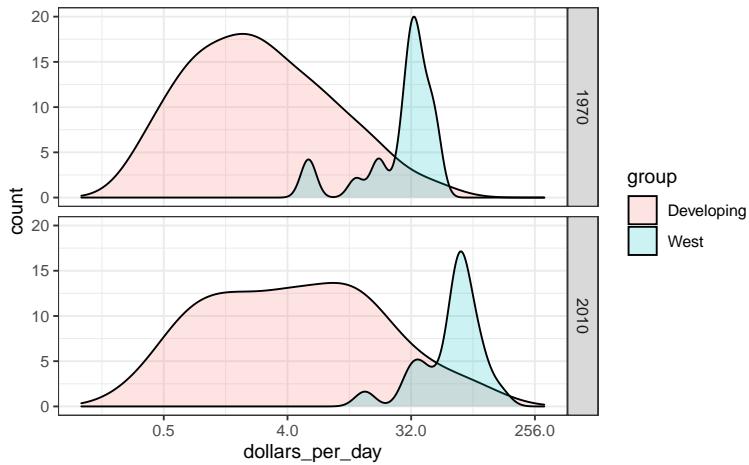
Em `ggplot2`, obtemos acesso a essas variáveis cercando o nome com dois pontos. Portanto, usaremos o seguinte mapeamento:

```
aes(x = dollars_per_day, y = ..count..)
```

Agora podemos criar o gráfico desejado simplesmente alterando o mapeamento do código anterior. Também estenderemos os limites do eixo x.

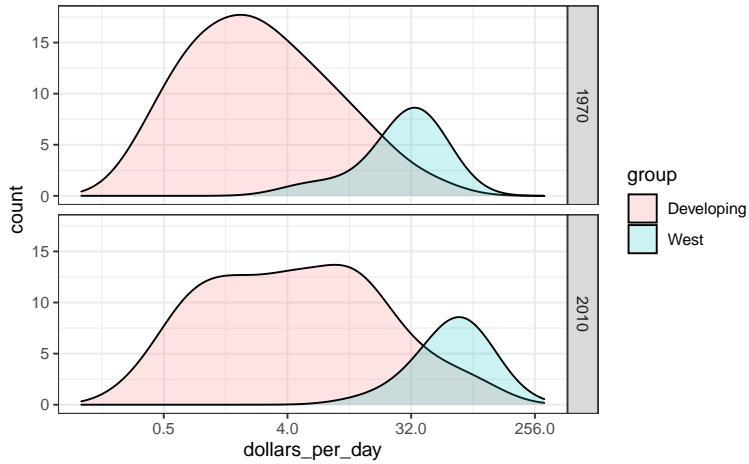
```
p <- gapminder %>%
  filter(year %in% years & country %in% country_list) %>%
  mutate(group = ifelse(group == "West", "West", "Developing")) %>%
  ggplot(aes(dollars_per_day, y = ..count.., fill = group)) +
  scale_x_continuous(trans = "log2", limit = c(0.125, 300))

p + geom_density(alpha = 0.2) +
  facet_grid(year ~ .)
```



Se queremos que as densidades sejam mais suaves, usamos o argumento `bw` para que o mesmo parâmetro de suavização seja usado para cada densidade. Selecioneamos 0,75 após testar vários valores.

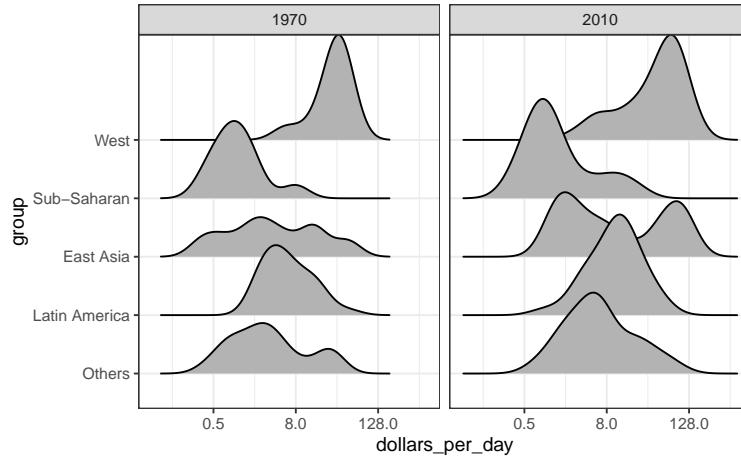
```
p + geom_density(alpha = 0.2, bw = 0.75) + facet_grid(year ~ .)
```



Esse gráfico agora mostra o que está acontecendo muito claramente. A distribuição do mundo em desenvolvimento está mudando. Uma terceira modalidade aparece, apresentando os países que mais diminuíram a diferença.

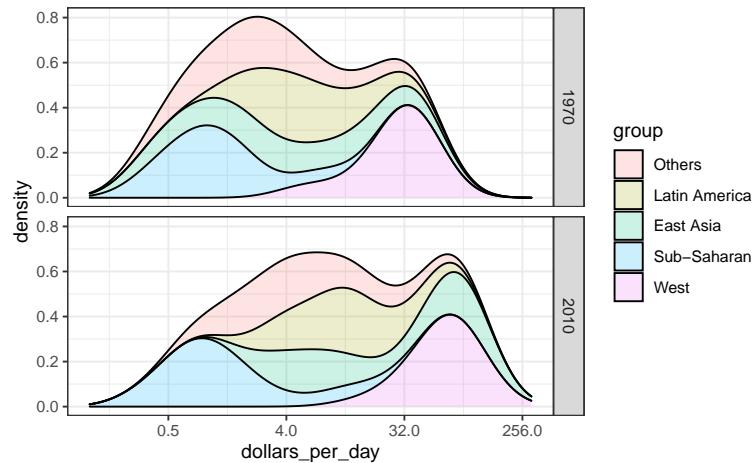
Para visualizar se algum dos grupos definidos acima é a principal causa dessas alterações, podemos criar rapidamente um gráfico *ridge*:

```
gapminder %>%
  filter(year %in% years & !is.na(dollars_per_day)) %>%
  ggplot(aes(dollars_per_day, group)) +
  scale_x_continuous(trans = "log2") +
  geom_density_ridges(adjust = 1.5) +
  facet_grid(. ~ year)
```



Outra maneira de conseguir isso é empilhando as densidades umas sobre as outras:

```
gapminder %>%
  filter(year %in% years & country %in% country_list) %>%
  group_by(year) %>%
  mutate(weight = population/sum(population)*2) %>%
  ungroup() %>%
  ggplot(aes(dollars_per_day, fill = group)) +
  scale_x_continuous(trans = "log2", limit = c(0.125, 300)) +
  geom_density(alpha = 0.2, bw = 0.75, position = "stack") +
  facet_grid(year ~ .)
```

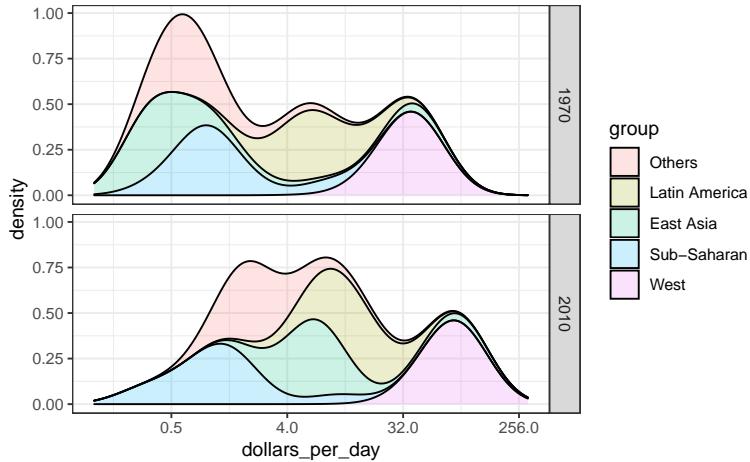


Aqui podemos ver claramente como as distribuições para o Leste da Ásia, América Latina e outros se deslocam visivelmente para a direita. Enquanto a África Subsaariana permanece estagnada.

Observe que ordenamos os níveis do grupo para que a densidade do Ocidente seja plotada primeiro, seguida pela África Subsaariana. Ter ambas as extremidades representadas graficamente primeiro nos permite ver melhor a bimodalidade restante.

9.7.5 Densidades ponderadas

Como ponto final, notamos que essas distribuições têm o mesmo peso para cada país. Portanto, se a maioria da população está melhorando, mas se tratando em um país muito grande, como a China, talvez não avaliemos isso. De fato, podemos ponderar as densidades suaves usando o argumento de mapeamento `weight`. O gráfico então ficaria assim:



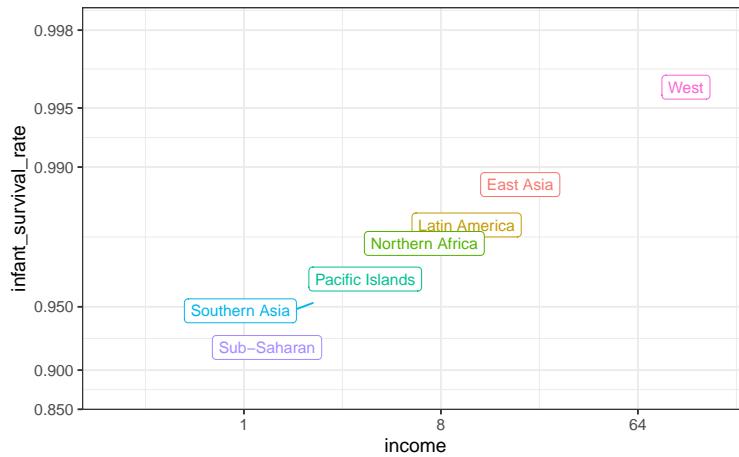
Essa figura em particular mostra muito claramente como a diferença na distribuição de renda está diminuindo e que a maioria dos países ainda em situação de pobreza está na África Subsaariana.

9.8 A falácia ecológica e a importância de mostrar os dados

Ao longo desta seção, comparamos regiões do mundo. Vimos que, em média, algumas regiões têm melhor desempenho que outras. Nesta seção, vamos nos concentrar na descrição da importância da variabilidade dentro dos grupos, examinando a relação entre as taxas de mortalidade infantil de um país e a renda média.

Vamos definir mais algumas regiões e comparar as médias entre elas:

```
#> `summarise()`` ungrouping output (override with `groups` argument)
```



A relação entre essas duas variáveis é quase perfeitamente linear e o gráfico mostra uma diferença dramática. Enquanto menos de 0,5% dos bebês morrem no Ocidente, na África Subsaariana a taxa é superior a 6%!

Observe que o gráfico usa uma nova transformação, a transformação logística.

9.8.1 Transformação logística

Transformação logística ou *logit* para uma proporção ou taxa p é definida como:

$$f(p) = \log\left(\frac{p}{1-p}\right)$$

Quando p é uma proporção ou probabilidade, a quantidade que transformamos com o logaritmo, $p/(1-p)$, é chamado de *chance*. Neste caso p é a proporção de bebês que sobreviveram. As chances nos dizem quantos bebês a mais devem sobreviver do que morrer. A transformação logarítmica torna isso simétrico. Se as taxas forem iguais, o *log* das chances será 0. Aumentos multiplicativos são convertidos em aumentos positivos ou negativos, respectivamente.

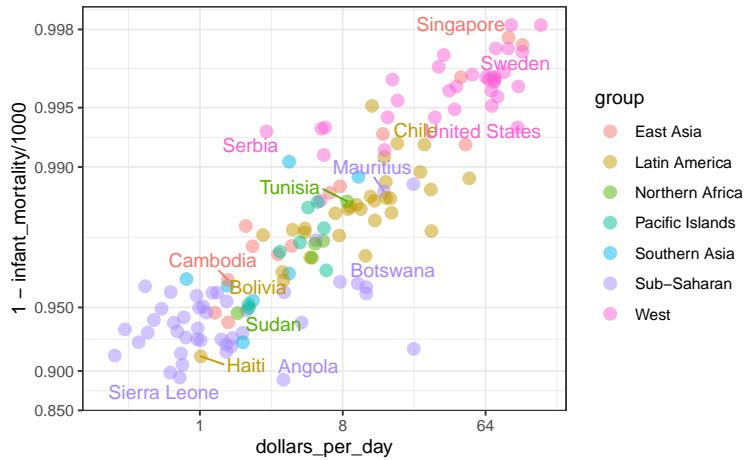
Essa escala é útil quando queremos destacar diferenças próximas de 0 ou 1. Para taxas de sobrevivência, isso é importante porque uma taxa de sobrevivência de 90% é inaceitável, enquanto uma sobrevivência de 99% é relativamente boa. Preferiríamos muito uma taxa de sobrevivência próxima de 99,9%. Queremos que nossa escala destaque essas diferenças e o *logit* o faz. Lembre-se de que $99,9/0,1$ é aproximadamente 10 vezes maior que $99/1$, que é aproximadamente 10 vezes maior que $90/10$. Usando o logaritmo, esses incrementos multiplicativos são convertidos em aumentos constantes.

9.8.2 Mostre os dados

Agora, de volta ao nosso gráfico. Com base no gráfico acima, podemos concluir que um país de baixa renda está destinado a ter uma baixa taxa de sobrevivência? Além disso, podemos concluir que as taxas de sobrevivência na África Subsaariana são mais baixas do que no Sul da Ásia, que por sua vez, são mais baixas do que nas ilhas do Pacífico e assim por diante?

Ir direto para essa conclusão com base em um gráfico que mostra as médias é conhecido como *falácia ecológica*. A relação quase perfeita entre taxas de sobrevivência e renda é observada

apenas para médias regionais. Depois de mostrar todos os dados, vemos uma história um pouco mais complicada:



Especificamente, vemos que há muita variabilidade. Vemos que os países nas mesmas regiões podem ser bem diferentes e que os países com a mesma renda podem ter taxas de sobre-vivência diferentes. Por exemplo, enquanto a África Subsaariana teve os piores resultados econômicos e de saúde em média, há uma grande variabilidade dentro desse grupo. Maurício e Botsuana são melhores que Angola e Serra Leoa, com Maurício comparável aos países ocidentais.

10

Princípios de visualização de dados

Já definimos algumas regras a serem seguidas ao criar gráficos para nossos exemplos. Aqui, nosso objetivo é oferecer alguns princípios gerais que podemos usar como um guia para uma visualização eficaz dos dados. Grande parte desta seção é baseada em uma palestra de Karl Broman¹ intitulada “Criando figuras e tabelas eficazes”² e inclui algumas das figuras que foram criadas com códigos que Karl disponibiliza em seu repositório GitHub³. Também nos baseamos em anotações das aulas do curso “Introdução à visualização de dados” de Peter Aldhous⁴. Seguindo a abordagem de Karl, mostramos alguns exemplos de estilos de gráficos que devem ser evitados, explicamos como melhorá-los e depois os usamos como motivação para uma lista de princípios. Além disso, comparamos e contrastamos gráficos que seguem esses princípios com outros que os ignoram.

Os princípios são baseados principalmente em pesquisas relacionadas à forma como os seres humanos detectam padrões e fazem comparações visuais. As abordagens preferidas são as que melhor se adaptam à maneira como nossos cérebros processam as informações visuais. Ao escolher as ferramentas de visualização, é importante ter em mente nosso objetivo. Podemos comparar um número suficientemente pequeno de números que podem ser distinguidos, descrevendo distribuições de dados categóricos ou valores numéricos, comparando os dados de dois grupos ou descrevendo a relação entre duas variáveis e isso afeta a apresentação que escolheremos. Como observação final, queremos enfatizar que é importante para cientistas de dados adaptar e otimizar gráficos para o público. Por exemplo, um gráfico exploratório feito para nós mesmos será diferente de um gráfico destinado a comunicação de uma descoberta a um público em geral.

Vamos usar estas bibliotecas:

```
library(tidyverse)
library(dslabs)
library(gridExtra)
```

10.1 Codificando dados usando dicas visuais

Vamos começar descrevendo alguns princípios para codificar dados. Existem várias abordagens à nossa disposição, incluindo posição, comprimento, ângulos, área, brilho e cor.

Para ilustrar como algumas dessas estratégias se comparam, vamos supor que desejamos relatar os resultados de duas pesquisas hipotéticas relacionadas à preferência por navegador

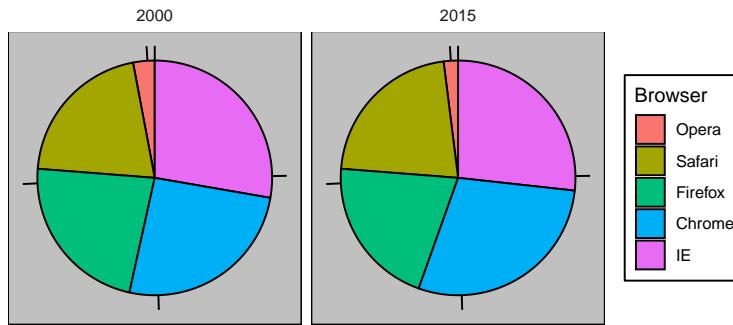
¹<http://kbroman.org/>

²<https://www.biostat.wisc.edu/~kbroman/presentations/graphs2017.pdf>

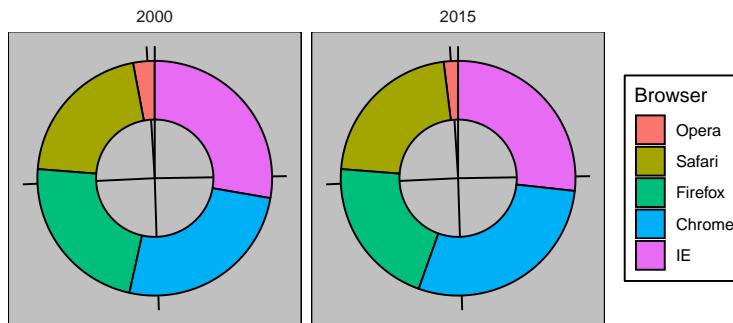
³https://github.com/kbroman/Talk_Graphs

⁴<http://paldhous.github.io/ucb/2016/dataviz/index.html>

de Internet (*browser*), realizadas em 2000 e 2015, respectivamente. Para cada ano, estamos simplesmente comparando cinco quantidades: as cinco porcentagens. Uma representação gráfica de porcentagens amplamente usada e popularizada pelo Microsoft Excel é o gráfico de pizza:



Aqui, estamos representando as quantidades através das áreas e dos ângulos, uma vez que o ângulo e a área de cada seção do gráfico são proporcionais à quantidade que o setor representa. Isso acaba sendo uma opção abaixo do ideal, pois, como demonstrado por estudos perceptivos, os seres humanos não são bons em quantificar com precisão ângulos e são ainda piores quando a área é o único sinal visual disponível. O gráfico de rosca é um exemplo de gráfico que usa apenas a área:



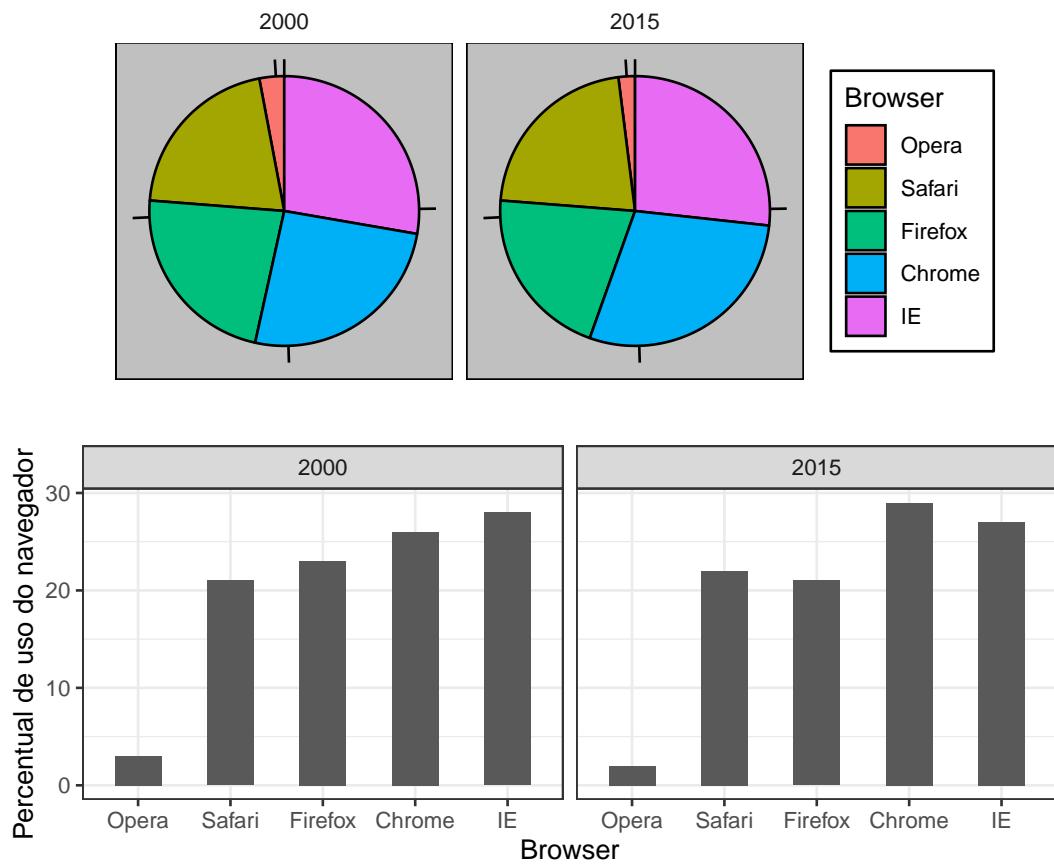
Para ver como é difícil quantificar ângulos e áreas, note que as classificações e todas as porcentagens nos gráficos acima foram alteradas entre 2000 e 2015. Você pode determinar as porcentagens reais e classificar a popularidade dos navegadores? Você pode ver como as porcentagens mudaram de 2000 para 2015? Não é fácil identificar isso através desse gráfico. De fato, a função `pie` da página de ajuda de R afirma que:

Os gráficos de pizza são uma forma ruim de exibir informações. O olho humano é bom em julgar medições lineares, mas ruim em julgar áreas relativas. Gráficos de barras ou de pontos são maneiras preferíveis de exibir esse tipo de dados.

Nesse caso, simplesmente exibir os números não é apenas mais claro, mas também reduziria custos caso você queira imprimir uma cópia em papel:

Browser	2000	2015
Opera	3	2
Safari	21	22
Firefox	23	21
Chrome	26	29
IE	28	27

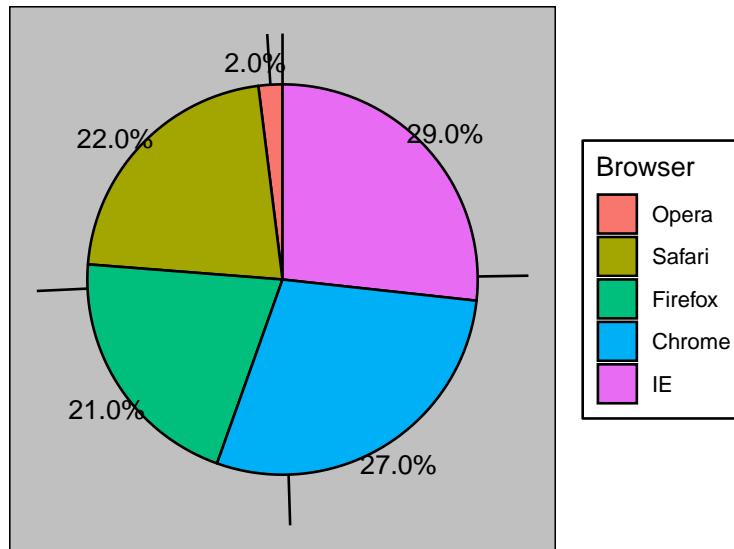
Uma maneira preferivelmente melhor de representar graficamente essas quantidades é usar comprimento e posição como pistas visuais, uma vez que os humanos são muito melhores em julgar medições lineares. O gráfico de barras usa essa abordagem usando barras de comprimento proporcionais às quantias de interesse. Ao adicionar linhas horizontais a valores estratégicamente escolhidos, neste caso a cada múltiplo de 10, amenizamos a carga visual permitindo a identificação de quantidades através da posição da parte superior das barras. Compare e contraste as informações que podemos extrair das duas figuras abaixo.



Observe como é fácil ver as diferenças no gráfico de barras. De fato, agora podemos determinar as porcentagens reais seguindo as linhas horizontais ao eixo x.

Se, por algum motivo, você precisar criar um gráfico de pizza, rotule cada seção do círculo com sua respectiva porcentagem para que o público não precise inferi-las a partir dos ângulos ou da área:

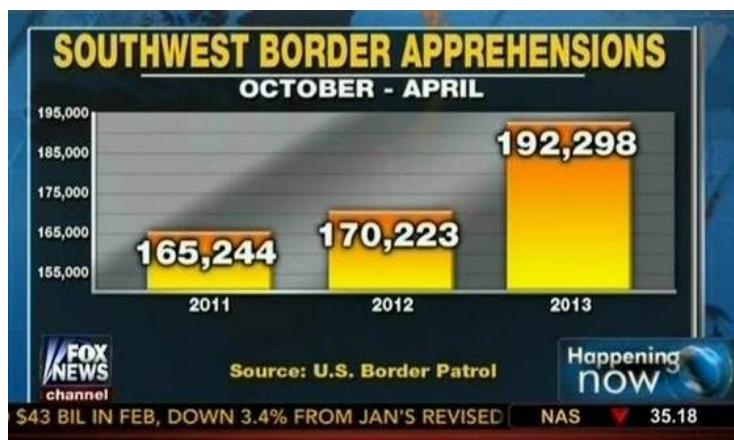
2015



Em geral, ao exibir quantidades, posições e comprimentos são preferíveis do que exibir ângulos e/ou áreas. Além disso, brilho e cor são ainda mais difíceis de quantificar do que os ângulos. Mas, como veremos mais adiante, às vezes são úteis quando mais de duas dimensões devem ser exibidas ao mesmo tempo.

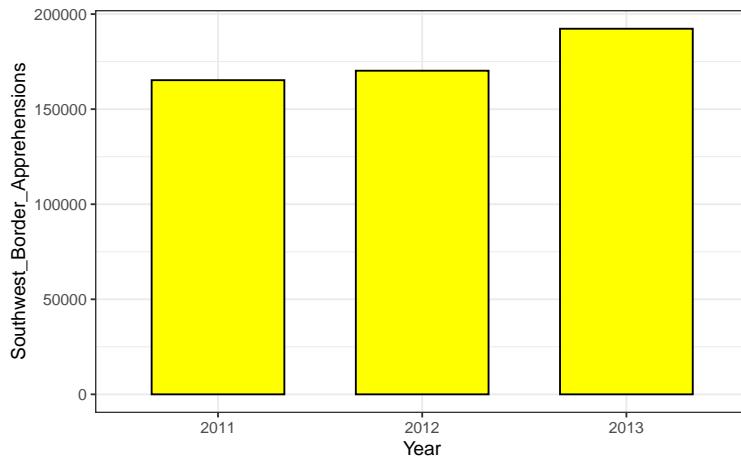
10.2 Saiba quando incluir 0

Ao usar gráficos de barras, é considerado desinformativo não iniciar as barras em 0. Isso ocorre porque, ao usar um gráfico de barras, estamos sugerindo que o comprimento é proporcional às quantidades mostradas. Evitando iniciar em 0, você pode fazer com que diferenças relativamente pequenas pareçam muito maiores do que realmente são. Essa abordagem é frequentemente usada por políticos ou organizações midiáticas tentando exagerar a diferença. Abaixo está um exemplo ilustrativo usado por Peter Aldhous nesta palestra: <http://paldhous.github.io/ucb/2016/dataviz/week2.html>.

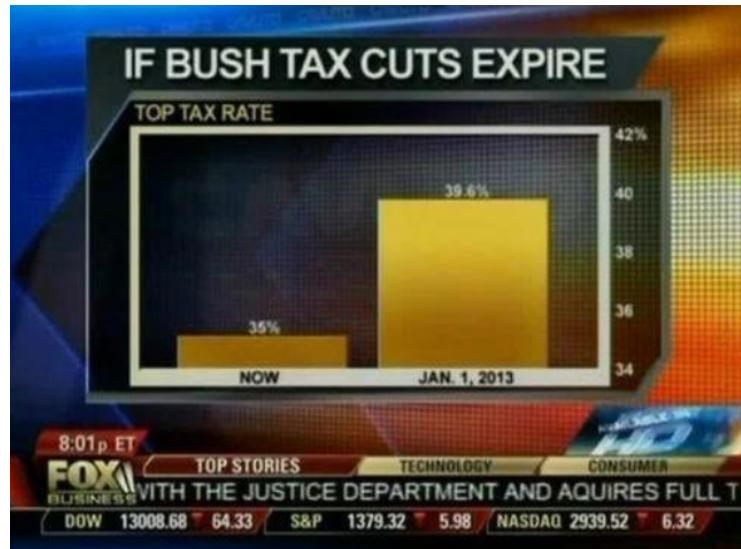


(Detenções nas fronteiras dos Estados Unidos entre 2011 e 2013. Fonte: Fox News, via Media Matters⁵.)

No gráfico acima, as detenções parecem ter triplicado quando, de fato, elas aumentaram apenas cerca de 16%. Iniciar o gráfico em 0 ilustra isso claramente:



Abaixo, vemos outro exemplo, que é descrito em detalhes em um artigo do blog “Flowing Data”:



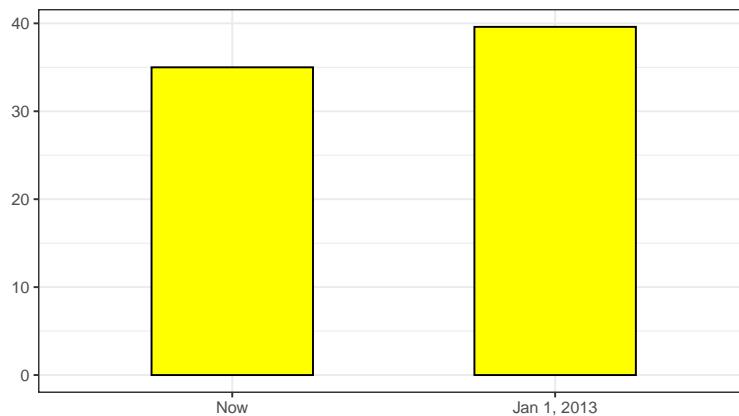
(Previsão do aumento de taxas nos Estados Unidos se o corte de impostos determinado no governo do presidente Bush expirar. Fonte: Fox News, através da Flowing Data⁶)

Este gráfico faz um aumento de 13% parecer cinco vezes maior. Aqui está um gráfico mais apropriado:

⁵<http://mediamatters.org/blog/2013/04/05/fox-news-newest-dishonest-chart-immigration-enf/193507>

⁶<http://flowingdata.com/2012/08/06/fox-news-continues-charting-excellence/>

Aumento nas taxas se o corte de impostos de Bush expirar

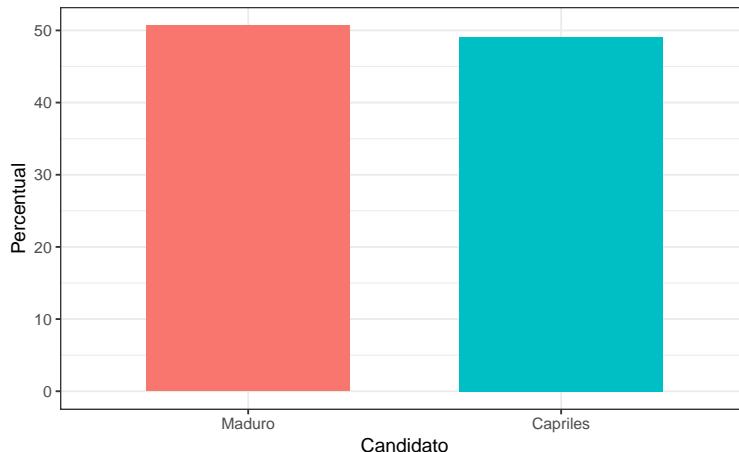


Finalmente, aqui está um exemplo extremo que faz uma diferença muito pequena de menos de 2% parecer 10 a 100 vezes maior:



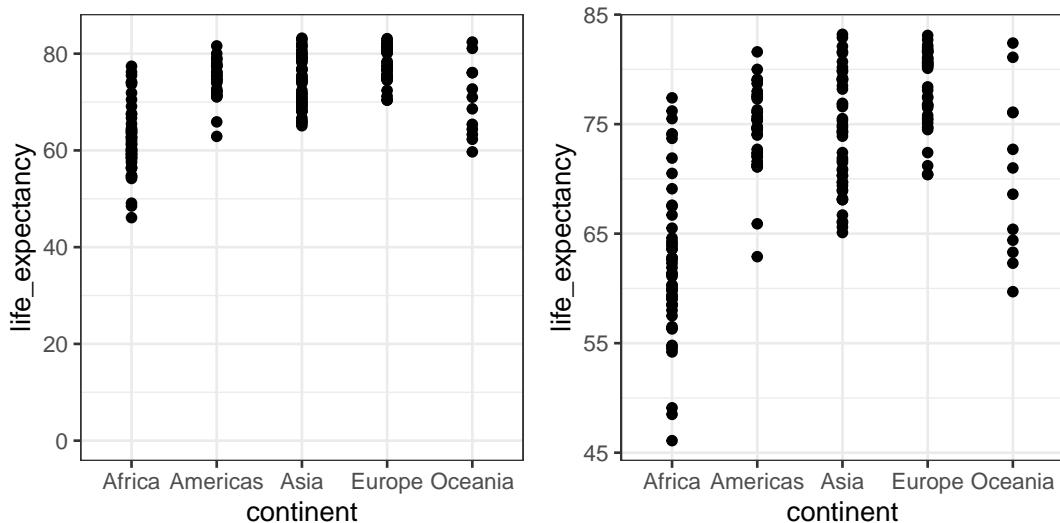
(Fonte: Televisão venezuelana via Pakistan Today⁷ e Diego Mariano)

Aqui está um gráfico mais apropriado:



⁷<https://www.pakistantoday.com.pk/2018/05/18/whats-at-stake-in-venezuelan-presidential-vote>

Ao construir gráficos que usam posições em vez de comprimentos, não é necessário iniciar o eixo y em 0. Particularmente, esse é o caso quando queremos comparar as diferenças entre grupos considerando o quanto seus dados variam. Aqui está um exemplo ilustrativo mostrando a expectativa média de vida de cada país estratificada por continente em 2012:



Observe que, no gráfico à esquerda (que inicia em 0), o espaço entre 0 e 43 não adiciona informações relevantes e, ainda, dificulta a comparação da variabilidade dentro de cada grupo.

10.3 Não distorça quantidades

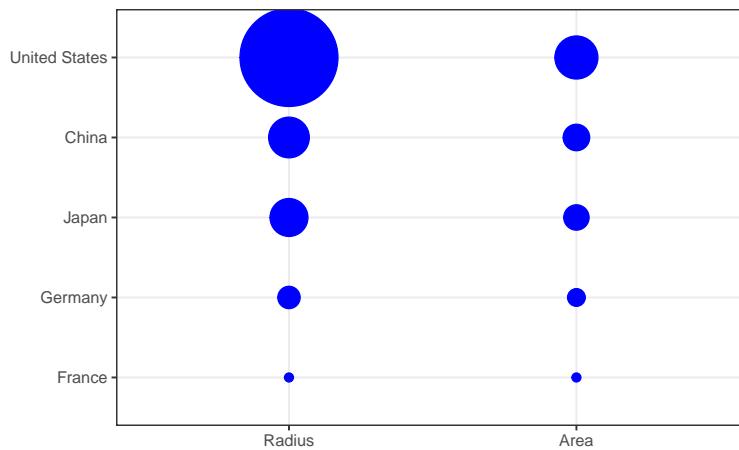
Durante o discurso do Estado da União do presidente Barack Obama em 2011, o gráfico a seguir foi usado para comparar o PIB dos EUA com o PIB de quatro nações concorrentes:



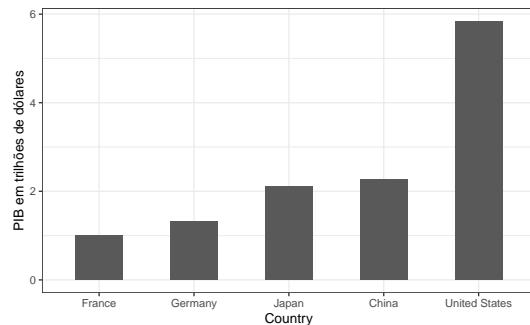
(Fonte: Discurso do Estado da União de 2011⁸)

⁸<https://www.youtube.com/watch?v=kl2g40GoRxg>

Se julgarmos pela área dos círculos, os Estados Unidos parecem ter uma economia cinco vezes maior do que a da China e mais de 30 vezes maior do que a da França. No entanto, se olharmos para os números atuais, veremos que esse não é o caso. As proporções são 2,6 e 5,8 vezes superiores às da China e da França, respectivamente. A razão para essa distorção é que foram utilizados os raios dos círculos para representar as quantidades, em vez das áreas. Como a proporção entre as áreas é ao quadrado, 2,6 se torna 6,5 e 5,8 se torna 34,1. Aqui está uma comparação entre círculos para valores proporcionais ao raio (esquerda) e à área (direita):



Não é surpresa que **ggplot2** use por padrão a área em vez do raio. Obviamente, nesse caso, não devemos usar a área, pois podemos usar a posição e o comprimento:

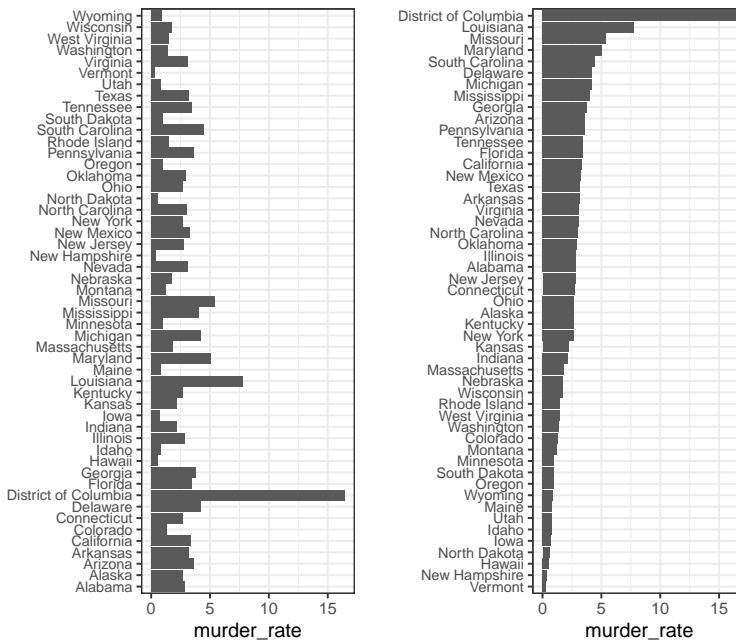


10.4 Ordenando categorias por um valor significativo

Quando um dos eixos é usado para exibir categorias, como nos gráficos de barras, o comportamento padrão do **ggplot2** é classificar as categorias em ordem alfabética quando definidas por cadeias de caracteres. Se as categorias forem definidas por fatores, elas são ordenados de acordo com os níveis dos fatores. Nesses casos, raramente queremos usar a ordem alfabética. Em vez disso, devemos ordenar por quantidades significativa. Em todos os casos apresentados anteriormente, gráficos de barras foram ordenados de acordo com os valores mostrados, exceto nos gráficos de barras comparando navegadores. Nesse caso, mantivemos a ordem igual em todos os gráficos de barras para facilitar a comparação. Especificamente, em vez de

ordenar os navegadores separadamente nos dois anos, ordenamos ambos os anos pelo valor médio de 2000 e 2015.

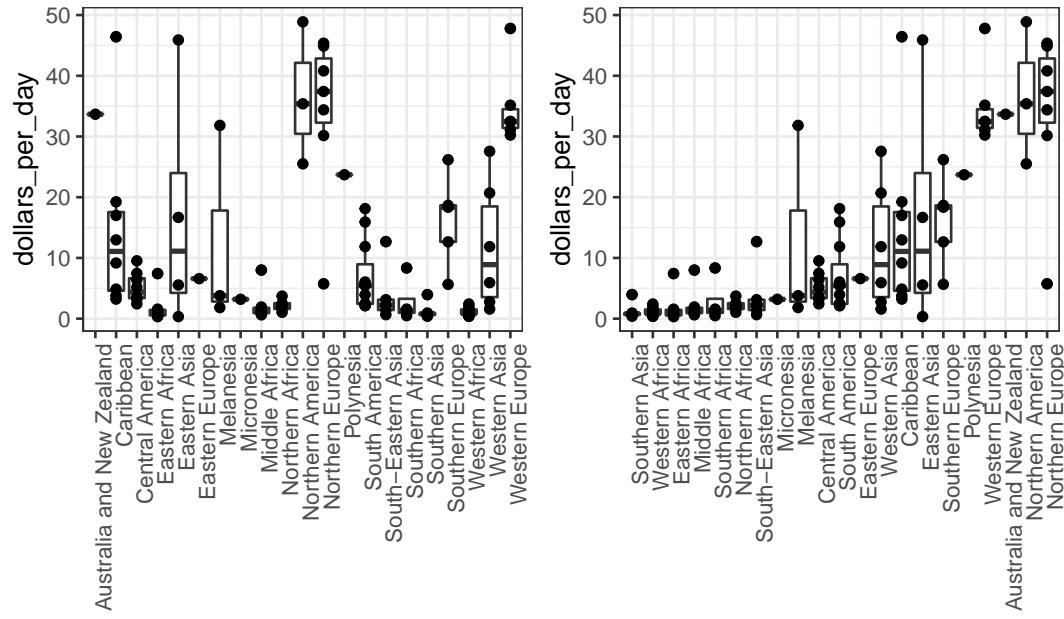
Anteriormente, aprendemos a usar a função `reorder`, o que nos ajuda a alcançar esse objetivo. Para avaliar como a ordem correta pode ajudar a transmitir uma mensagem, suponha que desejamos criar um gráfico para comparar a taxa de homicídios em todos os estados dos EUA. Estamos particularmente interessados nos estados mais perigosos e nos mais seguros. Note a diferença de quando ordenamos alfabeticamente (a ação padrão) versus quando ordenamos pela taxa real:



Podemos fazer o segundo gráfico desta forma:

```
data(murders)
murders %>% mutate(murder_rate = total/ population * 100000) %>%
  mutate(state = reorder(state, murder_rate)) %>%
  ggplot(aes(state, murder_rate)) +
  geom_bar(stat="identity") +
  coord_flip() +
  theme(axis.text.y = element_text(size = 6)) +
  xlab("")
```

A função `reorder` também nos permite reordenar grupos. Anteriormente, vimos um exemplo relacionado à distribuição de renda entre regiões. Aqui vemos as duas versões representadas graficamente lado a lado:



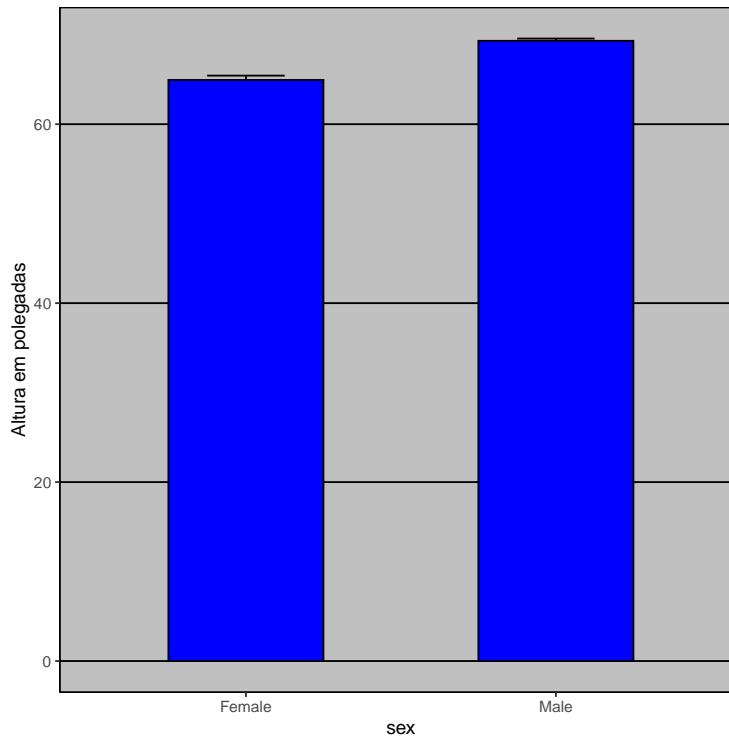
O primeiro gráfico classifica as regiões em ordem alfabética, enquanto o segundo as classifica de acordo com a mediana do grupo.

10.5 Mostre os dados

Nós temos focando em exibir quantidades únicas em todas as categorias. Agora, voltamos nossa atenção para a visualização de dados, com foco na comparação de grupos.

Para motivar nosso primeiro princípio, “mostre os dados”, retornamos ao nosso exemplo artificial de descrição de alturas para ET, o extraterrestre. Desta vez, suponha que o ET esteja interessado nas diferenças de alturas entre homens e mulheres. Um gráfico comumente usado para comparações de grupos, popularizado por programas como o Microsoft Excel, é o gráfico de dinamite (*dynamite plot*), que mostra a média e o erro padrão (os erros padrão são definidos em um capítulo posterior, mas não os confundem com o desvio padrão de dados). O gráfico fica assim:

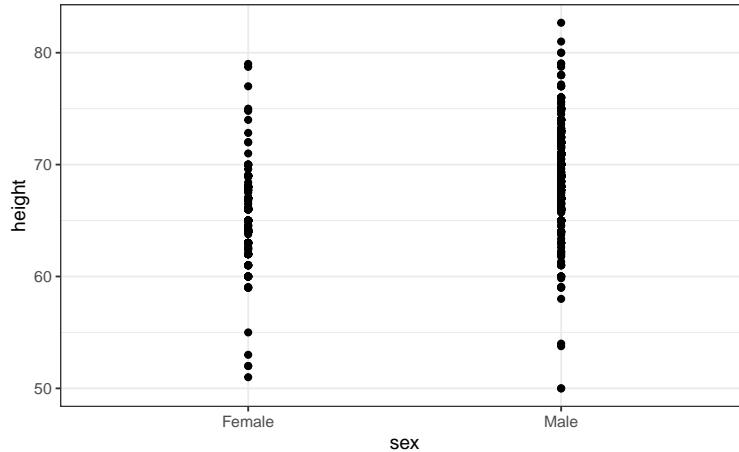
```
#> `summarise()` ungrouping output (override with `groups` argument)
```



A média de cada grupo é representada pelo topo de cada barra e as antenas se estendem da média até mais dois erros padrão. Se tudo o que o ET receber for esse gráfico, ele terá poucas informações sobre o que esperar se encontrar um grupo de homens e mulheres. As barras iniciam em 0: isso significa que existem seres humanos pequenos com menos de 30 centímetros de altura? Todos os homens são mais altos mesmo comparados com as mulheres mais altas? Existe uma variedade de alturas? O ET é incapaz de responder a essas perguntas, pois mal lhe fornecemos informações sobre a distribuição de alturas.

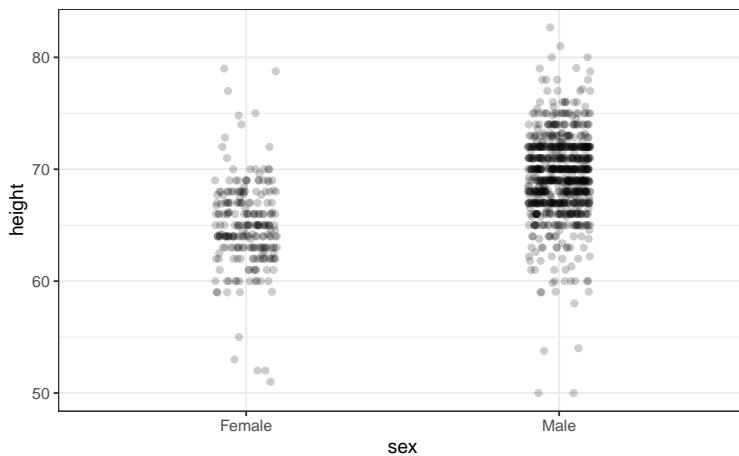
Isso nos leva ao nosso primeiro princípio: exibir os dados. Este simples código do **ggplot2** gera um gráfico mais informativo que o gráfico de barras, simplesmente exibindo todos os pontos de dados:

```
heights %>%
  ggplot(aes(sex, height)) +
  geom_point()
```



A primeira é adicionar *jitter*, que adiciona um pequeno deslocamento aleatório a cada ponto. Nesse caso, adicionar *jitter* horizontal não altera a interpretação, pois as alturas dos pontos não mudam. Além disso, minimizamos o número de pontos que se sobrepõem e, assim, temos uma melhor ideia visual de como os dados estão distribuídos. Uma segunda melhoria vem do uso de *alpha blending*, que torna os pontos um pouco transparentes. Quanto mais pontos se sobrepuarem, mais escuro será o gráfico, o que também nos ajudará a ter uma ideia de como os pontos estão distribuídos. Aqui está o mesmo gráfico com *jitter* e *alpha blending* aplicados:

```
heights %>%  
  ggplot(aes(sex, height)) +  
  geom_jitter(width = 0.1, alpha = 0.2)
```

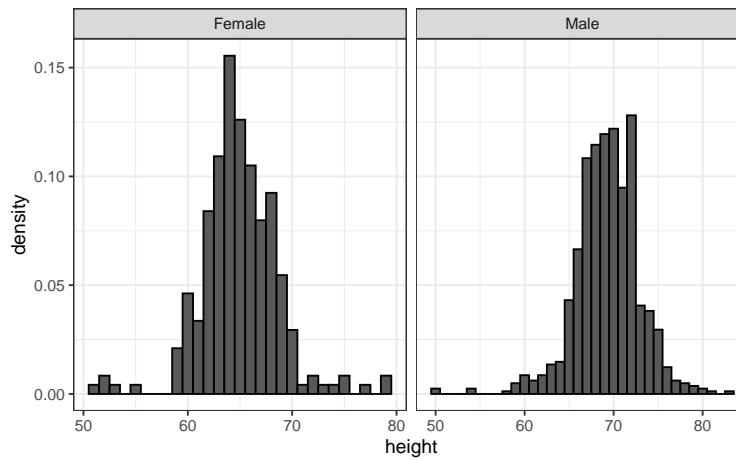


Agora começamos a sentir que, em média, os homens são mais altos que as mulheres. Também observamos faixas horizontais de pontos mais escuras, demonstrando que muitos valores relatados foram arredondados para o número inteiro mais próximo.

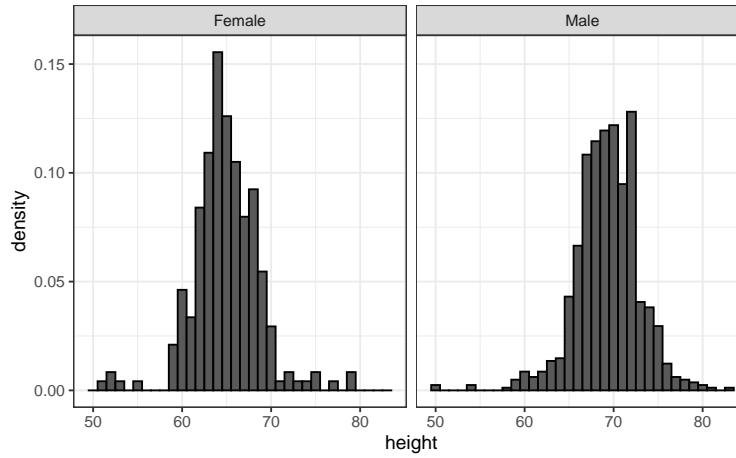
10.6 Facilite comparações

10.6.1 Use eixos em comum

Uma vez que existem muitos pontos, é mais eficaz mostrar distribuições do que pontos individuais. Portanto, mostramos histogramas para cada grupo:

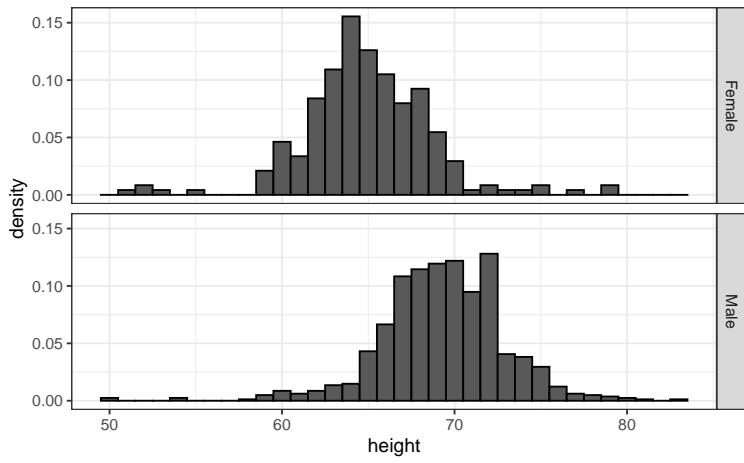


No entanto, olhando para o gráfico acima, não é imediatamente óbvio que homens são, em média, mais altos que mulheres. Temos que observar com atenção para perceber que o eixo x tem uma faixa de valores mais alta no histograma masculino. Um princípio importante aqui é **manter os eixos iguais** ao comparar dados em dois gráficos. A seguir, vemos como a comparação se torna mais fácil:



10.6.2 Alinhe os gráficos verticalmente para ver alterações horizontais e horizontalmente para ver alterações verticais

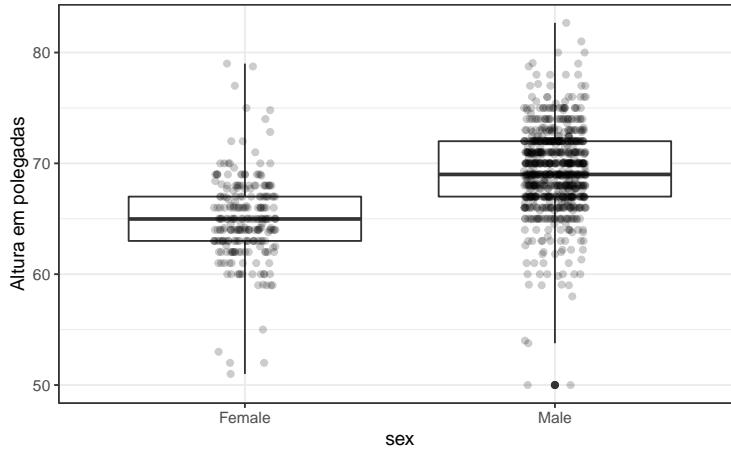
Nesses histogramas, o sinal visual relacionado a reduções ou aumentos de altura são deslocamentos para a esquerda ou para a direita, respectivamente: alterações horizontais. O alinhamento vertical dos gráficos nos ajuda a ver essa alteração quando os eixos são fixos:



```
heights %>%
ggplot(aes(height, ..density..)) +
geom_histogram(binwidth = 1, color="black") +
facet_grid(sex~.)
```

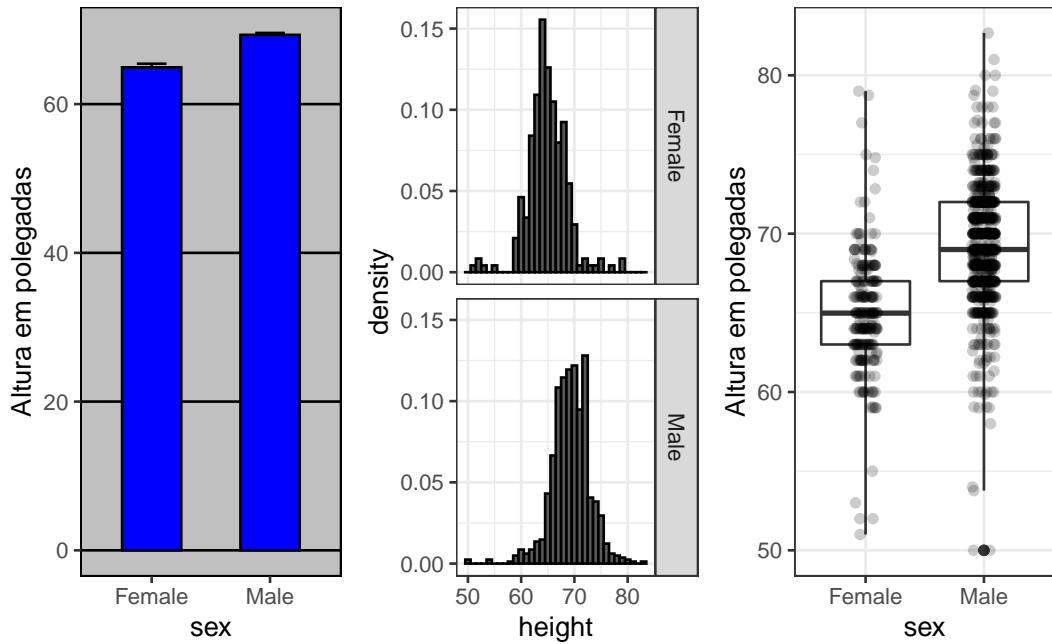
O gráfico acima facilita notar que homens são, em média, mais altos.

Se quisermos obter o resumo compacto que os boxplots oferecem, precisamos alinhá-los horizontalmente, uma vez que, por padrão, os boxplots se movem para cima e para baixo com base nas mudanças de altura. Seguindo o princípio “mostre os dados”, devemos sobrepor todos os pontos de dados:



```
heights %>%
ggplot(aes(sex, height)) +
geom_boxplot(coef=3) +
geom_jitter(width = 0.1, alpha = 0.2) +
ylab("Altura em polegadas")
```

Agora compare e contraste estes três gráficos baseados exatamente nos mesmos dados:



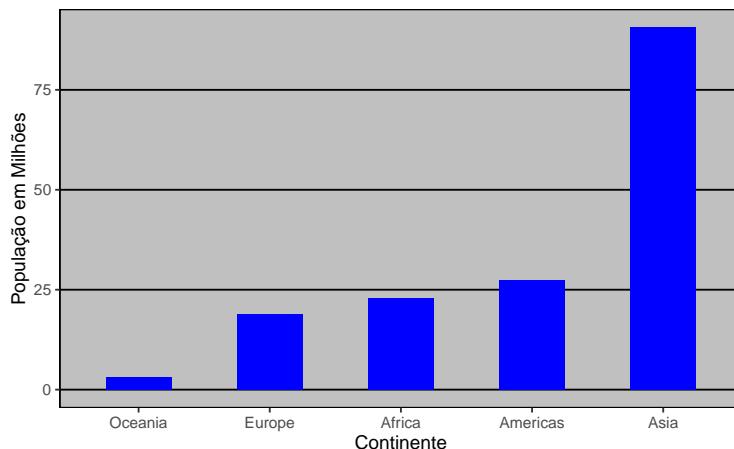
Observe o quanto aprendemos mais nos dois gráficos à direita. Os gráficos de barras são úteis para exibir um número, mas não são muito úteis quando queremos descrever distribuições.

10.6.3 Considere transformações

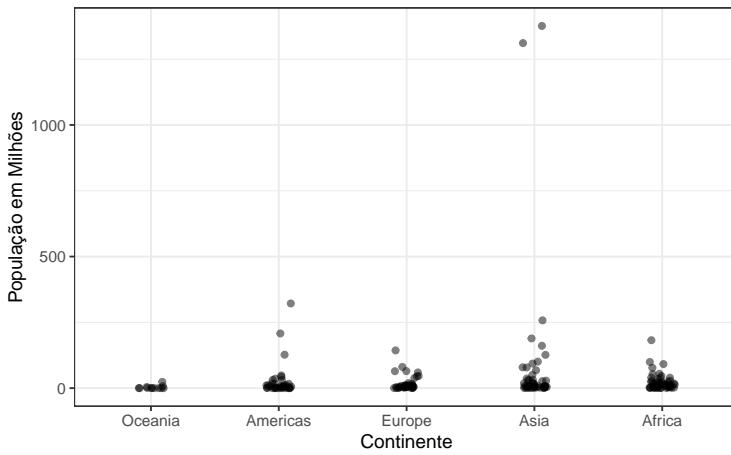
Incentivamos o uso de transformação logarítmica nos casos em que as mudanças são multiplicativas. O tamanho da população foi um exemplo em que estabelecemos uma transformação logarítmica para produzir uma transformação mais informativa.

A combinação de um gráfico de barras escolhido incorretamente e a não utilização de uma transformação logarítmica, quando necessária, podem particularmente causar distorções. Como exemplo, considere este gráfico de barras mostrando os tamanhos médios da população para cada continente em 2015:

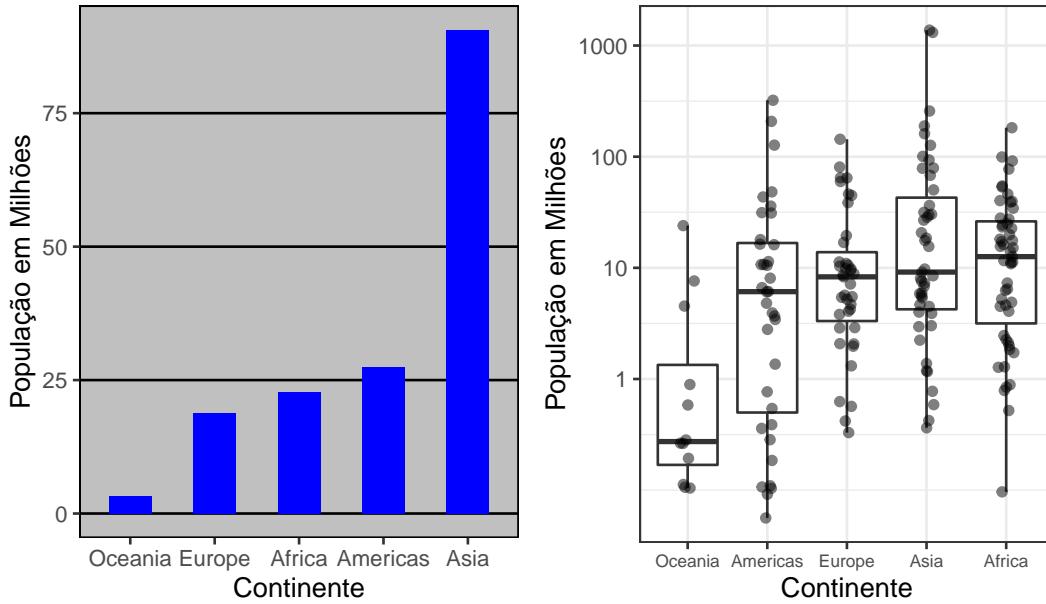
```
#> `summarise()` ungrouping output (override with ` .groups` argument)
```



Observando o gráfico acima, conclui-se que os países asiáticos são muito mais populosos do que os de outros continentes. Seguindo o princípio “mostre os dados”, notamos rapidamente que isso se deve a dois países muito grandes, que presumimos serem a Índia e a China:



Usar uma transformação logarítmica aqui produz um gráfico muito mais informativo. Comparamo o gráfico de barras original com um *boxplot* usando a transformação de escala logarítmica para o eixo y:

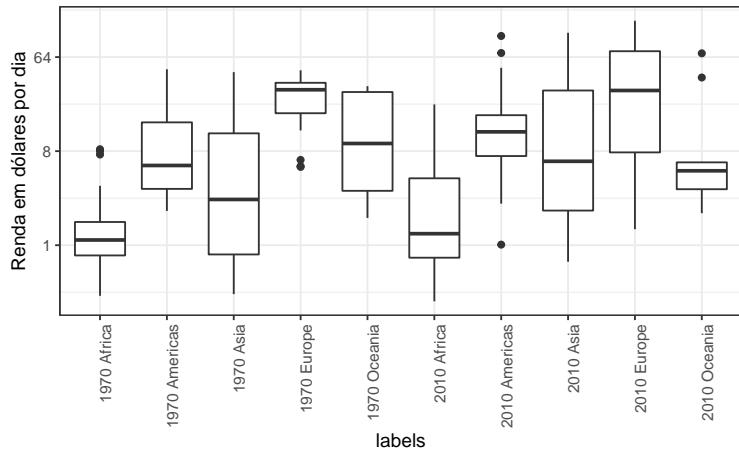


Com o novo gráfico, percebemos que os países africanos realmente têm uma população mediana maior que os da Ásia.

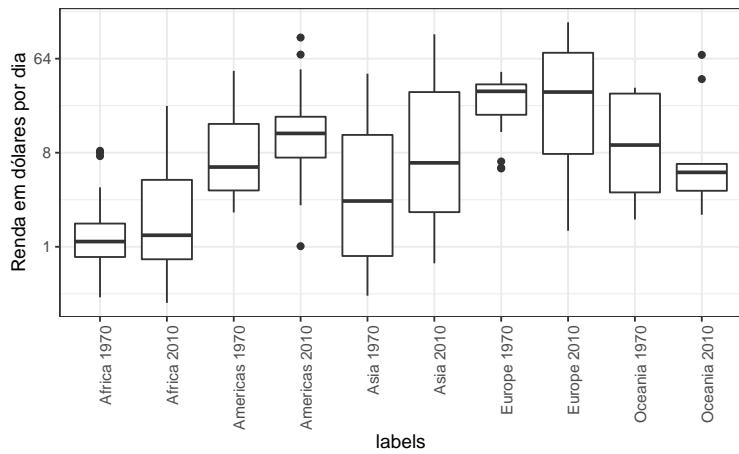
Outras transformações a considerar são a transformação logística (*logit*), que é útil para ver melhor as alterações nas probabilidades e a transformação da raiz quadrada (*sqrt*), que é útil para contagens.

10.6.4 As indicações visuais comparadas devem ser adjacentes

Para cada continente, vamos comparar a renda em 1970 versus 2010. Quando comparamos dados de renda por regiões entre 1970 e 2010, construímos um gráfico semelhante à figura abaixo. Desta vez, investigamos os continentes em vez de regiões.

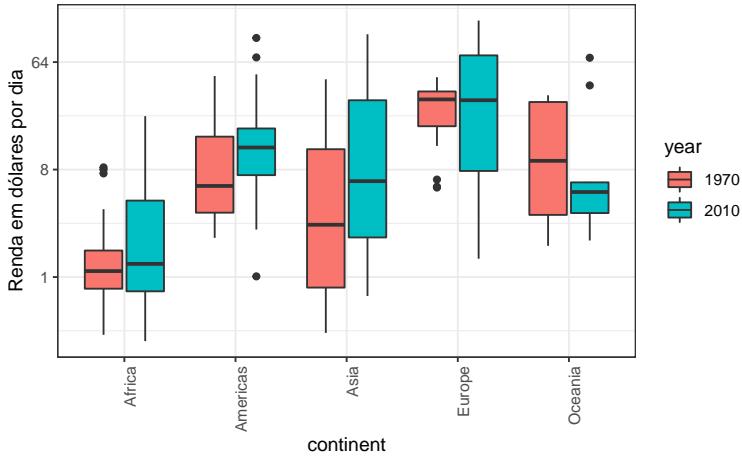


O comportamento padrão do **ggplot2** é classificar os rótulos em ordem alfabética de modo que os rótulos com 1970 venham antes dos rótulos com 2010. Isso dificulta as comparações porque a distribuição de um continente em 1970 é visualmente distante da sua distribuição em 2010. É muito mais fácil fazer a comparação entre 1970 e 2010 para cada continente quando seus *boxplots* estiverem lado a lado:



10.6.5 Use cores

A comparação é ainda mais fácil se usarmos cores para indicar as duas coisas que queremos comparar:



10.7 Pense no daltônico

Cerca de 10% da população é daltônica. Infelizmente, as cores padrão usadas em **ggplot2** não são ideais para esse grupo. No entanto, **ggplot2** facilita a alteração da paleta de cores usada nos gráficos. Aqui está um exemplo de como podemos usar uma paleta que considera daltônicos: [http://www.cookbook-r.com/Graphs/Colors_\(ggplot2\)/#a-colorblind-friendly-palette](http://www.cookbook-r.com/Graphs/Colors_(ggplot2)/#a-colorblind-friendly-palette):

```
color_blind_friendly_cols <-
c("#999999", "#E69F00", "#56B4E9", "#009E73",
 "#FOE442", "#0072B2", "#D55E00", "#CC79A7")
```

Aqui estão as cores:



Além disso, existem vários recursos que podem ajudá-lo a selecionar cores, por exemplo, este: <http://bconnelly.net/2013/10/creating-colorblind-friendly-figures/>.

10.8 Plotagens para duas variáveis

Em geral, deve-se usar gráficos de dispersão para visualizar o relacionamento entre duas variáveis. Em todos os casos em que examinamos a relação entre duas variáveis, incluindo assassinatos totais versus tamanho da população, expectativa de vida versus taxas de fertilidade e mortalidade infantil versus renda, usamos gráficos de dispersão. Esse é o gráfico que geralmente recomendamos. Entretanto, existem algumas exceções e aqui descrevemos dois gráficos alternativos: o gráfico de inclinação (*slope chart*) e o gráfico de Bland-Altman (*Bland-Altman plot*).

10.8.1 Gráficos de inclinação

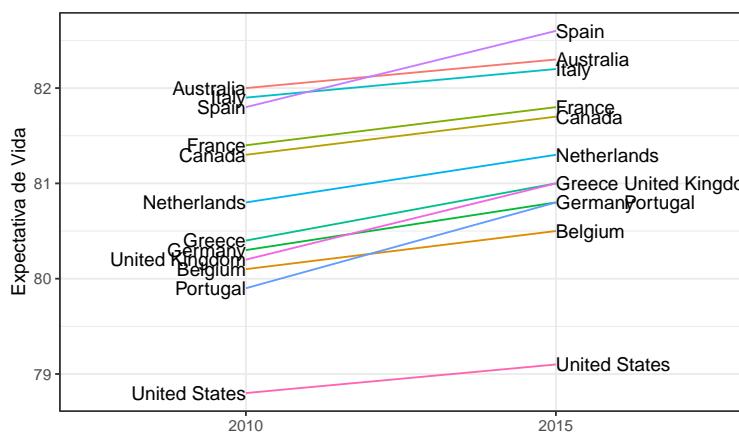
Uma exceção onde outro tipo de gráfico pode ser mais informativo é quando compara-se variáveis do mesmo tipo, mas em momentos diferentes e para um número relativamente pequeno de comparações. Por exemplo, se estivermos comparando a expectativa de vida entre 2010 e 2015. Nesse caso, poderíamos recomendar um gráfico de inclinação.

Não há geometria para gráficos de inclinação no **ggplot2**, mas podemos construir um usando **geom_line**. Precisamos apenas fazer alguns ajustes para adicionar os rótulos. Abaixo, é apresentado um exemplo comparando a expectativa de vida de 2010 a 2015 para os grandes países ocidentais:

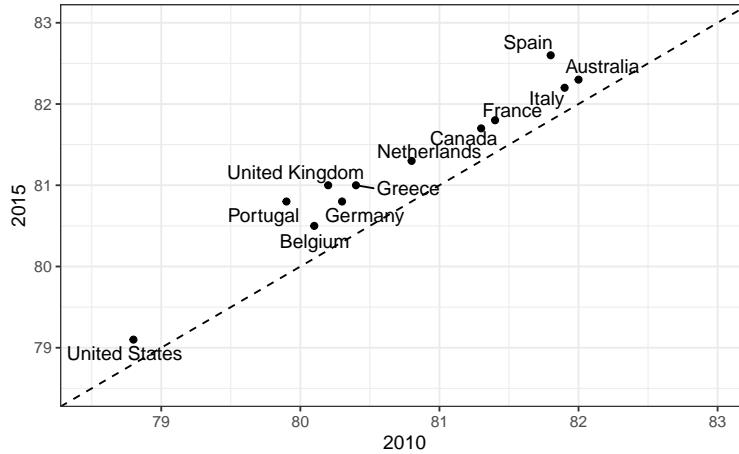
```
west <- c("Western Europe", "Northern Europe", "Southern Europe",
"Northern America", "Australia and New Zealand")

dat <- gapminder %>%
filter(year %in% c(2010, 2015) & region %in% west &
!is.na(life_expectancy) & population > 10^7)

dat %>%
mutate(location = ifelse(year == 2010, 1, 2),
location = ifelse(year == 2015 &
country %in% c("United Kingdom", "Portugal"),
location+0.22, location),
hjust = ifelse(year == 2010, 1, 0)) %>%
mutate(year = as.factor(year)) %>%
ggplot(aes(year, life_expectancy, group = country)) +
geom_line(aes(color = country), show.legend = FALSE) +
geom_text(aes(x = location, label = country, hjust = hjust),
show.legend = FALSE) +
xlab("") + ylab("Expectativa de Vida")
```



Uma vantagem do gráfico de inclinação é que ele rapidamente nos dá uma ideia das mudanças com base na inclinação das linhas. Embora estejamos usando o ângulo como uma sugestão visual, também estamos usando a posição para determinar valores exatos. Comparar melhorias é um pouco mais difícil com um gráfico de dispersão:

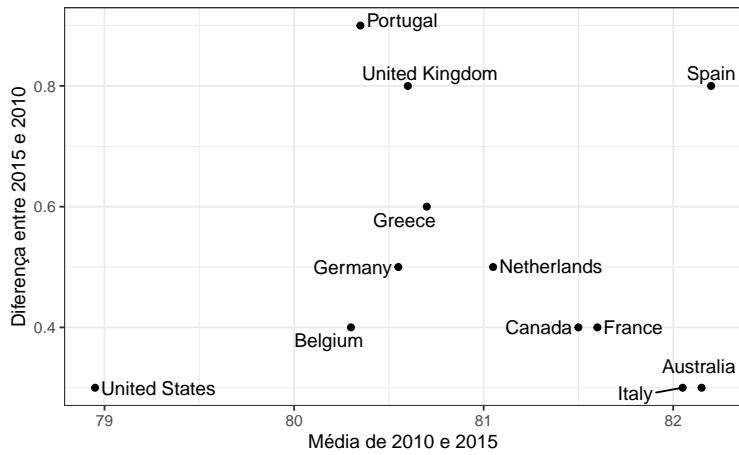


No gráfico de dispersão, seguimos o princípio *use eixos comuns*, uma vez que estamos comparando antes e depois. No entanto, se tivermos muitos pontos, os gráficos de inclinação deixam de ser úteis, pois se torna difícil ver todas as linhas.

10.8.2 Gráfico de Bland-Altman

Como estamos interessados principalmente na diferença, faz sentido dedicar um de nossos eixos a ela. O gráfico de Bland-Altman, também conhecido como gráfico de diferença média de Tukey ou *MA plot*, mostra a diferença em relação à média:

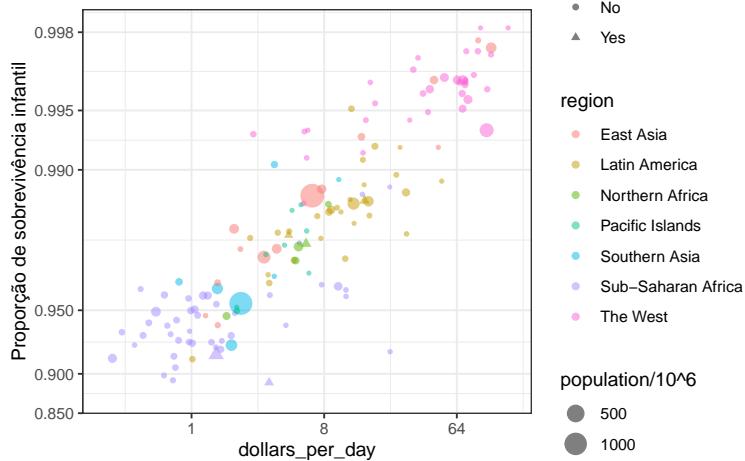
```
library(ggrepel)
dat %>%
  mutate(year = paste0("life_expectancy_", year)) %>%
  select(country, year, life_expectancy) %>%
  spread(year, life_expectancy) %>%
  mutate(average = (life_expectancy_2015 + life_expectancy_2010)/2,
  difference = life_expectancy_2015 - life_expectancy_2010) %>%
  ggplot(aes(average, difference, label = country)) +
  geom_point() +
  geom_text_repel() +
  geom_abline(lty = 2) +
  xlab("Média de 2010 e 2015") +
  ylab("Diferença entre 2015 e 2010")
```



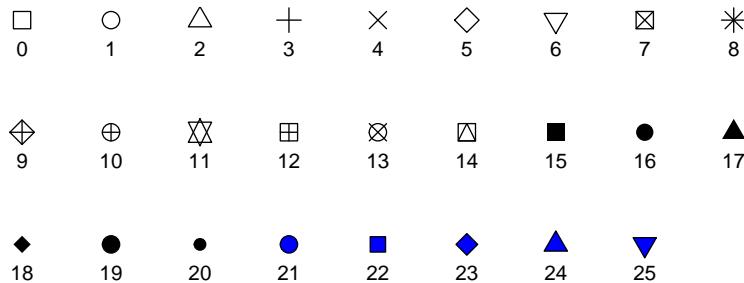
Aqui, simplesmente olhando o eixo y, vemos rapidamente quais países mostraram a maior melhoria. Além disso, temos uma ideia do valor geral do eixo x.

10.9 Codificando uma terceira variável

Um gráfico de dispersão, apresentado anteriormente, mostrou a relação entre sobrevivência infantil e renda média. Abaixo está uma versão desse gráfico que codifica três variáveis: participação na OPEP (Organização dos Países Exportadores de Petróleo), região e população.



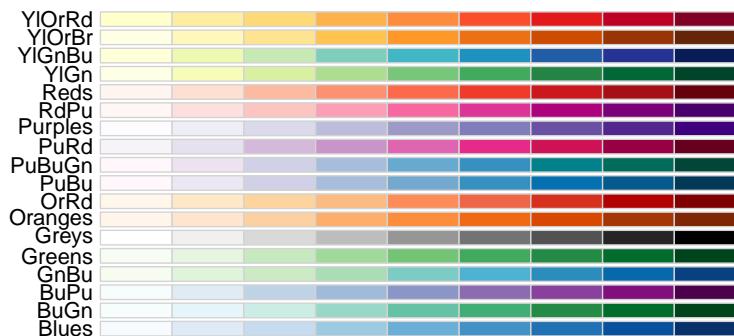
Codificamos variáveis categóricas com cor e forma. Essas formas podem ser controladas com o argumento `shape`. Abaixo, mostramos as formas disponíveis para uso no R. Para as últimas cinco, cores podem ser usadas para preencher o interior da forma.



Para variáveis contínuas, podemos usar cor, intensidade ou tamanho. Aqui está um exemplo de como fazer isso com um estudo de caso.

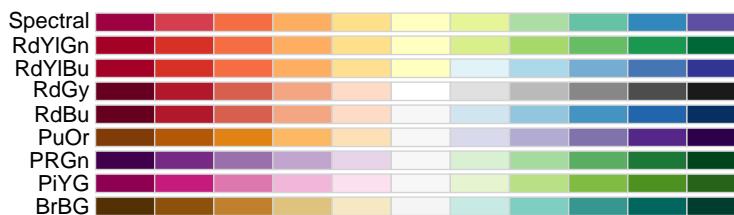
Ao selecionar cores para quantificar uma variável numérica, escolhemos entre duas opções: sequencial ou divergente. Cores sequenciais são adequadas para dados que variam do mais alto ao mais baixo. Valores altos são claramente diferenciados de valores baixos. Aqui estão alguns exemplos oferecidos pelo pacote **RColorBrewer**:

```
library(RColorBrewer)
display.brewer.all(type="seq")
```



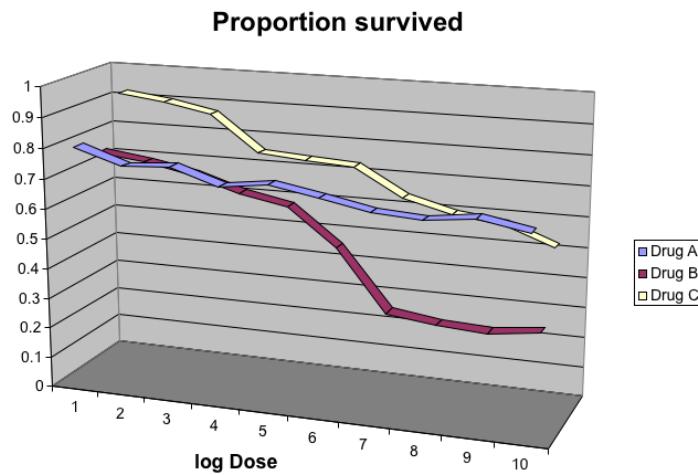
Cores divergentes são usadas para representar valores que divergem de um centro. Colocamos ênfase igual nos dois extremos do intervalo de dados: mais alto que o centro e mais baixo que o centro. Um exemplo de quando usariammos um padrão divergente seria se tivéssemos que mostrar a altura em desvios-padrão da média. Aqui estão alguns exemplos de padrões divergentes:

```
library(RColorBrewer)
display.brewer.all(type="div")
```



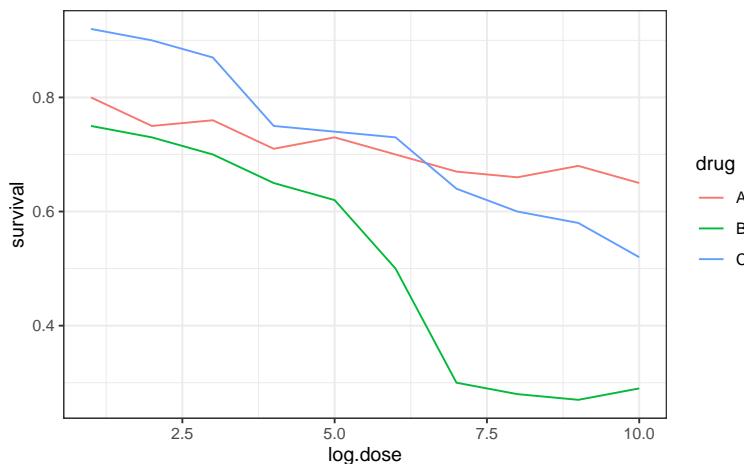
10.10 Evite gráficos pseudo-tridimensionais

A figura a seguir, extraída da literatura científica⁹, mostra três variáveis: dose, tipo de medicamento e sobrevida. Embora telas ou páginas de livro sejam planas e bidimensionais, o gráfico tenta imitar três dimensões, atribuindo uma dimensão para cada variável.



(Imagen cortesia de Karl Bromann)

Seres humanos não são bons em ver em três dimensões (o que explica por que é tão difícil realizar uma baliza para estacionar). Nossa limitação é ainda pior em relação aos espaços pseudo-tridimensionais. Para ver isso, tente determinar os valores da variável de sobrevivência no gráfico acima. Você pode dizer quando a fita roxa intercepta a fita vermelha? Abaixo está um exemplo em que podemos facilmente usar cores para representar a variável categórica em vez de um pseudo-3D:

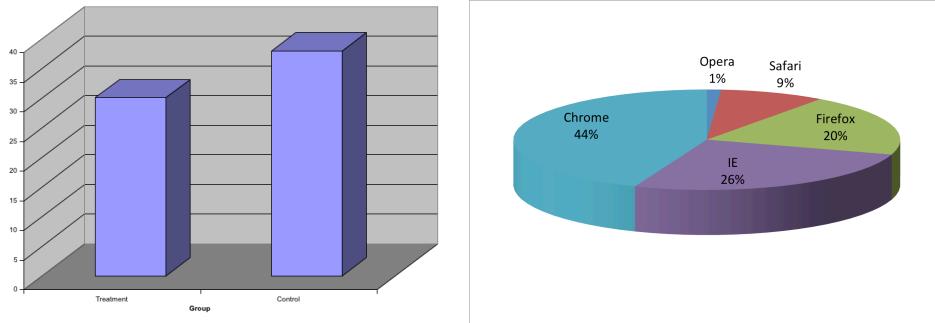


Note como é mais fácil determinar os valores de sobrevivência.

Às vezes, o pseudo-3D é usado de forma totalmente gratuita: os gráficos são criados para

⁹https://projecteuclid.org/download/pdf_1/euclid.ss/1177010488

parecer 3D, mesmo quando a terceira dimensão não representa uma quantidade. Isso apenas aumenta a confusão e torna mais difícil transmitir sua mensagem. Aqui estão dois exemplos:



(Imagens cortesia de Karl Broman)

10.11 Evite muitos dígitos significativos

Por padrão, softwares estatísticos como R retornam muitos dígitos significativos. O comportamento padrão do R é exibir 7 dígitos significativos. Esse número de dígitos geralmente não adiciona informações e a desordem visual adicionada pode dificultar o entendimento da mensagem. Como exemplo, aqui estão as taxas de doenças por 10.000 habitantes para a Califórnia em cinco décadas, calculadas a partir dos totais e da população com R:

state	year	Measles	Pertussis	Polio
California	1940	37.8826320	18.3397861	0.8266512
California	1950	13.9124205	4.7467350	1.9742639
California	1960	14.1386471	NA	0.2640419
California	1970	0.9767889	NA	NA
California	1980	0.3743467	0.0515466	NA

(Measles: Sarampo, Pertussis: Coqueluche, Polio: Poliomielite)

Nesse caso, estamos relatando precisão de até 0,00001 casos por 10.000, um valor muito pequeno no contexto das mudanças que estão ocorrendo nas datas. Nesse caso, dois algarismos significativos são mais do que suficientes e deixam claro que as taxas estão diminuindo:

state	year	Measles	Pertussis	Polio
California	1940	37.9	18.3	0.8
California	1950	13.9	4.7	2.0
California	1960	14.1	NA	0.3
California	1970	1.0	NA	NA
California	1980	0.4	0.1	NA

Para alterar o número de dígitos significativos ou números redondos, usamos `signif` e `round`. Você pode definir globalmente o número de dígitos significativos, configurando as opções desta forma: `options(digits = 3)`.

Outro princípio relacionado à exibição de tabelas é colocar os valores que são comparados em colunas em vez de linhas. Observe que nossa tabela acima é mais fácil de ler do que esta:

state	disease	1940	1950	1960	1970	1980
California	Measles	37.9	13.9	14.1	1	0.4
California	Pertussis	18.3	4.7	NA	NA	0.1
California	Polio	0.8	2.0	0.3	NA	NA

10.12 Conheça seu público

Frágrafos podem ser usados para: (1) nossas próprias análises exploratórias de dados, (2) para transmitir uma mensagem a especialistas, ou (3) para ajudar a contar uma história para o público em geral. Logo, certifique-se de que o público-alvo entenda cada elemento do gráfico.

Como um simples exemplo, considere que, para sua própria exploração, pode ser mais útil transformar os dados logaritmicamente e depois plotá-los. No entanto, para uma audiência geral que não está familiarizada com a conversão de valores logarítmicos em medições originais, será muito mais fácil entender o uso de uma escala logarítmica para o eixo, em vez de valores transformados logaritmicamente.

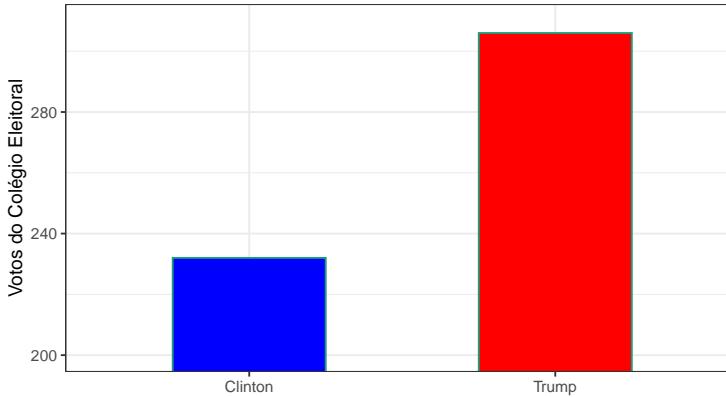
10.13 Exercícios

Para esses exercícios, usaremos os dados de vacinas do pacote **dslabs**:

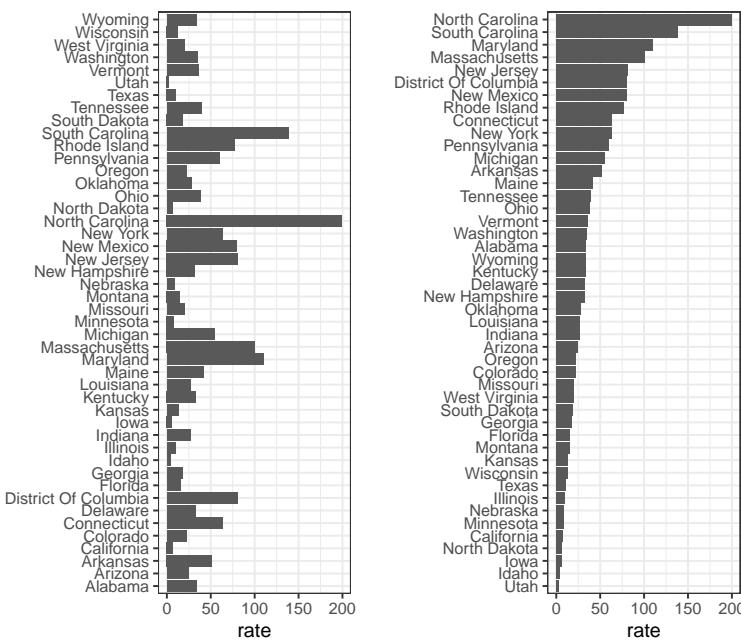
```
library(dslabs)
data(us_contagious_diseases)
```

1. Os gráficos de pizza são adequados:
 - a. Quando queremos mostrar porcentagens.
 - b. Quando **ggplot2** não estiver disponível.
 - c. Quando estou em uma pizzaria.
 - d. Nunca. Gráficos de barras e tabelas são sempre melhores.
2. Qual é o problema com o gráfico abaixo:

Resultados da Eleição Presidencial de 2016



- a. Os valores estão errados. A votação final foi 306-232.
 b. O eixo não começa em 0. A julgar pelo comprimento, parece que Trump recebeu três vezes mais votos quando, na verdade, recebeu aproximadamente 30% a mais.
 c. As cores devem ser as mesmas.
 d. As porcentagens devem ser mostradas como um gráfico de pizza.
3. Veja os dois gráficos a seguir. Eles mostram a mesma informação: taxas de sarampo em 1928 em todos os 50 estados dos EUA.



Qual gráfico é mais fácil de ler se você deseja determinar quais são os melhores e os piores estados em termos de taxas? Por quê?

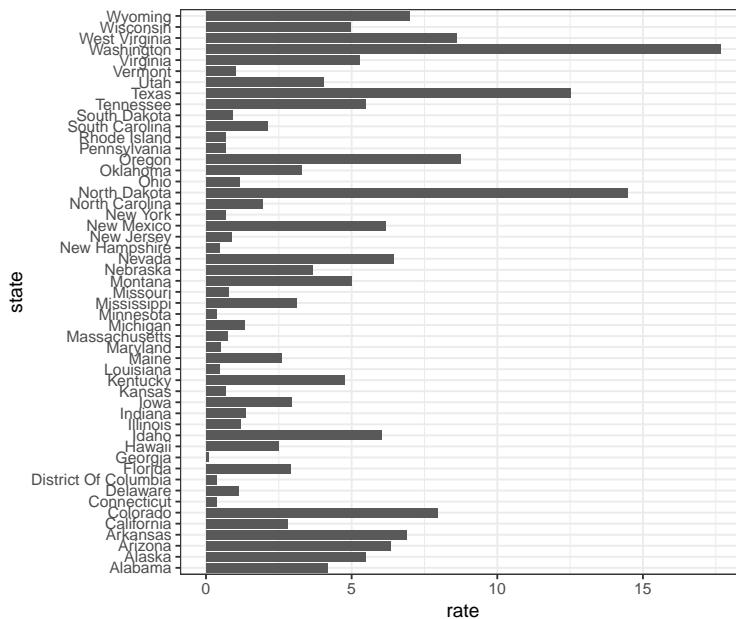
- a. Eles dão a mesma informação, então ambos são igualmente bons.
 b. O gráfico à esquerda é melhor porque ordena os estados em ordem alfabética.
 c. O gráfico à direita é melhor porque a ordem alfabética não tem nada a ver com a doença e, ordenando de acordo com a taxa real, vemos rapidamente os estados com as taxas mais altas e mais baixas.

- d. Ambos os gráficos deveriam ser um gráfico de pizza.
4. Para fazer o gráfico à esquerda, precisamos reordenar os níveis das variáveis dos estados.

```
dat <- us_contagious_diseases %>%
  filter(year == 1967 & disease=="Measles" & !is.na(population)) %>%
  mutate(rate = count/ population * 10000 * 52/ weeks_reporting)
```

Note o que acontece quando criamos um gráfico de barras:

```
dat %>% ggplot(aes(state, rate)) +
  geom_bar(stat="identity") +
  coord_flip()
```



Defina estes objetos:

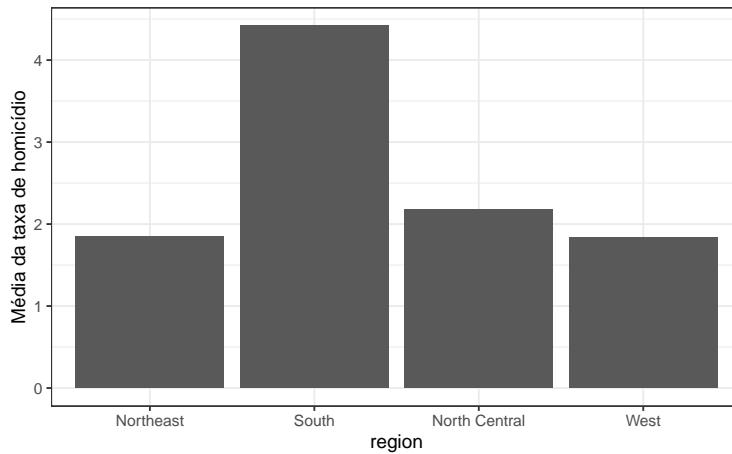
```
state <- dat$state
rate <- dat$count/dat$population*10000*52/dat$weeks_reporting
```

Redefina o objeto **state** para que os níveis sejam reorganizados. Imprima o novo objeto **state** e seus níveis para que você possa ver que os níveis não reorganizam o vetor.

5. Agora, edite o código acima redefinindo **dat** para que os níveis da variável **state** sejam reordenados pela variável **rate**. Em seguida, faça um gráfico de barras usando o código acima, mas para este novo **dat**.
6. Digamos que você esteja interessado em comparar as taxas de homicídio por arma de fogo em todas as regiões dos EUA. Ao ver este gráfico:

```
library(dslabs)
data("murders")
murders %>% mutate(rate = total/population*100000) %>%
  group_by(region) %>%
  summarize(avg = mean(rate)) %>%
  mutate(region = factor(region)) %>%
```

```
ggplot(aes(region, avg)) +
  geom_bar(stat="identity") +
  ylab("Média da taxa de homicídio")
#> `summarise()` ungrouping output (override with `.groups` argument)
```



(Northeast: Nordeste, South: Sul, North Central: Centro-norte, West: Oeste)

você decide se mudar para um estado na região oeste. Qual é o principal problema com essa interpretação?

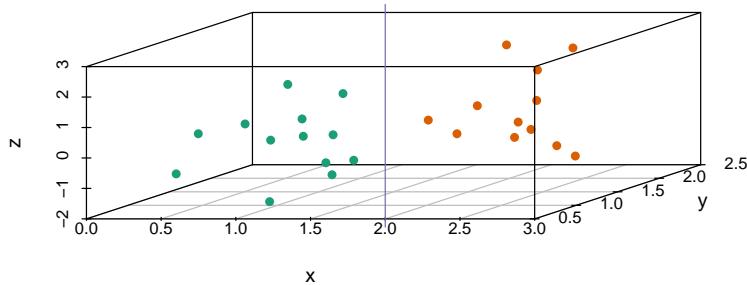
- a. As categorias estão organizadas em ordem alfabética.
- b. O gráfico não mostra erros padrão.
- c. O gráfico não mostra todos os dados. Não vemos variabilidade dentro de uma região. É possível que estados mais seguros possam não estar no oeste.
- d. O Nordeste tem a menor média.

7. A taxa de homicídios pode ser definida da seguinte forma:

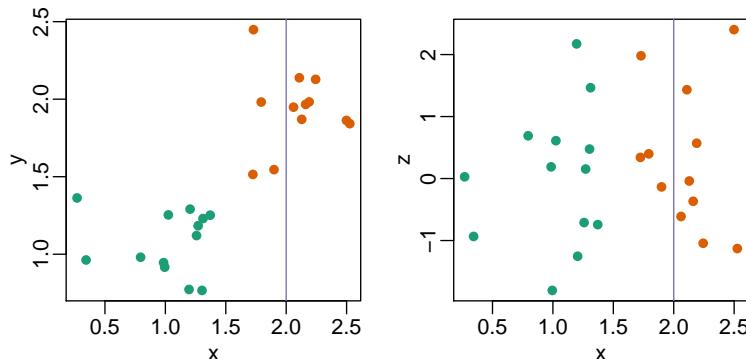
```
data("murders")
murders %>% mutate(rate = total/population*100000)
```

Faça um *boxplot* das taxas de homicídio (*murders*) por região, mostrando todos os pontos e ordenando as regiões pela taxa média.

8. Os gráficos abaixo mostram três variáveis contínuas.



A linha $x = 2$ parece separar os pontos. Mas, na realidade, não é o caso, como vemos quando representamos graficamente os dados em um par de pontos bidimensionais.



Por que acontece isso?

- a. Os seres humanos não são bons em ler gráficos pseudo-3D.
 - b. Deve haver um erro no código.
 - c. Cores nos confundem.
 - d. Diagramas de dispersão não devem ser usados para comparar duas variáveis quando tivermos acesso a três variáveis.
9. Reproduza a imagem do gráfico que criamos anteriormente para varíola (*smallpox*). Para este gráfico, não inclua anos em que nenhum caso foi relatado em 10 ou mais semanas.
10. Agora, repita o gráfico de séries temporais que criamos anteriormente, mas desta vez seguindo as instruções da pergunta anterior.
11. Para o estado da Califórnia, faça gráficos de séries temporais que mostrem as taxas de todas as doenças. Inclua apenas anos nos quais os dados são fornecidos em 10 ou mais semanas. Use uma cor diferente para cada doença.
12. Agora faça o mesmo para as taxas dos EUA. Dica: calcule a taxa dos EUA usando `summarize`, o número total de casos dividido pela população total.

10.14 Estudo de caso: vacinas e doenças infecciosas

Vacinas ajudaram a salvar milhões de vidas. No século 19, antes da imunidade de rebanho ser alcançada por meio de programas de vacinação, as mortes por doenças infecciosas, como varíola e poliomielite, eram comuns. No entanto, hoje os programas de vacinação se tornaram um tanto controversos, apesar de todas as evidências científicas de sua importância.

A controvérsia começou com um artigo¹⁰ publicado em 1988 e liderado por Andrew Wakefield, que alegava a existência de uma ligação entre a administração da vacina contra sarampo, caxumba e rubéola e autismo e doenças intestinais. Apesar do grande conjunto de evidências científicas que contradizem esse achado, os relatos de tabloides e o sensacionalismo daqueles que acreditam em teorias da conspiração levaram parte do público a acreditar que as vacinas eram prejudiciais. Como resultado, muitos pais deixaram de vacinar seus filhos. Essa prática perigosa pode ser potencialmente desastrosa, uma vez que os Centros de Controle de Doenças dos EUA, ou CDC, estimam que a vacinação evitará mais

¹⁰[http://www.thelancet.com/journals/lancet/article/PIIS0140-6736\(97\)11096-0/abstract](http://www.thelancet.com/journals/lancet/article/PIIS0140-6736(97)11096-0/abstract)

de 21 milhões de hospitalizações e 732.000 mortes entre crianças americanas nascidas nos últimos 20 anos (consulte “Benefícios da imunização durante a era do programa Vacinas para crianças - Estados Unidos, 1994-2013, MMWR”¹¹). Desde então, o periódico “The Lancet” retirou o artigo e Andrew Wakefield foi “removido do registro médico do Reino Unido, com uma declaração identificando falsificação deliberada na pesquisa publicada no *The Lancet*, e foi, portanto, impedido de praticar medicina no Reino Unido” (Fonte: Wikipedia¹²). Ainda assim, os equívocos permanecem, em parte por causa de ativistas autoproclamados que continuam a disseminar informações incorretas sobre vacinas.

A comunicação eficaz de dados é um forte antídoto para desinformação e os fomentadores de medo. Anteriormente, mostramos um exemplo de um artigo do *Wall Street Journal*¹³ que mostra dados relacionados ao impacto das vacinas na luta contra doenças infecciosas. Vamos reconstruir esse exemplo a seguir.

Os dados usados para esses gráficos foram coletados, organizados e distribuídos pelo *Tycho Project*¹⁴. Eles incluem, semanalmente, contagens reportadas para sete doenças de 1928 a 2011, para todos os 50 estados dos EUA. Incluímos os totais anuais no pacote **dslabs**:

```
library(tidyverse)
library(RColorBrewer)
library(dslabs)
data(us_contagious_diseases)
names(us_contagious_diseases)
#> [1] "disease"           "state"              "year"
#> [4] "weeks_reporting"   "count"              "population"
```

Criamos um objeto temporário **dat** que armazena apenas os dados do sarampo, inclui a taxa por 100.000, ordena os estados de acordo com o valor médio da doença e remove o Alasca e o Havaí desde que esses dois se tornaram estados no final da década de 1950. Observe que existem uma coluna **weeks_reporting** que nos diz para quantas semanas do ano há dados relatados. Temos que ajustar esse valor ao calcular a taxa:

```
the_disease <- "Measles"
dat <- us_contagious_diseases %>%
filter(!state%in%c("Hawaii", "Alaska") & disease == the_disease) %>%
mutate(rate = count/ population * 10000 * 52/ weeks_reporting) %>%
mutate(state = reorder(state, rate))
```

Agora podemos facilmente plotar as taxas de doenças por ano. Aqui estão os dados de sarampo na Califórnia:

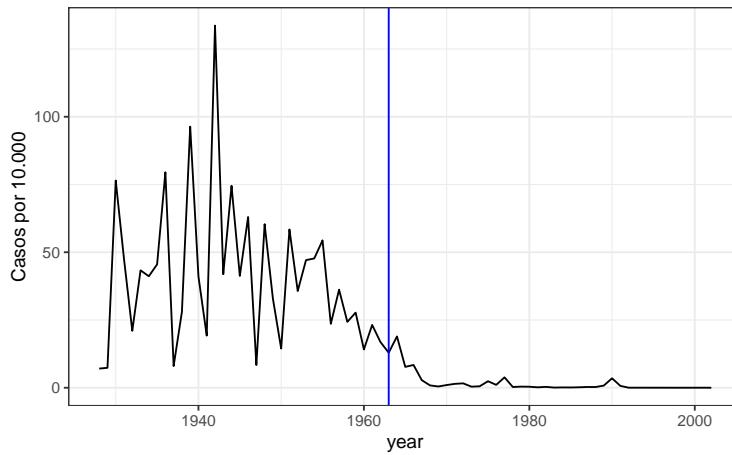
```
dat %>% filter(state == "California" & !is.na(rate)) %>%
ggplot(aes(year, rate)) +
geom_line() +
ylab("Casos por 10.000") +
geom_vline(xintercept=1963, col = "blue")
```

¹¹<https://www.cdc.gov/mmwr/preview/mmwrhtml/mm6316a4.htm>

¹²https://es.wikipedia.org/wiki/Andrew_Wakefield

¹³<http://graphics.wsj.com/infectious-diseases-and-vaccines/>

¹⁴<http://www.tycho.pitt.edu/>



Adicionamos uma linha vertical em 1963, já que foi quando a vacina foi introduzida¹⁵.

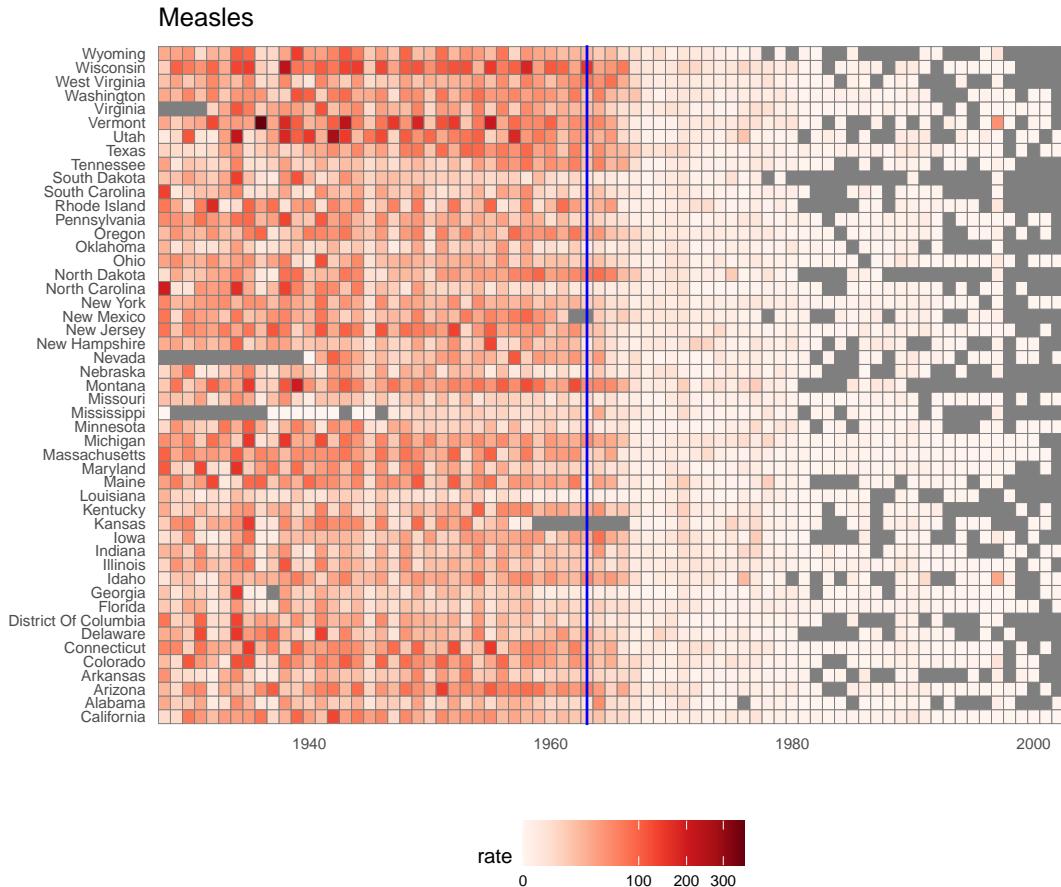
Agora podemos exibir dados para todos os estados em um gráfico? Temos três variáveis para incluir: ano, estado e taxa. Na figura do WSJ, eles usam o eixo x para o ano, o eixo y para o estado e o tom da cor para representar as taxas. No entanto, a escala de cores que eles usam, que vai de amarelo a azul, a verde, a laranja e a vermelho, pode ser melhorada.

Em nosso exemplo, queremos usar uma paleta sequencial, pois não há um centro significativo, apenas taxas baixas e altas.

Usamos a geometria `geom_tile` preencher a região com cores que representam as taxas de doenças. Usamos uma transformação de raiz quadrada para impedir que contagens particularmente altas dominem o gráfico. Observe que os valores ausentes são mostrados em cinza. Além disso, observe que, assim que uma doença foi praticamente erradicada, alguns estados deixaram de relatar casos. É por essa razão que vemos tanto cinza depois de 1980.

```
dat %>% ggplot(aes(year, state, fill = rate)) +
  geom_tile(color = "grey50") +
  scale_x_continuous(expand=c(0,0)) +
  scale_fill_gradientn(colors = brewer.pal(9, "Reds"), trans = "sqrt") +
  geom_vline(xintercept=1963, col = "blue") +
  theme_minimal() +
  theme(panel.grid = element_blank(),
        legend.position="bottom",
        text = element_text(size = 8)) +
  ggtitle(the_disease) +
  ylab("") + xlab("")
```

¹⁵Control, Centers for Disease; Prevenção (2014). Informações de saúde do CDC para viagens internacionais em 2014 (o livro amarelo). p. 250. ISBN 9780199948505



Este gráfico fornece impressionantes evidências em favor da contribuição das vacinas. Entretanto, uma limitação dessa visualização é que ela usa cores para representar quantidades, o que, como previamente explicado, dificulta conhecer exatamente quão altos são os valores. Posição e comprimento são melhores sinais. Se estamos dispostos a perder informações de estados, podemos fazer uma versão do gráfico que mostra os valores com posições. Também podemos mostrar a média para os EUA, que calculamos assim:

```
avg <- us_contagious_diseases %>%
filter(disease==the_disease) %>% group_by(year) %>%
summarize(us_rate = sum(count, na.rm = TRUE)/
sum(population, na.rm = TRUE) * 10000)
#> `summarise()` ungrouping output (override with `.`groups` argument)
```

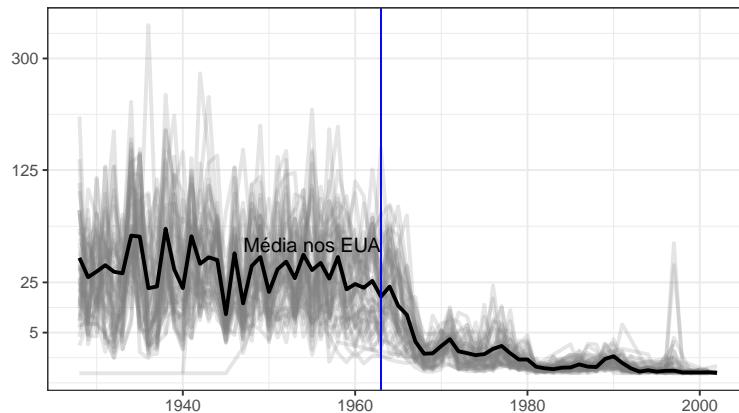
Agora, para fazer o gráfico, simplesmente usamos a geometria `geom_line`:

```
dat %>%
filter(!is.na(rate)) %>%
ggplot() +
geom_line(aes(year, rate, group = state), color = "grey50",
show.legend = FALSE, alpha = 0.2, size = 1) +
geom_line(mapping = aes(year, us_rate), data = avg, size = 1) +
scale_y_continuous(trans = "sqrt", breaks = c(5, 25, 125, 300)) +
ggttitle("Casos por 10.000 por estado") +
```

```

xlab("") + ylab("") +
geom_text(data = data.frame(x = 1955, y = 50),
mapping = aes(x, y, label="Média nos EUA"),
color="black") +
geom_vline(xintercept=1963, col = "blue")

```

Casos por 10.000 por estado

Em teoria, poderíamos usar cores para representar estados, que em uma variável categórica, mas é difícil escolher 50 cores diferentes.

10.15 Exercícios

1. Reproduza o mapa da matriz que fizemos anteriormente para varíola (*smallpox*). Para esse gráfico, não inclua os anos em que nenhum caso foi relatado por 10 ou mais semanas.
2. Agora, reproduza o gráfico de séries temporais que criamos anteriormente, mas desta vez seguindo as instruções da pergunta anterior para a varíola (*smallpox*).
3. Para o estado da Califórnia, faça um gráfico de série temporal mostrando as taxas de todas as doenças. Inclua apenas anos com 10 ou mais relatórios semanais. Use uma cor diferente para cada doença.
4. Agora faça o mesmo para as taxas dos EUA. Dica: calcule a taxa dos EUA usando `summarize: total dividido pelo tamanho da população total`.

11

Resumos robustos

11.1 Valores atípicos

Anteriormente, descrevemos como os boxplots mostram *outliers*, mas não oferecemos uma definição precisa. Aqui discutimos discrepâncias, abordagens que podem ajudar a detectá-las e resumos que levam sua presença em consideração.

Os valores discrepantes são muito comuns na ciência de dados. A coleta de dados pode ser complexa e é comum observar pontos de dados gerados com erro. Por exemplo, um dispositivo de monitoramento antigo pode ler medições sem sentido antes de falhar completamente. O erro humano também é uma fonte de outliers, principalmente quando a entrada de dados é feita manualmente. Um indivíduo, por exemplo, pode digitar erroneamente sua altura em centímetros em vez de polegadas ou colocar o decimal no lugar errado.

Como distinguimos um outlier de medições que são muito grandes ou muito pequenas simplesmente por causa da variabilidade esperada? Nem sempre é uma pergunta fácil de responder, mas tentaremos oferecer algumas orientações. Vamos começar com um caso simples.

Suponha que um colega seja responsável por coletar dados demográficos para um grupo de homens. Os dados indicam a altura em pés e são armazenados no objeto:

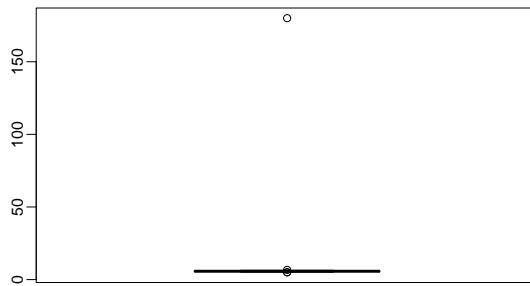
```
library(tidyverse)
library(dslabs)
data(outlier_example)
str(outlier_example)
#> num [1:500] 5.59 5.8 5.54 6.15 5.83 5.54 5.87 5.93 5.89 5.67 ...
```

Nosso colega usa o fato de que as alturas geralmente são bem aproximadas por uma distribuição normal e resume os dados com a média e o desvio padrão:

```
mean(outlier_example)
#> [1] 6.1
sd(outlier_example)
#> [1] 7.8
```

e escreva um relatório sobre o fato interessante de que esse grupo de meninos é muito mais alto que o normal. A altura média é superior a seis pés! No entanto, usando seus conhecimentos de ciência de dados, eles percebem algo inesperado: o desvio padrão é superior a 7 pés. Ao somar e subtrair dois desvios padrão, eles observam que 95% dessa população parece ter alturas entre -9.489, 21.697 pés, isso não faz sentido. Um gráfico rápido mostra o problema:

```
boxplot(outlier_example)
```



Parece haver pelo menos um valor que não faz sentido, pois sabemos que uma altura de 180 pés é impossível. O gráfico da caixa detecta esse ponto como um erro externo.

11.2 Mediana

Quando temos um erro externo como esse, a média pode ser muito grande. Matematicamente, podemos tornar a média tão grande quanto queremos simplesmente mudando um número: com 500 pontos de dados, podemos aumentar a média em qualquer quantidade Δ adicionando $\Delta \times \text{NA}$ para um único número. A mediana, definida como o valor pelo qual metade dos valores é menor e a outra metade é maior, é robusta para esses valores extremos. Por maior que seja o ponto maior, a mediana permanece a mesma.

Com esses dados, a mediana é:

```
median(outlier_example)
#> [1] 5.74
```

que é sobre 5 pés e 9 polegadas.

A mediana é o que os gráficos da caixa mostram como uma linha horizontal.

11.3 O intervalo interquartil (IQR)

A caixa em um gráfico de caixa é definida pelo primeiro e terceiro quartis. Estes têm como objetivo fornecer uma idéia da variabilidade nos dados: 50% dos dados estão dentro desse intervalo. A diferença entre o terceiro e o primeiro quartil (ou os percentis 75 e 25) é conhecida como intervalo interquartil, ou IQR. Assim como a mediana, essa quantidade será robusta para valores discrepantes, uma vez que valores grandes não a afetam. Podemos fazer alguns cálculos e observar que, para os dados que seguem a distribuição normal, o $IQR / 1.349$ se aproxima do desvio padrão dos dados se um erro externo não estiver presente. Podemos ver que isso funciona bem em nosso exemplo, pois obtemos uma estimativa do desvio padrão de:

```
IQR(outlier_example)/ 1.349
#> [1] 0.245
```

que fica perto de 3 polegadas.

11.4 Definição de Tukey de um outlier

Em R, os pontos que ficam fora dos bigodes do box plot são chamados de outliers, uma definição que Tukey introduziu. O bigode superior termina no 75^{o} percentil mais $1,5 \times \text{IQR}$, enquanto o bigode inferior termina no 25^{o} percentil menos $1,5 \times \text{IQR}$. Se definirmos o primeiro e o terceiro quartis como Q_1 e Q_3 , respectivamente, um valor externo é qualquer valor fora do intervalo:

$$[Q_1 - 1.5 \times (Q_3 - Q_1), Q_3 + 1.5 \times (Q_3 - Q_1)].$$

Quando os dados são normalmente distribuídos, as unidades padrão desses valores são:

```
q3 <- qnorm(0.75)
q1 <- qnorm(0.25)
iqr <- q3 - q1
r <- c(q1 - 1.5*iqr, q3 + 1.5*iqr)
r
#> [1] -2.7  2.7
```

Usando a função `pnorm`, nós vemos que 99.3% de dados cai nesse intervalo.

Observe que este não é um evento tão extremo: se tivermos 1000 pontos de dados normalmente distribuídos, esperamos ver cerca de 7 fora desse intervalo. Mas estes não seriam discrepantes, como esperamos vê-los sob variação típica.

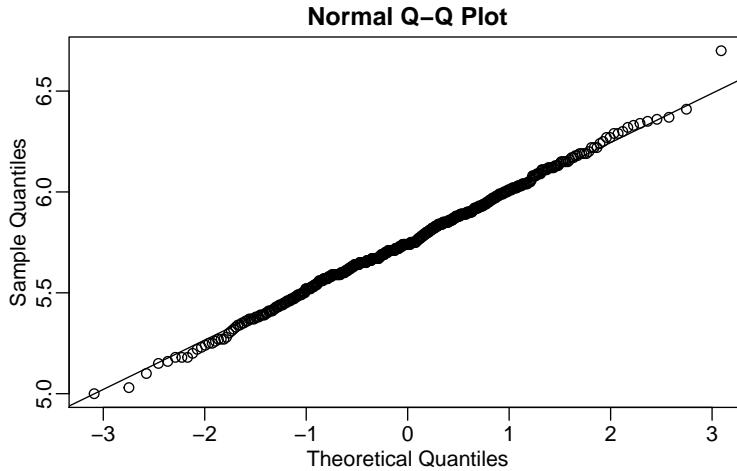
Se queremos que um outlier seja mais estranho, podemos mudar 1,5 para um número maior. Tukey também usou 3 e os chamou de outliers extremos ou outliers extremos. Com uma distribuição normal, 100% dos dados cai nesse intervalo. Isso se traduz em cerca de 2 em um milhão de chances de estar fora de alcance. Na função `geom_boxplot`, isso pode ser controlado usando o argumento `outlier.size`, que por padrão é 1,5.

A medição de 180 polegadas está além da faixa dos dados de altura:

```
max_height <- quantile(outlier_example, 0.75) + 3*IQR(outlier_example)
max_height
#> 75%
#> 6.91
```

Se removermos esse valor, podemos ver que os dados são normalmente distribuídos conforme o esperado:

```
x <- outlier_example[outlier_example < max_height]
qqnorm(x)
qqline(x)
```



11.5 Desvio absoluto mediano

Outra opção para estimar o desvio padrão de maneira robusta na presença de outliers é usar o desvio absoluto médio, ou MAD. Para calcular o MAD, primeiro calculamos a mediana e, em seguida, para cada valor, calculamos a distância entre esse valor e a mediana. MAD é definido como a mediana dessas distâncias. Por razões técnicas não discutidas aqui, esse valor deve ser multiplicado por 1,4826 para garantir que ele se aproxime do desvio padrão real. A função `mad` já incorpora essa correção. Para os dados de altura, obtemos um MAD de:

```
mad(outlier_example)
#> [1] 0.237
```

que fica perto de 3 polegadas.

11.6 Exercícios

Nós vamos usar o pacote **HistData**. Se você não o instalou, pode fazê-lo assim:

```
install.packages("HistData")
```

Carregue o conjunto de dados de altura e crie um vetor `x` ele contém apenas as alturas masculinas dos dados de Galton sobre pais e filhos de suas pesquisas históricas sobre herança.

```
library(HistData)
data(Galton)
x <- Galton$child
```

1. Calcule a média e a mediana desses dados.
2. Calcule a mediana e o MAD desses dados.

3. Agora, suponha que Galton tenha cometido um erro ao inserir o primeiro valor e esqueceu de usar o ponto decimal. Você pode imitar esse erro digitando:

```
x_with_error <- x
x_with_error[1] <- x_with_error[1]*10
```

Quantas polegadas a média cresce como resultado desse erro?

4. Quantas polegadas o SD cresce como resultado desse erro?
5. Quantas polegadas a mediana cresce como resultado desse erro?
6. Quantas polegadas o MAD cresce como resultado desse erro?

7. Como poderíamos usar a análise exploratória de dados para detectar que um erro foi cometido?

para. Como é apenas um valor entre muitos, isso não pode ser detectado. b. Veríamos uma mudança óbvia na distribuição. c. Um gráfico de caixa, histograma ou QQ revelaria um erro óbvio. d. Um diagrama de dispersão mostraria altos níveis de erro de medição.

8. Quanto a média pode crescer acidentalmente com erros como esse? Escreva uma função chamada `error_avg` que leva um valor `k` e retorna a média do vetor `x` após a primeira entrada mudar para `k`. Mostrar resultados para `k=10000` e `k=-10000`.

11.7 Estudo de caso: altura do aluno autorreferida

As alturas que estudamos não são as alturas originais relatadas pelos alunos. As alturas originais também estão incluídas no pacote `dslabs` e podem ser carregadas assim:

```
library(dslabs)
data("reported_heights")
```

`Height` é um vetor de caracteres, por isso criamos uma nova coluna com a versão numérica:

```
reported_heights <- reported_heights %>%
  mutate(original_heights = height, height = as.numeric(height))
#> Warning: Problem with `mutate()` `input` `height` .
#> i NAs introduced by coercion
#> i Input `height` is `as.numeric(height)` .
#> Warning in mask$eval_all_mutate(dots[[i]]): NAs introduced by coercion
```

Observe que recebemos um aviso sobre NAs. Isso ocorre porque algumas das alturas autorreferidas não eram números. Podemos ver por que temos essas NAs:

```
reported_heights %>% filter(is.na(height)) %>% head()
#>           time_stamp   sex height original_heights
#> 1 2014-09-02 15:16:28 Male     NA      5' 4"
#> 2 2014-09-02 15:16:37 Female   NA    165cm
#> 3 2014-09-02 15:16:52 Male     NA      5'7
#> 4 2014-09-02 15:16:56 Male     NA    >9000
#> 5 2014-09-02 15:16:56 Male     NA      5'7"
#> 6 2014-09-02 15:17:09 Female   NA      5'3"
```

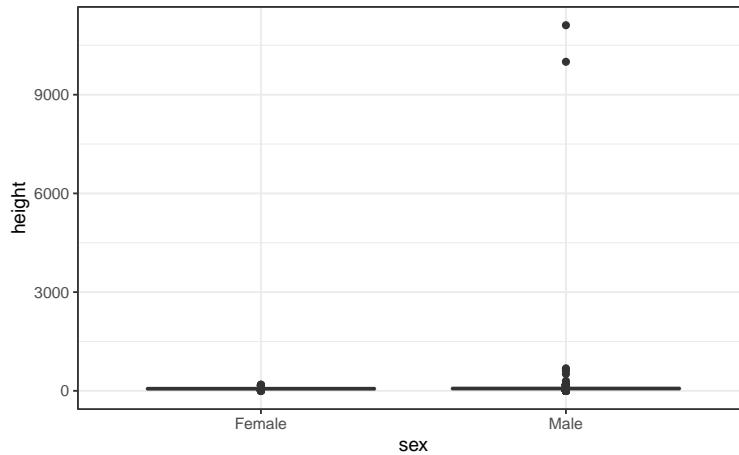
Alguns alunos relataram suas alturas usando pés e polegadas em vez de apenas polegadas. Outros usavam centímetros e outros estavam apenas trollando. Por enquanto, removeremos essas entradas:

```
reported_heights <- filter(reported_heights, !is.na(height))
```

Se calcularmos a média e o desvio padrão, observamos que obtemos resultados estranhos. A média e o desvio padrão são diferentes da mediana e do MAD:

```
reported_heights %>%
  group_by(sex) %>%
  summarize(average = mean(height), sd = sd(height),
            median = median(height), MAD = mad(height))
#> `summarise()` ungrouping output (override with `^.groups` argument)
#> # A tibble: 2 x 5
#>   sex     average    sd median    MAD
#>   <chr>     <dbl> <dbl> <dbl> <dbl>
#> 1 Female     63.4  27.9  64.2  4.05
#> 2 Male       103.   530.   70     4.45
```

Isso sugere que temos discrepâncias, o que é confirmado pela criação de um gráfico de caixa:



Vemos alguns valores extremos. Para ver quais são esses valores, podemos analisar rapidamente os valores maiores usando a função `arrange`:

```
reported_heights %>% arrange(desc(height)) %>% top_n(10, height)
#> #>           time_stamp      sex height original_heights
#> 1  2014-09-03 23:55:37  Male   11111          11111
#> 2  2016-04-10 22:45:49  Male   10000          10000
#> 3  2015-08-10 03:10:01  Male    684           684
#> 4  2015-02-27 18:05:06  Male    612           612
#> 5  2014-09-02 15:16:41  Male    511           511
#> 6  2014-09-07 20:53:43  Male    300           300
#> 7  2014-11-28 12:18:40  Male    214           214
#> 8  2017-04-03 16:16:57  Male    210           210
#> 9  2015-11-24 10:39:45  Male    192           192
#> 10 2014-12-26 10:00:12  Male    190           190
#> 11 2016-11-06 10:21:02 Female   190           190
```

As primeiras sete entradas parecem erros estranhos. No entanto, as seguintes entradas parecem ter sido inseridas em centímetros em vez de polegadas. Como 184 cm equivale a seis pés de altura, suspeitamos que 184 signifique 72 polegadas.

Podemos revisar todas as respostas sem sentido examinando os dados que Tukey considera *far out_ou_extremos*:

```
whisker <- 3*IQR(reported_heights$height)
max_height <- quantile(reported_heights$height, .75) + whisker
min_height <- quantile(reported_heights$height, .25) - whisker
reported_heights %>%
  filter(!between(height, min_height, max_height)) %>%
  select(original_heights) %>%
  head(n=10) %>% pull(original_heights)
#> [1] "6"      "5.3"    "511"   "6"      "2"      "5.25"   "5.5"    "11111"
#> [9] "6"      "6.5"
```

Verificando cuidadosamente essas alturas, vemos dois erros comuns: entradas em centímetros, que se tornam muito grandes, e entradas do tipo x.y com x e y representando pés e polegadas, respectivamente, que acabam sendo muito pequenos. Alguns dos valores ainda menores, como 1.6, podem ser entradas em metros.

Na parte *data wrangling* deste livro, aprenderemos técnicas para corrigir esses valores e convertê-los em polegadas. Aqui, conseguimos detectar esse problema explorando cuidadosamente os dados para descobrir problemas com eles - o primeiro passo na grande maioria dos projetos de ciência de dados.

Part III

Estatísticas com R

12

Introdução à estatística com R

Análise de dados é um dos principais focos deste livro. Embora as ferramentas computacionais que introduzimos sejam desenvolvimentos relativamente recentes, a análise de dados existe há mais de um século. Ao longo dos anos, analistas de dados trabalhando em projetos específicos criaram ideias e conceitos que podem ser generalizados para muitas aplicações. Eles também identificaram formas comuns de ser enganado por padrões aparentes nos dados e por realidades matemáticas importantes que não são imediatamente óbvias. O acúmulo dessas ideias e percepções deu origem à disciplina da estatística, que oferece uma estrutura matemática para facilitar a descrição formal e a avaliação de tais ideias.

Para evitar repetir erros comuns e perder tempo reinventando a roda, é importante que os analistas de dados tenham um profundo entendimento de estatística. Entretanto, devido à maturidade da disciplina, existem dezenas de excelentes livros já publicados sobre esse assunto e, portanto, não nos concentraremos em descrever a estrutura matemática aqui. Em vez disso, apresentamos brevemente conceitos e, em seguida, oferecemos estudos de caso que demonstram como as estatísticas são usadas na análise de dados, juntamente com o código em R que implementa essas ideias. Também usamos o código em R para ajudar a esclarecer alguns dos principais conceitos estatísticos geralmente descritos usando a matemática. Recomendamos fortemente complementar esta parte do livro com um manual básico de estatística. Dois exemplos são *Statistics* de Freedman, Pisani e Purves e *Statistical Inference* de Casella e Berger. Os conceitos específicos que discutimos nesta parte do livro são Probabilidade, Inferência Estatística, Modelos Estatísticos, Regressão e Modelos Lineares, que são os principais tópicos abordados em um curso de estatística. Os estudos de caso que apresentamos referem-se à crise financeira, previsão de resultados das eleições, entendendo a hereditariedade e como construir um time de beisebol.

13

Probabilidade

Nos jogos de azar, a probabilidade tem uma definição muito intuitiva. Por exemplo, sabemos o que significa quando dizemos que a probabilidade de um par de dados rolar sete é 1 em 6. No entanto, esse não é o caso em outros contextos. Hoje, a teoria das probabilidades é usada de maneira muito mais ampla com a palavra “probabilidade” e parte da linguagem cotidiana. Se digitarmos “Quais são as probabilidades de” no Google, a função de preenchimento automático nos fornecerá: “ter gêmeos”, “ter gêmeos” e “ganhar na loteria”. Um dos objetivos desta parte do livro é ajudá-lo a entender como a probabilidade é útil para entender e descrever eventos do mundo real quando realizamos análises de dados.

Desde que saber calcular as probabilidades oferece uma vantagem no jogo, muitas pessoas inteligentes ao longo da história, incluindo matemáticos famosos como Cardano, Fermat e Pascal, gastaram tempo e energia pensando na matemática desses jogos. . Como resultado, nasceu a teoria das probabilidades. A probabilidade ainda é muito útil nos jogos de azar modernos. Por exemplo, no pôquer, podemos calcular a probabilidade de ganhar uma mão com base nas cartas da mesa. Além disso, os cassinos confiam na teoria das probabilidades para desenvolver jogos que quase sempre garantem ganhos.

A teoria da probabilidade é útil em muitos outros contextos e, em particular, em áreas que de alguma forma dependem dos dados afetados pelo acaso. Todos os outros capítulos desta parte são baseados na teoria das probabilidades. Portanto, o conhecimento da probabilidade é indispensável para a ciência de dados.

13.1 Probabilidade discreta

Começamos explorando alguns princípios básicos relacionados aos dados categóricos. Essa parte da probabilidade é conhecida como a probabilidade discreta. Isso nos ajudará a entender a teoria das probabilidades que apresentaremos posteriormente para dados numéricos e contínuos, muito mais comuns em aplicativos de ciência de dados. Probabilidade discreta é mais útil em jogos de cartas e, portanto, nós as usamos como exemplos.

13.1.1 Frequência relativa

Embora a palavra probabilidade seja usada na linguagem cotidiana, é difícil responder a perguntas sobre probabilidade, se não impossível, porque o conceito de “probabilidade” não está bem definido. Aqui discutimos uma definição matemática de *probability* que nos permite dar respostas precisas a certas perguntas.

Por exemplo, se eu tiver 2 bolinhas vermelhas e 3 bolinhas azuis dentro de uma urna¹

¹https://en.wikipedia.org/wiki/Urn_problem

(muitos livros de probabilidade usam esse termo arcaico, também o fazemos) e escolha um aleatoriamente, qual é a probabilidade de escolher um vermelho? Nossa intuição nos diz que a resposta é $2/5$ ou 40%. Uma definição precisa pode ser dada observando que existem cinco resultados possíveis, dos quais dois atendem à condição necessária para o evento “escolha um mármore vermelho”. Dado que cada um dos cinco resultados tem a mesma probabilidade de ocorrência, concluímos que a probabilidade é de 0,4 para vermelho e 0,6 para azul.

Uma maneira mais tangível de pensar sobre a probabilidade de um evento é a proporção de vezes que o evento ocorre quando repetimos o experimento um número infinito de vezes, independentemente e sob as mesmas condições.

13.1.2 Notação

Usamos a notação $\Pr(A)$ para denotar a probabilidade de evento acontecer A . Usamos o termo geral *event* para nos referirmos a coisas que podem acontecer quando algo acontece por acaso. No exemplo anterior, o evento foi “escolha um mármore vermelho”. Em uma pesquisa política na qual chamamos aleatoriamente 100 prováveis eleitores americanos, um exemplo de evento é “ligar para 48 democratas e 52 republicanos”.

Em aplicativos de ciência de dados, frequentemente trabalharemos com variáveis contínuas. Esses eventos costumam ser coisas como “essa pessoa tem mais de um metro e oitenta”? Nesse caso, escrevemos eventos de uma forma mais matemática: $X \geq 6$. Veremos mais desses exemplos abaixo. Aqui nos concentramos em dados categóricos.

13.1.3 Distribuições de probabilidade

Se conhecermos a frequência relativa das diferentes categorias, definir uma distribuição para resultados categóricos é relativamente simples. Simplesmente atribuímos uma probabilidade a cada categoria. Nos casos que podem ser considerados bolinhas de gude em uma urna, para cada tipo de mármore, sua proporção define a distribuição.

Se estivermos chamando aleatoriamente prováveis eleitores de uma população que é 44% democrata, 44% republicana, 10% indecisa e 2% Partido Verde, essas proporções definem a probabilidade de cada grupo. A distribuição de probabilidade é:

$\Pr(\text{escolha um republicano})$	=	0,44
$\Pr(\text{escolha um democrata})$	=	0,44
$\Pr(\text{escolha indecisa})$	=	0,10
$\Pr(\text{escolha um verde})$	=	0,02

Simulações de Monte Carlo para dados categóricos

Os computadores oferecem uma maneira de realizar o experimento aleatório simples descrito acima: escolhendo um mármore aleatoriamente a partir de uma urna que contém três bolinhas azuis e duas vermelhas. Geradores de números aleatórios nos permitem imitar o processo de escolha aleatória.

Um exemplo é a função `sample` em R. Demonstramos seu uso no código abaixo. Primeiro, usamos a função `rep` para gerar a urna:

```
beads <- rep(c("red", "blue"), times = c(2,3))
beads
#> [1] "red"   "red"   "blue"  "blue"  "blue"
```

e depois usamos `sample` escolher um mármore aleatoriamente:

```
sample(beads, 1)
#> [1] "blue"
```

Essa linha de código produz um resultado aleatório. Queremos repetir esse experimento um número infinito de vezes, mas é impossível repeti-lo para sempre. No entanto, podemos repetir o experimento um número suficientemente grande de vezes para que os resultados sejam praticamente equivalentes a repeti-lo para sempre. **Este é um exemplo de simulação de Monte Carlo.**

Muito do que estudam estatísticos matemáticos e teóricos, que não discutimos neste livro, refere-se a fornecer definições rigorosas de “virtualmente equivalente”, além de estudar o quão perto um grande número de experimentos nos leva ao que acontece no limite. Mais adiante nesta seção, oferecemos uma abordagem prática para determinar o que é “grande o suficiente”.

Para realizar nossa primeira simulação de Monte Carlo, usamos a função `replicate`, o que nos permite repetir a mesma tarefa várias vezes. Aqui, repetimos o evento aleatório $B = 10.000$ vezes:

```
B <- 10000
events <- replicate(B, sample(beads, 1))
```

Agora podemos ver se nossa definição realmente concorda com essa abordagem de simulação de Monte Carlo. Podemos usar `table` para ver a distribuição:

```
tab <- table(events)
tab
#> events
#> blue red
#> 5981 4019
```

e `prop.table` nos dá as proporções:

```
prop.table(tab)
#> events
#> blue red
#> 0.598 0.402
```

Os números acima são probabilidades estimadas fornecidas por uma simulação de Monte Carlo. A teoria estatística, que não discutimos aqui, nos diz que onde B à medida que aumenta, as estimativas se aproximam de $3/5 = 0,6$ e $2/5 = 0,4$.

Embora este seja um exemplo simples e pouco útil, usaremos simulações de Monte Carlo para estimar probabilidades nos casos em que é difícil calcular quantidades exatas. Antes de nos aprofundarmos em exemplos mais complexos, usaremos alguns simples para demonstrar as ferramentas de computação disponíveis em R.

13.1.4 Defina a semente aleatória

Antes de continuar, explicaremos brevemente a seguinte linha de código importante:

```
set.seed(1986)
```

Ao longo deste livro, usamos geradores de números aleatórios. Isso implica que muitos dos

resultados que apresentamos podem mudar por acaso e uma versão congelada do livro pode mostrar um resultado diferente do que eles obtêm quando tentam codificar como observam no livro. Isso não é um problema, pois os resultados são aleatórios e podem mudar. No entanto, se você quiser garantir que os resultados sejam exatamente os mesmos sempre que executá-los, poderá definir a semente de geração de número aleatório R (*seed*) para um número específico. Definimos isso em 1986. Queremos evitar o uso da mesma semente todas as vezes. Uma maneira popular de escolher a semente é subtraindo o mês e o dia do ano. Por exemplo, para 20 de dezembro de 2018, lançamos a semente em 1986: $2018 - 12 - 20 = 1986$.

Você pode obter mais informações sobre como corrigir a semente consultando a documentação:

```
?set.seed
```

Nos exercícios, podemos pedir que você conserte a semente para garantir que seus resultados sejam exatamente o que esperamos.

13.1.5 Com e sem substituição

A função `sample` ele tem um argumento que nos permite escolher mais de um item da urna. No entanto, por padrão, essa seleção ocorre *sem substituição*: após a seleção de uma bola de gude, ela não é colocada de volta na urna. Observe o que acontece quando pedimos para selecionar cinco bolas de gude aleatoriamente:

```
sample(beads, 5)
#> [1] "red"  "blue" "blue" "blue" "red"
sample(beads, 5)
#> [1] "red"  "red"  "blue" "blue" "blue"
sample(beads, 5)
#> [1] "blue" "red"  "blue" "red"  "blue"
```

Isso resulta em rearranjos que sempre têm três bolinhas azuis e duas vermelhas. Se pedirmos seis bolas de gude para serem selecionadas, obteremos um erro:

```
sample(beads, 6)
```

```
Error in sample.int(length(x), size, replace, prob) : cannot take a sample
larger than the population when 'replace = FALSE'
```

No entanto, a função `sample` pode ser usado diretamente, sem o uso de `replicate`, para repetir o mesmo experimento, escolhendo 1 dos 5 marmores, continuamente, nas mesmas condições. Para fazer isso, amostramos *com substituição*: o mármore é devolvido à urna após a seleção. Nós podemos dizer `sample` eu fazer isso mudando o argumento `replace` que por padrão é `FALSE`, para `replace = TRUE`:

```
events <- sample(beads, B, replace = TRUE)
prop.table(table(events))
#> events
#> blue   red
#> 0.602 0.398
```

Não surpreende que obtenham resultados muito semelhantes aos obtidos anteriormente com `replicate`.

13.2 Independência

Dizemos que dois eventos são independentes se o resultado de um não afeta o outro. O exemplo clássico é o lançamento de moedas. Toda vez que jogamos uma moeda, a probabilidade de ver caras é $1/2$, independentemente dos resultados dos lançamentos anteriores. O mesmo acontece quando coletamos bolas de gude de uma urna de substituição. No exemplo acima, a probabilidade de vermelho é $0,40$, independentemente das seleções anteriores.

Muitos exemplos de eventos não independentes vêm de jogos de cartas. Quando negociamos a primeira carta, a probabilidade de obter um K é $1/13$, pois há treze possibilidades: Duas, Três, ..., Ten, J, Q, K e As. Mas se dermos um K como primeira carta e não a substituirmos no baralho, a probabilidade de uma segunda carta ser K é menor, porque só restam três Ks: a probabilidade é $3/51$. Portanto, esses eventos **não são independentes**: o primeiro resultado afeta o seguinte.

Para ver um caso extremo de eventos não independentes, considere nosso exemplo de escolha de cinco bolinhas aleatoriamente **sem substituição**:

```
x <- sample(beads, 5)
```

Se eles tiverem que adivinhar a cor do primeiro mármore, irão prever o azul, pois o azul tem 60% de chance. Mas se mostrarmos o resultado dos últimos quatro resultados:

```
x[2:5]
#> [1] "blue" "blue" "blue" "red"
```

você ainda adivinharia o azul? Claro que não. Agora você sabe que a probabilidade de vermelho é 1 , pois o único mármore restante é vermelho. Os eventos não são independentes, portanto as probabilidades mudam.

13.3 Probabilidades condicionais

Quando os eventos não são independentes, as *probabilidades condicionais* são úteis. Já vimos um exemplo de probabilidade condicional: calculamos a probabilidade de que uma segunda carta seja K desde que a primeira carta fosse K. Em probabilidade, usamos a seguinte notação:

$$\Pr(\text{Card 2 is a king} \mid \text{Card 1 is a king}) = 3/51$$

Nós usamos o \mid como uma abreviação de “dado isso” ou “condicional”.

Quando dois eventos, digamos A e B , eles são independentes, temos:

$$\Pr(A \mid B) = \Pr(A)$$

Esta é a maneira matemática de dizer: o fato de que B aconteceu não afeta a probabilidade de A acontecer. De fato, isso pode ser considerado a definição matemática de independência.

13.4 Regras de adição e multiplicação

Regra de multiplicação

Se queremos saber a probabilidade de dois eventos ocorrerem, digamos A e B , podemos usar a regra de multiplicação:

$$\Pr(A \text{ and } B) = \Pr(A)\Pr(B | A)$$

Vamos usar o jogo de cartas do Blackjack como exemplo. No Blackjack, eles recebem duas cartas aleatoriamente. Depois de ver o que têm, podem pedir mais cartões. O objetivo é chegar mais perto dos 21 do que o revendedor, sem passar. Os *face cards* valem 10 pontos e os ases valem 11 ou 1 (um escolhe).

Portanto, no Blackjack, para calcular as probabilidades de conseguir um 21 recebendo um ás e depois uma carta de face, calculamos a probabilidade de a primeira carta ser um ás e multiplicamos pela probabilidade de tirar uma carta de cara ou 10, dado que o primeiro foi um ás: $1/13 \times 16/51 \approx 0.025$.

A regra de multiplicação também se aplica a mais de dois eventos. Podemos usar a indução para incluir mais eventos:

$$\Pr(A \text{ and } B \text{ and } C) = \Pr(A)\Pr(B | A)\Pr(C | A \text{ and } B)$$

Regra de multiplicação sob independência

Quando temos eventos independentes, a regra de multiplicação se torna mais simples:

$$\Pr(A \text{ and } B \text{ and } C) = \Pr(A)\Pr(B)\Pr(C)$$

Mas devemos ter muito cuidado antes de usar isso, pois assumir a independência quando ela realmente não existe pode resultar em cálculos de probabilidade muito diferentes e incorretos.

Como exemplo, imagine um caso em que o suspeito seja descrito como tendo bigode e barba. O réu tem bigode e barba e a acusação traz um “especialista” que testemunha que $1/10$ dos homens têm barba e $1/5$ têm bigode, portanto, usando a regra da multiplicação, concluímos que apenas $1/10 \times 1/5$ ou $0,02$ tem ambos.

Mas para se multiplicar assim, precisamos assumir a independência! Digamos que a probabilidade condicional de um homem ter um bigode condicionado no qual ele tem barba é $0,95$. Portanto, o cálculo correto da probabilidade resulta em um número muito maior: $1/10 \times 95/100 = 0.095$.

A regra de multiplicação também nos fornece uma fórmula geral para calcular probabilidades condicionais:

$$\Pr(B | A) = \frac{\Pr(A \text{ and } B)}{\Pr(A)}$$

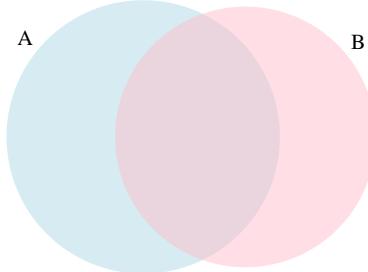
Para ilustrar como usamos essas fórmulas e conceitos na prática, usaremos vários exemplos relacionados a jogos de cartas.

Regra de adição

A regra de adição nos diz que:

$$\Pr(A \text{ or } B) = \Pr(A) + \Pr(B) - \Pr(A \text{ and } B)$$

Essa regra é intuitiva: pense em um diagrama de Venn. Se simplesmente adicionarmos as probabilidades, contaremos a interseção duas vezes, portanto, devemos subtrair uma instância.



13.5 Combinações e permutações

Em nosso primeiro exemplo, imaginamos uma urna com cinco bolinhas de gude. Lembre-se de que, para calcular a distribuição de probabilidade de um empate, simplesmente listamos todas as probabilidades. Havia cinco e, portanto, para cada evento, contamos quantas dessas probabilidades estavam associadas ao evento. A probabilidade de escolher um mármore azul é de 3/5, devido aos cinco resultados possíveis, três eram azuis.

Para casos mais complicados, os cálculos não são tão simples. Por exemplo, qual é a probabilidade de que, se eu escolher cinco cartas sem substituição, receberei todas as cartas do mesmo naipe (*suit* em inglês), conhecidas como “flush” no poker? Em um curso de probabilidade discreta, você aprende a teoria de como fazer esses cálculos. Aqui, focamos em como usar o código R para calcular as respostas.

Primeiro, vamos construir um baralho de cartas. Para isso, usaremos as funções `expand.grid` e `paste`. Nós usamos `paste` para criar cadeias juntando cadeias menores. Para fazer isso, pegamos o número e o naipe de um cartão e criamos o nome do cartão assim:

```
number <- "Three"
suit <- "Hearts"
paste(number, suit)
#> [1] "Three Hearts"
```

`paste` também funciona em pares de vetores que executam a operação elemento a elemento:

```
paste(letters[1:5], as.character(1:5))
#> [1] "a 1" "b 2" "c 3" "d 4" "e 5"
```

A função `expand.grid` nos fornece todas as combinações de duas entradas de vetor. Por exemplo, se eles têm calças azuis e pretas e camisas brancas, cinza e xadrez (*plaid*), todas as suas combinações são:

```
expand.grid(pants = c("blue", "black"), shirt = c("white", "grey", "plaid"))
#>   pants shirt
#> 1 blue white
#> 2 black white
#> 3 blue grey
#> 4 black grey
#> 5 blue plaid
#> 6 black plaid
```

Aqui está como geramos um baralho de cartas:

```
suits <- c("Diamonds", "Clubs", "Hearts", "Spades")
numbers <- c("Ace", "Deuce", "Three", "Four", "Five", "Six", "Seven",
"Eight", "Nine", "Ten", "Jack", "Queen", "King")
deck <- expand.grid(number=numbers, suit=suits)
deck <- paste(deck$number, deck$suit)
```

Com o baralho construído, podemos verificar se a probabilidade de um K ser a primeira carta é 1/13 calculando a proporção de possíveis resultados que satisfazem nossa condição:

```
kings <- paste("King", suits)
mean(deck %in% kings)
#> [1] 0.0769
```

Agora, que tal a probabilidade condicional de que a segunda carta seja um K desde que a primeira carta fosse um K? Anteriormente, deduzimos que, se um K já estiver fora do baralho e restarem 51 cartas, a probabilidade será de 3/51. Vamos confirmar listando todos os resultados possíveis.

Para fazer isso, podemos usar a função `permutations` do pacote `gtools`. Para qualquer lista de tamanho `n`, essa função calcula todas as diferentes combinações que podemos obter quando selecionamos `r` artigos. Aqui estão todas as maneiras pelas quais podemos escolher dois números de uma lista que consiste em 1, 2, 3:

```
library(gtools)
permutations(3, 2)
#>      [,1] [,2]
#> [1,]    1    2
#> [2,]    1    3
#> [3,]    2    1
#> [4,]    2    3
#> [5,]    3    1
#> [6,]    3    2
```

Observe que a ordem é importante aqui: 3.1 é diferente de 1.3. Além disso, observe que (1,1), (2,2) e (3,3) não aparecem porque, quando escolhemos um número, ele não pode aparecer novamente.

Opcionalmente, podemos adicionar um vetor. Se desejar ver cinco números de telefone aleatórios (sete dígitos) de todos os números de telefone possíveis (sem repetição), você pode escrever:

```
all_phone_numbers <- permutations(10, 7, v = 0:9)
n <- nrow(all_phone_numbers)
```

```
index <- sample(n, 5)
all_phone_numbers[index,]
#> [,1] [,2] [,3] [,4] [,5] [,6] [,7]
#> [1,] 1 3 8 0 6 7 5
#> [2,] 2 9 1 6 4 8 0
#> [3,] 5 1 6 0 9 8 2
#> [4,] 7 4 6 0 2 8 1
#> [5,] 4 6 5 9 2 8 0
```

Em vez de usar os números de 1 a 10, o padrão, R usa o que fornecemos através v: os dígitos de 0 a 9.

Para calcular todas as maneiras possíveis, podemos escolher duas cartas quando a ordem importa, escrevemos:

```
hands <- permutations(52, 2, v = deck)
```

Esta é uma matriz com duas colunas e 2652 linhas. Com uma matriz, podemos obter a primeira e a segunda letras assim:

```
first_card <- hands[,1]
second_card <- hands[,2]
```

Agora, os casos para os quais a primeira letra é um K podem ser calculados assim:

```
kings <- paste("King", suits)
sum(first_card %in% kings)
#> [1] 204
```

Para obter a probabilidade condicional, calculamos qual fração deles tem um K como a segunda letra:

```
sum(first_card %in% kings & second_card %in% kings) / sum(first_card %in% kings)
#> [1] 0.0588
```

que é exatamente 3/51, como já deduzimos. Observe que o código acima é equivalente a:

```
mean(first_card %in% kings & second_card %in% kings) / mean(first_card %in% kings)
#> [1] 0.0588
```

o que você usa `mean` ao invés de `sum` e é uma versão R de:

$$\frac{\Pr(A \text{ and } B)}{\Pr(A)}$$

E se a ordem não importar? Por exemplo, no Blackjack, se você receber um Ás e uma carta de face como primeira mão, ela será chamada *Natural 21* e você ganhará automaticamente. Se quiséssemos calcular a probabilidade de isso acontecer, listaríamos as *combinações*, não as permutações, pois a ordem não importa.

```
combinations(3,2)
#> [,1] [,2]
#> [1,] 1 2
#> [2,] 1 3
#> [3,] 2 3
```

Na segunda linha, o resultado não inclui (2.1) porque (1.2) já foi listado. O mesmo se aplica a (3.1) e (3.2).

Portanto, para calcular a probabilidade de um *Natural 21*, podemos fazer o seguinte:

```
aces <- paste("Ace", suits)

facecard <- c("King", "Queen", "Jack", "Ten")
facecard <- expand.grid(number = facecard, suit = suits)
facecard <- paste(facecard$number, facecard$suit)

hands <- combinations(52, 2, v = deck)
mean(hands[,1] %in% aces & hands[,2] %in% facecard)
#> [1] 0.0483
```

Na última linha, assumimos que o ás é a primeira carta que recebemos. Sabemos disso porque, sabendo como `combination` listar as probabilidades, entendemos que você listará esse caso primeiro. Mas, com certeza, poderíamos ter produzido a mesma resposta escrevendo o seguinte:

```
mean((hands[,1] %in% aces & hands[,2] %in% facecard) |
(hands[,2] %in% aces & hands[,1] %in% facecard))
#> [1] 0.0483
```

Exemplo de Monte Carlo

Ao invés de usar `combinations` para deduzir a probabilidade exata de um *Natural 21*, podemos usar uma simulação de Monte Carlo para estimar essa probabilidade. Nesse caso, escolhemos duas cartas repetidamente e observamos quantos 21s temos. Nós podemos usar a função `sample` escolher duas placas sem substituições:

```
hand <- sample(deck, 2)
hand
#> [1] "Queen Clubs"  "Seven Spades"
```

E depois verifique se uma carta é um Ás e a outra uma figura ou 10. A partir de agora, incluímos 10 quando dizemos *figure card_ou_figure*. Agora precisamos verificar as duas probabilidades:

```
(hands[1] %in% aces & hands[2] %in% facecard) |
(hands[2] %in% aces & hands[1] %in% facecard)
#> [1] FALSE
```

Se repetirmos isso 10.000 vezes, obteremos uma aproximação muito boa da probabilidade de um *Natural 21*.

Vamos começar escrevendo uma função que escolhe uma mão e retorna VERDADEIRO se obtivermos 21. A função não precisa de argumentos porque usa objetos definidos no ambiente global.

```
blackjack <- function(){
hand <- sample(deck, 2)
(hand[1] %in% aces & hand[2] %in% facecard) |
(hand[2] %in% aces & hand[1] %in% facecard)
}
```

Aqui temos que verificar as duas probabilidades: Ás primeiro ou Ás segundo, porque não estamos usando a função `combinations`. A função retorna TRUE se tivermos 21 e FALSE de outra maneira:

```
blackjack()
#> [1] FALSE
```

Agora podemos jogar este jogo, digamos 10.000 vezes:

```
B <- 10000
results <- replicate(B, blackjack())
mean(results)
#> [1] 0.0475
```

13.6 Exemplos

Nesta seção, descrevemos dois exemplos populares de probabilidade discreta: o problema de Monty Hall e o problema do aniversário. Usamos R para ajudar a ilustrar conceitos matemáticos.

Problema de Monty Hall

Nos anos 1970, nos EUA, houve um programa de perguntas e respostas chamado “Let’s Make a Deal” e Monty Hall foi o apresentador. Em algum momento do jogo, o competidor foi convidado a escolher uma das três portas. Atrás de uma porta havia um prêmio, enquanto atrás das outras portas eles tinham uma cabra que indicava que o competidor havia perdido. Depois que o competidor escolheu uma porta, e antes de revelar se a porta continha um prêmio, Monty Hall abriu uma das outras duas portas e mostrou ao competidor que não havia prêmio atrás daquela porta. Então ele perguntou ao competidor: “Você quer trocar de porta?” O que você faria?

Podemos usar a probabilidade para mostrar que, se eles mantiverem a opção do portão original, suas chances de ganhar um prêmio permanecerão 1 em 3. No entanto, se mudarem para o outro portão, suas chances de ganhar o dobro para 2 em 3! ! Isso parece contraditório. Muitas pessoas pensam incorretamente que ambas as probabilidades são de 1 em 2, uma vez que se escolhe entre duas opções. Você pode ver uma explicação matemática detalhada na Khan Academy² ou leia uma na Wikipedia³. Em seguida, usamos uma simulação de Monte Carlo para ver qual é a melhor estratégia. Observe que este código é escrito com mais detalhes do que o necessário para fins pedagógicos.

Vamos começar com a estratégia de não trocar de porta:

```
B <- 10000
monty_hall <- function(strategy){
  doors <- as.character(1:3)
  prize <- sample(c("car", "goat", "goat"))
  prize_door <- doors[prize == "car"]
  my_pick <- sample(doors, 1)
```

²<https://www.khanacademy.org/math/precalculus/prob-comb/dependent-events-precalc/v/monty-hall-problem>

³https://en.wikipedia.org/wiki/Monty_Hall_problem

```

show <- sample(doors[!doors %in% c(my_pick, prize_door)], 1)
stick <- my_pick
stick == prize_door
switch <- doors[!doors %in% c(my_pick, show)]
choice <- ifelse(strategy == "stick", stick, switch)
choice == prize_door
}
stick <- replicate(B, monty_hall("stick"))
mean(stick)
#> [1] 0.342
switch <- replicate(B, monty_hall("switch"))
mean(switch)
#> [1] 0.668

```

Enquanto escrevemos o código, notamos que as linhas que começam com `my_pick` e `show` eles não afetam a última operação lógica quando mantemos nossa escolha original. A partir disso, devemos perceber que a probabilidade é de 1 em 3, a mesma com a qual começamos. Quando mudamos, a estimativa de Monte Carlo confirma o cálculo de 2/3. Isso nos ajuda a entender melhor o problema, mostrando que estamos removendo uma porta, `show`, que definitivamente não esconde um prêmio de nossas opções. Também vemos que, a menos que acertemos na primeira escolha, você ganha: $1 - 1/3 = 2/3$.

13.6.1 Problema de aniversário

Imagine que você está em uma sala de aula com 50 pessoas. Se assumirmos que este é um grupo de 50 pessoas selecionadas aleatoriamente, qual é a probabilidade de pelo menos duas pessoas terem o mesmo aniversário? Embora seja um pouco avançado, podemos deduzir isso matematicamente. Faremos isso mais tarde, mas aqui usamos uma simulação de Monte Carlo. Por simplicidade, assumimos que ninguém nasceu em 29 de fevereiro. Isso realmente não muda muito a resposta.

Primeiro, lembre-se de que aniversários podem ser representados como números entre 1 e 365, para que você possa obter uma amostra de 50 aniversários como este:

```

n <- 50
bdays <- sample(1:365, n, replace = TRUE)

```

Para verificar se neste conjunto específico de 50 pessoas, temos pelo menos dois com o mesmo aniversário, podemos usar a função `duplicated`, que retorna `TRUE` sempre que um elemento de um vetor for duplicado. Aqui está um exemplo:

```

duplicated(c(1,2,3,1,4,3,5))
#> [1] FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE

```

Na segunda vez que 1 e 3 aparecem, obtemos um `TRUE`. Portanto, para verificar se dois aniversários são iguais, simplesmente usamos as funções `any` e `duplicated` assim:

```

any(duplicated(bdays))
#> [1] TRUE

```

Nesse caso, vemos o que aconteceu. Pelo menos duas pessoas tiveram o mesmo aniversário.

Para estimar a probabilidade de um aniversário compartilhado no grupo, repetimos esse experimento amostrando conjuntos de 50 aniversários repetidamente:

```
B <- 10000
same_birthday <- function(n){
  bdays <- sample(1:365, n, replace=TRUE)
  any(duplicated(bdays))
}
results <- replicate(B, same_birthday(50))
mean(results)
#> [1] 0.969
```

Você esperava que a probabilidade fosse tão alta?

As pessoas tendem a subestimar essas probabilidades. Para ter uma idéia de por que é tão alto, pense no que acontece quando o tamanho do grupo se aproxima dos 365. Com o 365, estamos fora de dias e a probabilidade é uma.

Digamos que queremos usar esse conhecimento para apostar com os amigos se duas pessoas em um grupo têm o mesmo aniversário. Com um grupo de qual tamanho as probabilidades estão acima de 50%? Mais de 75%?

Vamos começar criando uma tabela de pesquisa. Podemos criar rapidamente uma função para calcular isso para qualquer tamanho de grupo:

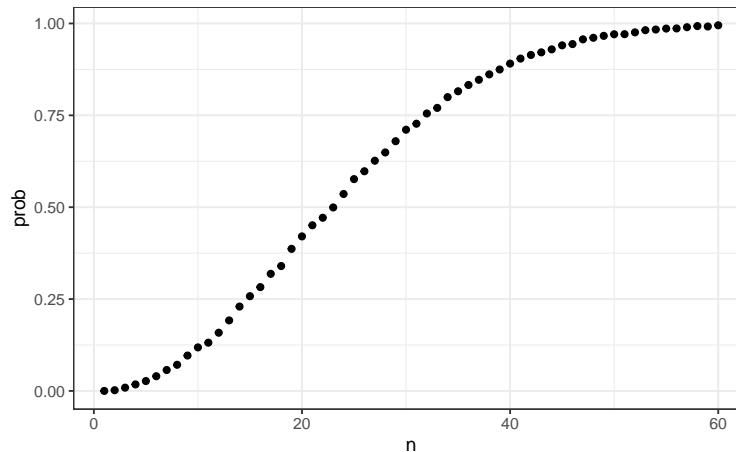
```
compute_prob <- function(n, B=10000){
  results <- replicate(B, same_birthday(n))
  mean(results)
}
```

Usando a função `sapply`, podemos executar operações elemento a elemento em qualquer função:

```
n <- seq(1,60)
prob <- sapply(n, compute_prob)
```

Agora podemos representar graficamente as probabilidades estimadas de duas pessoas que têm o mesmo aniversário em um grupo de tamanho n :

```
library(tidyverse)
prob <- sapply(n, compute_prob)
qplot(n, prob)
```



Agora vamos calcular as probabilidades exatas em vez de usar simulações de Monte Carlo. Não apenas obtemos a resposta exata usando a matemática, mas os cálculos são muito mais rápidos, pois não precisamos realizar experimentos.

Para simplificar a matemática, em vez de calcular a probabilidade de ocorrência, calcularemos a probabilidade de não ocorrer. Para isso, usamos a regra de multiplicação.

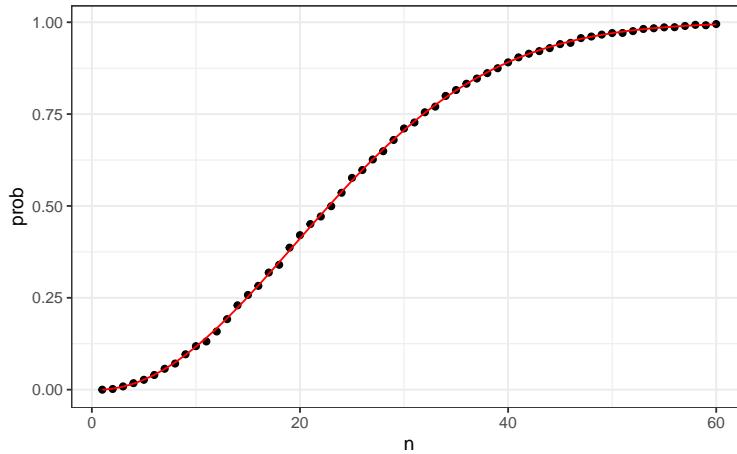
Vamos começar com a primeira pessoa. A probabilidade de que a Pessoa 1 tenha um aniversário único é 1. A probabilidade de que a Pessoa 2 tenha um aniversário único, considerando que a Pessoa 1 já foi atribuída por dia, é 364/365. Então, como as duas primeiras pessoas têm aniversários únicos, a pessoa 3 tem 363 dias para escolher. Continuamos assim e descobrimos que as chances de todas as 50 pessoas terem um aniversário único são:

$$1 \times \frac{364}{365} \times \frac{363}{365} \dots \frac{365-n+1}{365}$$

Podemos escrever uma função que faça isso para qualquer número:

```
exact_prob <- function(n){
  prob_unique <- seq(365, 365-n+1)/365
  1 - prod(prob_unique)
}

eprob <- sapply(n, exact_prob)
qplot(n, prob) + geom_line(aes(n, eprob), col = "red")
```



Este gráfico mostra que a simulação de Monte Carlo fornece uma estimativa muito boa da probabilidade exata. Se não fosse possível calcular as probabilidades exatas, ainda poderíamos estimar com precisão as probabilidades.

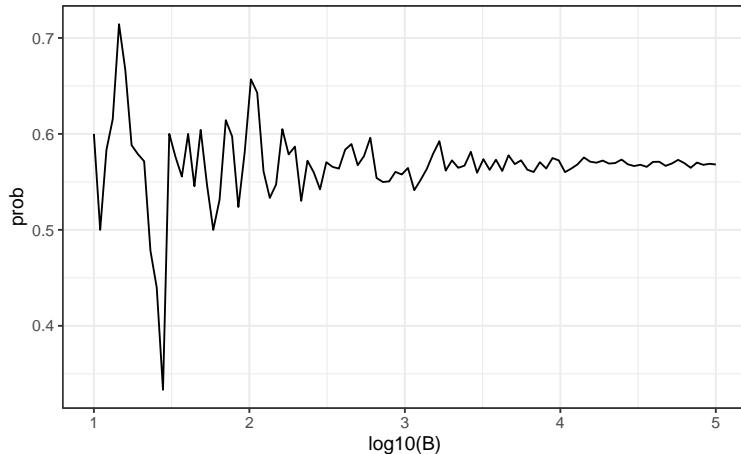
Infinito na prática

A teoria descrita aqui requer repetir experimentos repetidamente para sempre. Na prática, não podemos fazer isso. Nos exemplos acima, usamos $B = 10,000$ experimentos em Monte Carlo, e isso nos deu estimativas precisas. Quanto maior esse número, mais precisa será a estimativa até que a aproximação seja tão boa que seus computadores não consigam perceber a diferença. Mas em cálculos mais complexos, 10.000 podem ser insuficientes. Além disso, para alguns cálculos, 10.000 experimentos podem não ser computacionalmente viáveis. Na prática, não saberemos qual é a resposta, portanto, não saberemos se nossa estimativa de Monte Carlo é precisa. Sabemos que quanto maior é B , melhor será a aproximação. Mas quão

grande precisamos que seja? Essa é realmente uma pergunta desafiadora e, para respondê-la com frequência, é necessário treinamento avançado em estatística teórica.

Uma abordagem prática que descreveremos aqui é verificar a estabilidade da estimativa. Aqui está um exemplo do problema de aniversário para um grupo de 25 pessoas.

```
B <- 10^seq(1, 5, len = 100)
compute_prob <- function(B, n=25){
  same_day <- replicate(B, same_birthday(n))
  mean(same_day)
}
prob <- sapply(B, compute_prob)
qplot(log10(B), prob, geom = "line")
```



Neste gráfico, podemos ver que os valores começam a se estabilizar (ou seja, eles variam menos de 0,01) em torno de 1000. Observe que a probabilidade exata, que neste caso sabemos, é 0.569.

13.7 Exercícios

1. Um mármore é escolhido aleatoriamente a partir de uma caixa contendo: 3 bolinhas de ciano, 5 bolinhas de magenta e 7 bolinhas de gude amarelas. Qual é a probabilidade de o mármore ser ciano?
2. Qual é a probabilidade de o mármore não ser ciano?
3. Em vez de escolher apenas uma bola de gude, escolha duas bolas de gude. Retire o primeiro mármore sem devolvê-lo à caixa. Esta é uma amostra **sem** substituição. Qual é a probabilidade de o primeiro mármore ser ciano e o segundo mármore não ser ciano?
4. Agora repita o experimento, mas desta vez, depois de tirar o primeiro mármore e anotar a cor, coloque-o novamente na caixa e agite a caixa. Esta é uma amostra **com** substituição. Qual é a probabilidade de o primeiro mármore ser ciano e o segundo mármore não ser ciano?
5. Dois eventos A e B são independentes se $\Pr(A \text{ and } B) = \Pr(A)\Pr(B)$. Em que situação a seleção é independente?

para. Não substitui o artigo selecionado. b. Substitua o item selecionado. c. Nenhum. d. Ambos.

6. Digamos que você removeu 5 bolinhas de gude da caixa, com reposição, e todas elas foram amarelas. Qual é a probabilidade de o próximo ser amarelo?

7. Se você rolar um dado de 6 lados seis vezes, qual é a probabilidade de não ver um 6?

8. Dois times de basquete, dizem os Celtics e os Cavs, estão jogando uma série de sete jogos. Os Cavs são um time melhor e têm 60% de chance de vencer cada jogo. Qual é a probabilidade de o Celtics vencer **pelo menos** um jogo?

9. Crie uma simulação de Monte Carlo para confirmar sua resposta ao problema anterior. Usar $B \leftarrow 10000$ simulações. Dica: use o seguinte código para gerar os resultados dos quatro primeiros jogos:

```
celtic_wins <- sample(c(0,1), 4, replace = TRUE, prob = c(0.6, 0.4))
```

O Celtics deve vencer um desses 4 jogos.

10. Dois times de basquete, dizem os Cavs e Warriors, estão jogando uma série de sete jogos do campeonato. O primeiro a vencer quatro jogos, portanto, vence a série. As equipes são igualmente boas, então cada uma tem uma chance de 50 a 50 de vencer cada jogo. Se os Cavs perderem o primeiro jogo, qual é a probabilidade de ganharem a série?

Onze. Confirme os resultados da pergunta anterior com uma simulação de Monte Carlo.

12. Duas equipes A e B , eles estão jogando uma série de sete jogos. Equipamento A é melhor que equipe B e tem um $p > 0.5$ probabilidade de ganhar cada jogo. Dado um valor p , a probabilidade de o time não favorito B a série win pode ser calculada com a seguinte função com base em uma simulação de Monte Carlo:

```
prob_win <- function(p){
  B <- 10000
  result <- replicate(B, {
    b_win <- sample(c(1,0), 7, replace = TRUE, prob = c(1-p, p))
    sum(b_win)>=4
  })
  mean(result)
}
```

Usar função `sapply` para calcular a probabilidade, chame-o Pr ganhar por $p \leftarrow \text{seq}(0.5, 0.95, 0.025)$. Em seguida, faça um gráfico do resultado.

13. Repita o exercício anterior, mas agora mantenha a probabilidade fixa em $p \leftarrow 0.75$ e calcule a probabilidade de diferentes números de jogos necessários para concluir a série: ganhe 1 jogo, ganhe 2 de 3 jogos, ganhe 3 de 5 jogos, ... Especificamente, $N \leftarrow \text{seq}(1, 25, 2)$. Dica: use esta função:

```
prob_win <- function(N, p=0.75){
  B <- 10000
  result <- replicate(B, {
    b_win <- sample(c(1,0), N, replace = TRUE, prob = c(1-p, p))
    sum(b_win)>=(N+1)/2
  })
  mean(result)
}
```

13.8 Probabilidade contínua

Na seção 8.4 explicamos por que, resumindo uma lista de valores numéricos, como alturas, não é útil construir uma distribuição que defina uma proporção para cada resultado possível. Por exemplo, suponha que medimos cada pessoa em uma grande população, digamos em tamanho n , com precisão extremamente alta. Como não há duas pessoas exatamente da mesma altura, devemos atribuir a proporção $1/n$ em cada valor observado e, como consequência, nenhum resumo útil é obtido. Da mesma forma, ao definir distribuições de probabilidade, não é útil atribuir uma probabilidade muito pequena para cada altura.

Assim como quando as distribuições são usadas para resumir dados numéricos, é muito mais prático definir uma função que opera em intervalos, em vez de valores individuais. A maneira padrão de fazer isso é usar a *Cumulated Distribution Function*, ou CDF.

Descrevemos a função de distribuição cumulativa empírica, ou eCDF, na Seção 8.4 como um resumo básico de uma lista de valores numéricos. Como exemplo, definimos anteriormente a distribuição de altura para estudantes adultos do sexo masculino. Aqui nós definimos o vetor x para conter essas alturas:

```
library(tidyverse)
library(dslabs)
data(heights)
x <- heights %>% filter(sex=="Male") %>% pull(height)
```

Definimos a função de distribuição cumulativa empírica como:

```
F <- function(a) mean(x<=a)
```

isso por qualquer valor a , fornece a proporção de valores na lista x que são menores ou iguais a a .

Observe que ainda não discutimos a probabilidade no contexto das CDFs. Vamos fazer isso perguntando o seguinte: Se eu escolher um dos alunos do sexo masculino aleatoriamente, qual é a probabilidade de ele ser mais alto que 70,5 polegadas? Como cada aluno tem a mesma probabilidade de ser escolhido, a resposta é equivalente à proporção de alunos com mais de 70,5 polegadas. Usando o CDF, obtemos uma resposta escrevendo:

```
1 - F(70)
#> [1] 0.377
```

Uma vez definido um CDF, podemos usá-lo para calcular a probabilidade de qualquer subconjunto. Por exemplo, a probabilidade de um aluno estar entre a altura a e altura b é:

```
F(b)-F(a)
```

Como podemos calcular a probabilidade de qualquer evento possível dessa maneira, a função de probabilidade cumulativa define a distribuição de probabilidade para escolher uma altura aleatória do nosso vetor de altura. x .

13.9 Distribuições teóricas contínuas

Na seção 8.8 apresentamos a distribuição normal como uma aproximação útil para muitas distribuições naturais, incluindo a altura. A distribuição cumulativa para a distribuição normal é definida por uma fórmula matemática que pode ser obtida em R com a função `pnorm`. Dizemos que uma quantidade aleatória é normalmente distribuída com uma média m e desvio padrão s se sua distribuição de probabilidade for definida por:

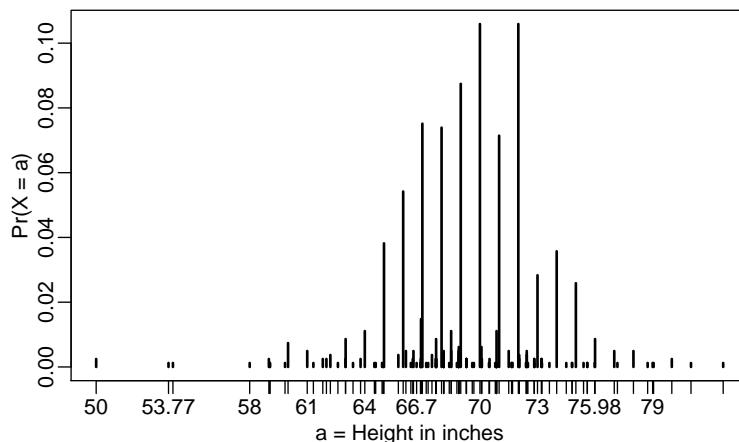
$$F(a) = \text{pnorm}(a, m, s)$$

Isso é útil porque, se estamos dispostos a usar a aproximação normal para, por exemplo, altura, não precisamos de todo o conjunto de dados para responder a perguntas como: qual é a probabilidade de um aluno selecionado aleatoriamente ter mais de 70 anos polegadas? Só precisamos da altura média e do desvio padrão:

```
m <- mean(x)
s <- sd(x)
1 - pnorm(70.5, m, s)
#> [1] 0.371
```

13.9.1 Distribuições teóricas como aproximações

A distribuição normal é derivada matematicamente: não precisamos de dados para defini-la. Para os cientistas de dados, quase tudo o que fazemos na prática envolve dados. Os dados são sempre, do ponto de vista técnico, discretos. Por exemplo, podemos considerar nossos dados de altura categóricos com cada altura específica como uma única categoria. A distribuição de probabilidade é definida pela proporção de alunos que indicam cada altura. Aqui está um gráfico dessa distribuição de probabilidade:



Enquanto a maioria dos estudantes arredondava suas alturas para a polegada mais próxima, outros indicavam valores com mais precisão. Um aluno indicou que sua altura era de 69,6850393700787 polegadas, o que equivale a 177 centímetros. A probabilidade atribuída a esta altura é 0.001 ou 1 em 812. A probabilidade de 70 polegadas é muito maior em 0.106 mas faz sentido pensar que a probabilidade de ter exatamente 70 polegadas é diferente de 69,6850393700787? Claramente, é muito mais útil para fins de análise de dados tratar esse resultado como uma variável numérica contínua, considerando que muito poucas pessoas, ou

talvez não, tenham exatamente 70 polegadas e que a razão pela qual obtemos mais valores 70 é porque as pessoas arredondam para a polegada mais próxima.

Com distribuições contínuas, a probabilidade de um valor singular não é definida. Por exemplo, não faz sentido perguntar qual é a probabilidade de um valor distribuído normalmente ser 70. Em vez disso, definimos probabilidades para intervalos. Portanto, poderíamos perguntar qual é a probabilidade de alguém medir entre 69,5 e 70,5.

Em casos como altura, onde os dados são arredondados, a aproximação normal é particularmente útil se estivermos trabalhando com intervalos que incluem exatamente um número redondo. Por exemplo, a distribuição normal é útil para aproximar a proporção de alunos que relatam valores de intervalo, como os três seguintes:

```
mean(x <= 68.5) - mean(x <= 67.5)
#> [1] 0.115
mean(x <= 69.5) - mean(x <= 68.5)
#> [1] 0.119
mean(x <= 70.5) - mean(x <= 69.5)
#> [1] 0.122
```

Observe como chegamos perto da abordagem normal:

```
pnorm(68.5, m, s) - pnorm(67.5, m, s)
#> [1] 0.103
pnorm(69.5, m, s) - pnorm(68.5, m, s)
#> [1] 0.11
pnorm(70.5, m, s) - pnorm(69.5, m, s)
#> [1] 0.108
```

No entanto, a aproximação não é tão útil para outros intervalos. Por exemplo, observe como a aproximação se divide quando tentamos estimar:

```
mean(x <= 70.9) - mean(x<=70.1)
#> [1] 0.0222
```

com:

```
pnorm(70.9, m, s) - pnorm(70.1, m, s)
#> [1] 0.0836
```

Em geral, chamamos essa situação de “discriminação”. Embora a distribuição real da altura seja contínua, as alturas relatadas tendem a ser mais comuns em valores discretos, neste caso, devido ao arredondamento. Desde que saibamos lidar com essa realidade, a abordagem normal pode ser uma ferramenta muito útil.

13.9.2 A densidade de probabilidade

Para distribuições categóricas, podemos definir a probabilidade de uma categoria. Por exemplo, um dado, vamos chamá-lo X , pode ser 1,2,3,4,5 ou 6. A probabilidade de 4 é definida como:

$$\Pr(X = 4) = 1/6$$

O CDF pode então ser facilmente definido:

$$F(4) = \Pr(X \leq 4) = \Pr(X = 4) + \Pr(X = 3) + \Pr(X = 2) + \Pr(X = 1)$$

Embora para distribuições contínuas a probabilidade de um único valor $\Pr(X = x)$ não definido, existe uma definição teórica que tem uma interpretação semelhante. A densidade de probabilidade em x é definido como a função $f(a)$ tal que:

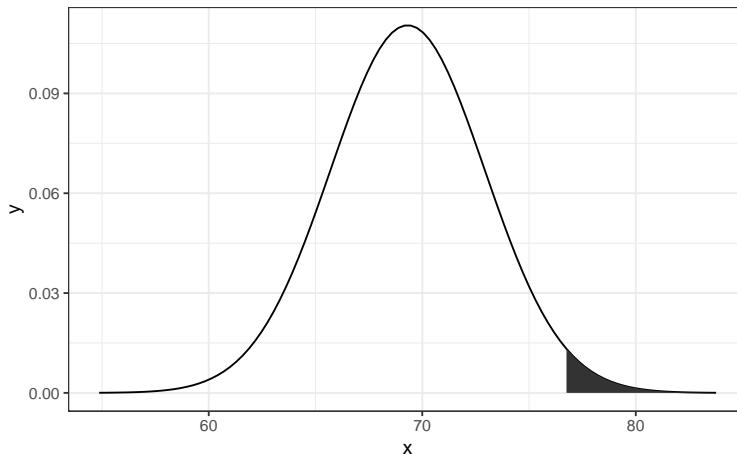
$$F(a) = \Pr(X \leq a) = \int_{-\infty}^a f(x) dx$$

Para quem conhece o cálculo, lembre-se de que a integral está relacionada a uma soma: é a soma das barras com larguras próximas a 0. Se você não conhece o cálculo, pode pensar em $f(x)$ como uma curva para a qual a área sob essa curva até o valor a dá a eles a probabilidade $\Pr(X \leq a)$.

Por exemplo, para usar a aproximação normal para estimar a probabilidade de alguém ter mais de 76 polegadas, usamos:

```
1 - pnorm(76, m, s)
#> [1] 0.0321
```

que matematicamente é a área cinza abaixo:



A curva que você vê é a densidade de probabilidade para a distribuição normal. Em R, obtemos isso usando a função `dnorm`.

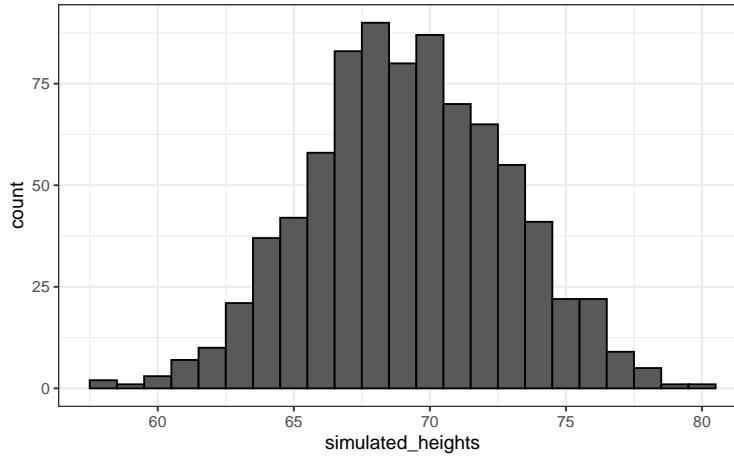
Embora possa não ser imediatamente óbvio por que é útil conhecer as densidades de probabilidade, entender esse conceito será essencial para aqueles que desejam ajustar modelos a dados para os quais não há funções predefinidas disponíveis.

Simulações de Monte Carlo para variáveis contínuas

R fornece funções para gerar resultados normalmente distribuídos. Especificamente, a função `rnorm` são necessários três argumentos: tamanho, média (padrão 0) e desvio padrão (padrão 1) e produz números aleatórios. Aqui está um exemplo de como podemos gerar dados que se parecem com nossas alturas:

```
n <- length(x)
m <- mean(x)
s <- sd(x)
simulated_heights <- rnorm(n, m, s)
```

Não é de surpreender que a distribuição pareça normal:



Essa é uma das funções mais úteis em R, pois permite gerar dados que imitam eventos naturais e responder a perguntas relacionadas ao que poderia acontecer por acaso ao executar simulações de Monte Carlo.

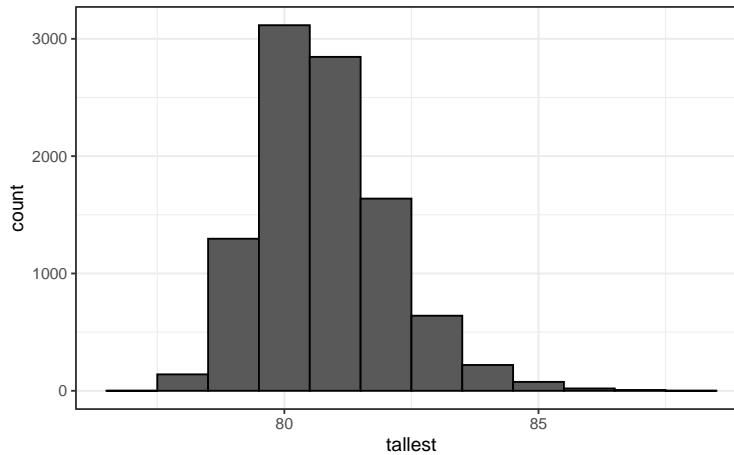
Se, por exemplo, escolhermos 800 homens aleatoriamente, qual é a distribuição da pessoa mais alta? Quão raro é um homem de sete pés e sete pés de rodapé em um grupo de 800 homens? A seguinte simulação de Monte Carlo nos ajuda a responder a essa pergunta:

```
B <- 10000
tallest <- replicate(B, {
  simulated_data <- rnorm(800, m, s)
  max(simulated_data)
})
```

Ter um *pés de rodapé* é bastante raro:

```
mean(tallest >= 7*12)
#> [1] 0.0181
```

Aqui vemos a distribuição resultante:



Observe que isso não parece normal.

13.10 Distribuições contínuas

Apresentamos a distribuição normal na Seção 8.8 e foi usado como um exemplo acima. A distribuição normal não é a única distribuição teórica útil. Outras distribuições contínuas que podemos encontrar são t de Student (*Estudante t* em inglês), qui-quadrado, exponencial, gama, beta e beta-binomial. R fornece funções para calcular densidade, quantis, funções de distribuição cumulativa e gerar simulações de Monte Carlo. R usa uma convenção que nos ajuda a lembrar nomes: use letras **d**, **q**, **p** e **r** na frente de uma abreviação do nome da distribuição. Já vimos as funções **dnorm**, **pnorm** e **rnorm** para a distribuição normal. A função **qnorm** nos dá os quantis. Portanto, podemos traçar uma distribuição como esta:

```
x <- seq(-4, 4, length.out = 100)
qplot(x, f, geom = "line", data = data.frame(x, f = dnorm(x)))
```

Para a distribuição t do aluno, descrita mais adiante na Seção ??, a abreviação **t** é usado para funções serem **dt** para densidade, **qt** para quantis, **pt** para a função de distribuição cumulativa e **rt** para a simulação de Monte Carlo.

13.11 Exercícios

- Suponha que a distribuição das alturas femininas seja aproximada por uma distribuição normal com uma média de 64 polegadas e um desvio padrão de 3 polegadas. Se escolhermos uma mulher aleatoriamente, qual é a probabilidade de ela ter um metro ou menos?
- Suponha que a distribuição das alturas femininas seja aproximada por uma distribuição normal com uma média de 64 polegadas e um desvio padrão de 3 polegadas. Se escolhermos uma mulher aleatoriamente, qual é a probabilidade de ela ter um metro e oitenta ou mais?
- Suponha que a distribuição das alturas femininas seja aproximada por uma distribuição normal com uma média de 64 polegadas e um desvio padrão de 3 polegadas. Se escolhermos uma mulher aleatoriamente, qual é a probabilidade de ela ter entre 61 e 67 polegadas?
- Repita o exercício anterior, mas converta tudo em centímetros. Ou seja, multiplique cada altura, incluindo o desvio padrão, por 2,54. Qual é a resposta agora?
- Observe que a resposta à pergunta não muda quando as unidades mudam. Isso faz sentido, pois a resposta à pergunta não deve ser afetada pelas unidades que usamos. De fato, se você olhar de perto, verá que 61 e 64 estão a 1 SD da média. Encontre a probabilidade de uma variável aleatória aleatória e normalmente distribuída estar dentro de 1 DP da média.
- Para ver a matemática que explica por que as respostas às perguntas 3, 4 e 5 são as mesmas, suponha que você tenha uma variável aleatória média m e erro padrão s . Suponha que você queira saber a probabilidade de que X é menor ou igual a a . Lembre-se de que, por definição a é $(a - m)/s$ desvio padrão s da média m . A probabilidade é:

$$\Pr(X \leq a)$$

Agora subtraia μ em ambos os lados e depois divida os dois lados por σ :

$$\Pr\left(\frac{X - m}{s} \leq \frac{a - m}{s}\right)$$

A quantidade à esquerda é uma variável aleatória normal padrão. Tem uma média de 0 e um erro padrão de 1. Vamos chamá-lo Z :

$$\Pr\left(Z \leq \frac{a - m}{s}\right)$$

Portanto, independentemente das unidades, a probabilidade de $X \leq a$ é igual à probabilidade de uma variável normal padrão ser menor que $(a - m)/s$. Sim `m` é a média e `sigma` o erro padrão, qual dos seguintes códigos R fornecerá a resposta correta em cada situação?

para. `mean(X<=a)` b. `pnorm((a - m)/s)` c. `pnorm((a - m)/s, m, s)` d. `pnorm(a)`

7. Imagine que a distribuição de machos adultos seja aproximadamente normal, com um valor esperado de 69 e um desvio padrão de 3. Qual é a altura do macho no percentil 99? Dica: use `qnorm`.

8. A distribuição das pontuações de QI, ou QI, é aproximadamente distribuída normalmente. A média é 100 e o desvio padrão é 15. Suponha que você queira saber a distribuição dos QIs mais altos em todas as turmas de graduação de cada distrito escolar, cada uma com 10.000 pessoas. Execute uma simulação de Monte Carlo com `B=1000` gerando 10.000 pontos de QI e mantendo os QIs mais altos. Faça um histograma.

14

Variáveis aleatórias

Na ciência de dados, geralmente trabalhamos com dados que são afetados de alguma forma pelo acaso. Alguns exemplos são dados provenientes de uma amostra aleatória, dados afetados por um erro de medição ou dados que medem algum resultado de natureza aleatória. Ser capaz de quantificar a incerteza introduzida pela aleatoriedade é um dos trabalhos mais importantes dos analistas de dados. A inferência estatística oferece uma estrutura para fazer isso, assim como diversas outras ferramentas práticas. O primeiro passo é aprender a como descrever matematicamente variáveis aleatórias.

Neste capítulo, apresentamos variáveis aleatórias (*random variables*) e suas propriedades, começando com sua aplicação em jogos de azar. Em seguida, descrevemos alguns dos eventos relacionados à crise financeira de 2007-2008¹ usando a teoria da probabilidade. Essa crise financeira foi causada em parte pela subestimação do risco de certos títulos vendidos por instituições financeiras. Especificamente, os riscos de títulos hipotecários e as obrigações de dívidas com garantias foram amplamente subestimados. Esses ativos foram vendidos a preços que presumiam que a maioria dos proprietários pagaria pontualmente, e a probabilidade de que isso não ocorresse foi calculada como baixa. Uma combinação de fatores resultou em muito mais inadimplências do que o esperado, levando a uma queda nos preços desses títulos. Como consequência, bancos perderam tanto dinheiro que precisaram de ajuda do governo americano para evitar o fechamento completo.

14.1 Variáveis aleatórias

Variáveis aleatórias são os resultados numéricos de processos aleatórios. Podemos facilmente gerar variáveis aleatórias usando alguns dos simples exemplos que mostramos anteriormente. Por exemplo, vamos definir X igual a 1 se uma bola for azul ou igual a 0 caso seja vermelha:

```
bolas <- rep( c("vermelho", "azul"), times = c(2,3))
X <- ifelse(sample(bolas, 1) == "azul", 1, 0)
```

Aqui X é uma variável aleatória: toda vez que selecionamos uma nova bola, o resultado muda aleatoriamente. Veja abaixo:

```
ifelse(sample(bolas, 1) == "azul", 1, 0)
#> [1] 1
ifelse(sample(bolas, 1) == "azul", 1, 0)
#> [1] 0
ifelse(sample(bolas, 1) == "azul", 1, 0)
#> [1] 0
```

¹https://pt.wikipedia.org/wiki/Crise_financeira_de_2007-2008

Às vezes é 1 e às vezes é 0.

14.2 Modelos de amostragem

Muitos procedimentos de geração de dados, aqueles que produzem os dados que estudamos, podem ser modelados relativamente bem como se estivéssemos sorteando de olhos vendados elementos em uma urna. Por exemplo, podemos modelar os votos de eleitores americanos como 0s (para eleitores do partido republicano) e 1s (para eleitores do partido democrata) em uma urna que contém os códigos 0 e 1 para todos os prováveis eleitores. Em estudos epidemiológicos, geralmente assumimos que os sujeitos de nosso estudo são uma amostra aleatória da população de interesse. Os dados relacionados a um resultado específico podem ser modelados como uma amostra aleatória de uma urna contendo o resultado para toda a população de interesse. Da mesma forma, na pesquisa experimental, geralmente assumimos que os organismos individuais que estamos estudando, como por exemplo, vermes, moscas ou ratos, são uma amostra aleatória de uma população maior. Experimentos aleatórios também podem ser modelados como o sorteio de uma urna, considerando a maneira como os indivíduos são designados para grupos: quando designados, o grupo é escolhido aleatoriamente. Os modelos de amostragem são, portanto, omnipresentes na ciência de dados. Jogos de cassino oferecem uma infinidade de exemplos de situações do mundo real em que modelos de amostragem são usados para responder a perguntas específicas. Portanto, começaremos com esses exemplos.

Suponha que um cassino muito pequeno o contrate para ver se eles devem incluir jogos de roleta. Para simplificar o exemplo, assumiremos que 1.000 pessoas jogarão e que o único tipo de aposta que podem fazer na roleta é apostar em vermelho ou preto. O cassino quer que você preveja quanto dinheiro ganharão ou perderão. Eles querem uma gama de valores e, em particular, querem saber qual é a probabilidade de perderem dinheiro. Se essa probabilidade for muito alta, eles não instalarão jogos de roleta.

Vamos definir uma variável aleatória S que irá representar o total de ganhos do cassino. Vamos começar construindo a urna. Uma roleta tem 18 posições vermelhas, 18 posições pretas e duas verdes. Portanto, jogar uma cor em um jogo de roleta é equivalente a escolher desta urna:

```
color <- rep(c("Preto", "Vermelho", "Verde"), c(18, 18, 2))
```

Os 1.000 resultados de 1.000 pessoas jogando são eventos independentes desta urna. Se o jogador apostar no vermelho e o resultado for vermelho, o jogador ganha e o cassino perde um dólar, então obtemos -\$1. Por outro lado, se o jogador apostar no vermelho e o resultado for preto, o cassino ganha um dólar e obtemos \$1. Para construir nossa variável aleatória S , podemos usar este código:

```
n <- 1000
X <- sample(ifelse(color == "Vermelho", -1, 1), n, replace = TRUE)
X[1:10]
#> [1] -1  1  1 -1 -1  1  1  1  1
```

Como sabemos as proporções de 1s e -1s, podemos simular os resultados da roleta com uma linha de código, sem a necessidade de definir a cor (`color`):

```
X <- sample(c(-1,1), n, replace = TRUE, prob=c(9/19, 10/19))
```

Chamamos isso de **modelo de amostragem**, pois estamos modelando o comportamento aleatório da roleta com a amostragem das opções disponíveis. O ganho total de S é simplesmente a soma desses 1.000 eventos independentes:

```
X <- sample(c(-1,1), n, replace = TRUE, prob=c(9/19, 10/19))
S <- sum(X)
S
#> [1] 22
```

14.3 A distribuição de probabilidade de uma variável aleatória

Se você executar o código acima, verá que S muda sempre. Isto é porque S é uma **variável aleatória**. A distribuição de probabilidade de uma variável aleatória nos diz a probabilidade de que o valor observado caia em um determinado intervalo. Por exemplo, se queremos saber a probabilidade de perdermos dinheiro, estamos perguntando a probabilidade de que S estar no intervalo $S < 0$.

Observe que se pudermos definir uma função de distribuição cumulativa $F(a) = \Pr(S \leq a)$, então podemos responder a qualquer pergunta relacionada à probabilidade de eventos definidos por nossa variável aleatória S , incluindo o evento $S < 0$. Chamamos isso de *função de distribuição* da variável aleatória.

Podemos estimar a função de distribuição de uma variável aleatória S usando a simulação de Monte Carlo para gerar multiplas repetições de uma variável randômica. No código a seguir, realizamos o experimento de ter 1.000 pessoas jogando roleta B vezes, especificamente $B = 10.000$ vezes neste exemplo:

```
n <- 1000
B <- 10000
roulette_winnings <- function(n){
  X <- sample(c(-1,1), n, replace = TRUE, prob=c(9/19, 10/19))
  sum(X)
}
S <- replicate(B, roulette_winnings(n))
```

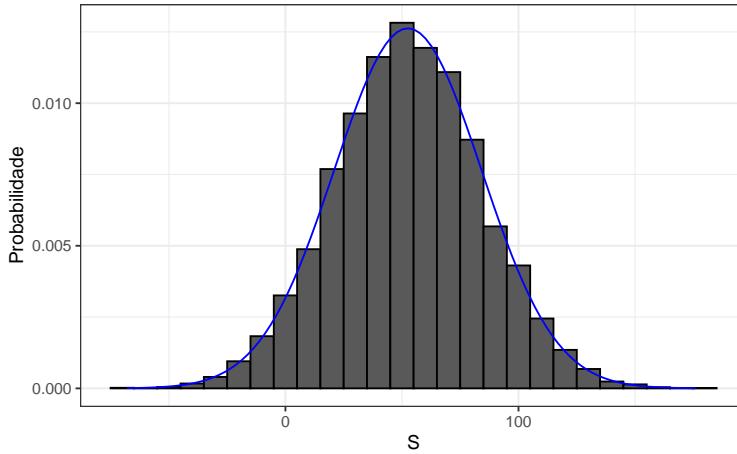
Agora podemos perguntar o seguinte: em nossas simulações, com que frequência recebemos somas inferiores ou iguais a a ?

```
mean(S <= a)
```

Essa será uma aproximação muito boa de $F(a)$. Podemos facilmente responder à pergunta do cassino: qual a probabilidade de perdermos dinheiro? Podemos ver que a probabilidade é bastante baixa:

```
mean(S<0)
#> [1] 0.0456
```

Podemos visualizar a distribuição de S criando um histograma mostrando a probabilidade $F(b) - F(a)$ para vários intervalos $(a, b]$:



Vemos que a distribuição parece ser aproximadamente normal. Um gráfico QQ confirmará que a aproximação normal está próxima de uma aproximação perfeita para essa distribuição. De fato, se a distribuição é normal, tudo o que precisamos para definir a distribuição é a média e o desvio padrão. Como temos os valores originais dos quais a distribuição é criada, podemos facilmente calculá-los usando `mean(S)` e `sd(S)`. A curva azul adicionada ao histograma acima representa uma densidade normal com essa média e desvio padrão.

Nesse caso, a média e o desvio padrão têm nomes especiais. Eles são conhecidos como valor esperado (*expected value*) e erro padrão (*standard error*) da variável aleatória S . Discutiremos mais sobre isso na próxima seção.

A teoria estatística fornece uma maneira de derivar a distribuição de variáveis aleatórias definidas como extrações aleatórias independentes de uma urna. Especificamente, em nosso exemplo anterior, podemos mostrar que $(S + n)/2$ segue uma distribuição binomial. Portanto, não precisamos executar simulações de Monte Carlo para conhecer a distribuição de probabilidade de S . Fizemos isso para fins ilustrativos.

Podemos usar as funções `dbinom` e `pbinom` para calcular as probabilidades exatamente. Por exemplo, para calcular $\Pr(S < 0)$ nós notamos que:

$$\Pr(S < 0) = \Pr((S + n)/2 < (0 + n)/2)$$

e podemos usar `pbinom` calcular:

$$\Pr(S \leq 0)$$

```
n <- 1000
pbinom(n/2, size = n, prob = 10/19)
#> [1] 0.0511
```

Por se tratar de uma função de probabilidade discreta, para obter $\Pr(S < 0)$ ao invés de $\Pr(S \leq 0)$, nós escrevemos:

```
pbinom(n/2-1, size = n, prob = 10/19)
#> [1] 0.0448
```

Para mais detalhes sobre a distribuição binomial, você pode consultar qualquer livro de probabilidades básico ou mesmo a Wikipedia².

Aqui, nós não iremos cobrir esses detalhes. Em vez disso, discutiremos uma abordagem incrivelmente útil fornecida pela teoria matemática que se aplica geralmente a somas e calcula as médias de qualquer caixa de votação: o Teorema do Limite Central.

14.4 Distribuições versus distribuições de probabilidade

Antes de continuar, vamos fazer uma distinção e uma conexão importante entre a distribuição de uma lista de números e uma distribuição de probabilidade. No capítulo de visualização, descrevemos como qualquer lista de números x_1, \dots, x_n tem uma distribuição. A definição é bastante direta. Definimos $F(a)$ como a função que nos diz que proporção da lista é menor ou igual a a . Como são resumos úteis quando a distribuição é aproximadamente normal, definimos a média e o desvio padrão. Eles são definidos com uma operação simples do vetor que contém a lista de números x :

```
m <- sum(x)/length(x)
s <- sqrt(sum((x - m)^2) / length(x))
```

Uma variável aleatória X tem uma função de distribuição. Para definir isso, não precisamos de uma lista de números. É um conceito teórico. Nesse caso, definimos a distribuição como o $F(a)$ respondendo à pergunta: qual é a probabilidade de X ser menor ou igual a a ? Não há lista de números.

No entanto, se X é definido como uma seleção de uma urna com números, então há uma lista: a lista de números dentro da urna. Nesse caso, a distribuição dessa lista é a distribuição de probabilidade de X e a média e o desvio padrão dessa lista são o valor esperado e o erro padrão da variável aleatória.

Outra maneira de pensar sobre isso, que não envolva uma urna, é executar uma simulação de Monte Carlo e gerar uma lista muito grande de resultados de X . Esses resultados são uma lista de números. A distribuição dessa lista será uma aproximação muito boa da distribuição de probabilidade de X . Quanto maior a lista, melhor a aproximação. A média e o desvio padrão dessa lista irão se aproximar do valor esperado e do erro padrão da variável aleatória.

14.5 Notação para variáveis aleatórias

Nos livros estatísticos, as letras maiúsculas são usadas para denotar variáveis aleatórias e seguimos essa convenção aqui. Letras minúsculas são usadas para valores observados. Você verá algumas notações que incluem ambos. Por exemplo, verá eventos definidos como $X \leq x$. Aqui, X é uma variável aleatória, fazendo disso um evento aleatório, e x é um valor arbitrário e não aleatório. Assim, X poderia, por exemplo, representar o número em um dado e x representaria um valor real que vemos: 1, 2, 3, 4, 5 ou 6. Portanto, neste caso, a probabilidade de $X = x$ é $1/6$ independentemente do valor observado x . Essa notação é um pouco estranha

²https://pt.wikipedia.org/wiki/Distribuição_binomial

porque, quando fazemos perguntas de probabilidade, X não é uma quantidade observada, mas uma quantidade aleatória que veremos no futuro. Podemos falar sobre o que esperamos que seja, quais valores são prováveis, mas não quais são exatamente. Mas, uma vez que temos os dados, vemos uma realização de X . Portanto, cientistas de dados falam sobre o que poderia ter sido depois de ver o que realmente aconteceu.

14.6 O valor esperado e o erro padrão

Descrevemos anteriormente modelos de amostragem para eventos. Agora, revisaremos a teoria matemática que nos permite aproximar as distribuições de probabilidade para a soma dos eventos. Depois de fazer isso estaremos aptos para ajudar o cassino a prever quanto dinheiro ganharão. A mesma abordagem que usamos para a soma dos eventos será útil para descrever a distribuição das médias e a proporção que precisaremos para entender como as pesquisas funcionam.

O primeiro conceito importante a aprender é o valor esperado. Nos livros de estatística, é comum usar a letra E desta forma:

$$\mathbb{E}[X]$$

para denotar o valor esperado da variável aleatória X .

Uma variável aleatória irá variar em torno do valor esperado de uma maneira que, se você pegar a média de muitos, muitos eventos, a média dos eventos se aproximará do valor esperado, aproximando-se cada vez mais à medida que os eventos aumentam.

A estatística teórica fornece técnicas que facilitam o cálculo dos valores esperados em diferentes circunstâncias. Por exemplo, uma fórmula útil nos diz que *o valor esperado de uma variável aleatória definida por um sorteio é a média dos números na urna*. Na urna que usamos para modelar apostas de roleta no vermelho, obtivemos 20 resultados de \$1 e 18 resultados de -\$1. O valor esperado é então:

$$\mathbb{E}[X] = (20 + -18)/38$$

que é cerca de 5 centavos. É um pouco contra-intuitivo dizer que X varia em torno de 0,05, quando os únicos valores utilizados são 1 e -1. Uma maneira de entender o valor esperado nesse contexto é perceber que, se jogarmos o jogo repetidamente, o cassino ganha, em média, 5 centavos por jogo. Uma simulação de Monte Carlo confirma isso:

```
B <- 10^6
x <- sample(c(-1,1), B, replace = TRUE, prob=c(9/19, 10/19))
mean(x)
#> [1] 0.0517
```

Em geral, se a urna tem dois resultados possíveis, digamos a e b com proporções p e $1 - p$, respectivamente, a média é:

$$\mathbb{E}[X] = ap + b(1 - p)$$

Para ver isso, observe que se houver n bolas na urna, então temos np as e $n(1 - p)$ bs, e como a média é a soma, $n \times a \times p + n \times b \times (1 - p)$, dividido pelo total n , obtemos que a média é $ap + b(1 - p)$.

Agora, a razão pela qual definimos o valor esperado é porque essa definição matemática é útil para aproximar as distribuições de probabilidade da soma, que é útil para descrever a distribuição de médias e proporções. O primeiro fato útil é que *o valor esperado da soma dos eventos* é:

$$\text{número de eventos} \times \text{média dos números na urna}$$

Portanto, se 1.000 pessoas jogam roleta, o cassino espera ganhar, em média, cerca de $1.000 \times \$0,05 = \50 . Mas esse é um valor esperado. Quão diferente pode ser uma observação do valor esperado? O cassino realmente precisa saber disso. Qual é o intervalo de probabilidades? Se números negativos forem muito prováveis, eles não instalarão roletas. A teoria estatística mais uma vez responde a essa pergunta. O *erro padrão* (do inglês *standard error* ou SE) nos dá uma ideia do tamanho da variação em torno do valor esperado. Nos livros de estatística, é comum usar:

$$\text{SE}[X]$$

para denotar o erro padrão de uma variável aleatória.

Se nossos eventos são independentes, o erro padrão da soma é dado pela equação:

$$\sqrt{\text{número de eventos}} \times \text{desvio padrão dos números na urna}$$

Usando a definição de desvio padrão, podemos derivar, com um pouco de matemática, que se uma urna contiver dois valores a e b com proporções p e $(1 - p)$, respectivamente, o desvio padrão é:

$$|b - a| \sqrt{p(1 - p)}.$$

Portanto, no nossa exemplo da roleta, o desvio padrão dos valores dentro da urna é: $|1 - (-1)| \sqrt{10/19 \times 9/19}$ ou:

```
2 * sqrt(90)/19
#> [1] 0.999
```

O erro padrão nos diz a diferença típica entre uma variável aleatória e sua expectativa. Como um sorteio é obviamente a soma de um único sorteio, podemos usar a fórmula acima para calcular que a variável aleatória definida por um sorteio tem um valor esperado de 0,05 e um erro padrão de aproximadamente 1. Isso faz sentido, já que obtemos 1 ou -1, com 1 levemente favorecido sobre -1.

Usando a fórmula acima, a soma de 1.000 pessoas jogando tem um erro padrão de aproximadamente \$32:

```
n <- 1000
sqrt(n) * 2 * sqrt(90)/19
#> [1] 31.6
```

Como resultado, quando 1.000 pessoas apostarem no vermelho, o cassino deverá ganhar \$50 com um erro padrão de \$32. Portanto, isso aparenta ser uma aposta segura. Mas ainda não respondemos à pergunta: qual a probabilidade de perder dinheiro? Aqui o teorema do limite central nos ajudará.

Nota Avançada: Antes de continuar, devemos apontar que os cálculos exatos de probabilidade de vitória no cassino podem ser feitos com a distribuição binomial. Entretanto, aqui nos concentramos no teorema do limite central, que geralmente pode ser aplicado a somas de variáveis aleatórias, algo que não pode ser feito com a distribuição binomial.

14.6.1 Desvio padrão da população versus desvio padrão da amostra

O desvio padrão (do inglês *standard deviation* ou SD) de uma lista x (usamos as alturas abaixo como exemplo) é definido como a raiz quadrada da média das diferenças quadradas:

```
library(dslabs)
x <- heights$height
m <- mean(x)
s <- sqrt(mean((x-m)^2))
```

Usando notação matemática, escrevemos:

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

No entanto, observe que a função `sd` retorna um resultado ligeiramente diferente:

```
identical(s, sd(x))
#> [1] FALSE
s-sd(x)
#> [1] -0.00194
```

Isso ocorre porque a função `sd` em R não retorna o desvio padrão da lista, mas sim usa uma fórmula que estima desvios padrão da população a partir de uma amostra aleatória X_1, \dots, X_N que, por razões não discutidas aqui, divide a soma dos quadrados por $N - 1$.

$$\bar{X} = \frac{1}{N} \sum_{i=1}^N X_i, \quad s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (X_i - \bar{X})^2}$$

Você pode ver que esse é o caso digitando:

```
n <- length(x)
s-sd(x)*sqrt((n-1)/ n)
#> [1] 0
```

Para toda a teoria discutida aqui, você precisa calcular o desvio padrão real, conforme definido:

```
sqrt(mean((x-m)^2))
```

Portanto, tenha cuidado ao usar a função `sd` no R. No entanto, lembre-se de que, ao longo do livro, às vezes usamos a função `sd` quando realmente queremos o verdadeiro desvio

padrão. Isso ocorre porque quando o tamanho da lista é grande, os valores obtidos pelas duas fórmulas são praticamente equivalentes, pois $\sqrt{(N-1)/N} \approx 1$.

14.7 Teorema do Limite Central

O Teorema do Limite Central (TLC) nos diz que quando o número de eventos, também chamado *tamanho da amostra*, é grande, a distribuição de probabilidade da soma de eventos independentes é aproximadamente normal. Como modelos de amostragem são usados para muitos processos de geração de dados, o TLC é considerado uma das ideias matemáticas mais importantes da história.

Anteriormente, discutimos que, se soubermos que a distribuição de uma lista de números se aproxima da distribuição normal, tudo o que precisamos para descrever a lista é a média e o desvio padrão. Também sabemos que o mesmo se aplica às distribuições de probabilidade. Se uma variável aleatória tem uma distribuição de probabilidade que se aproxima da distribuição normal, tudo o que precisamos para descrever a distribuição de probabilidade é a média e o desvio padrão, referidos como valor esperado e erro padrão.

Anteriormente, executamos esta simulação de Monte Carlo:

```
n <- 1000
B <- 10000
roulette_winnings <- function(n){
  X <- sample(c(-1,1), n, replace = TRUE, prob=c(9/19, 10/19))
  sum(X)
}
S <- replicate(B, roulette_winnings(n))
```

O Teorema do Limite Central nos diz que a soma S é aproximada por uma distribuição normal. Usando as fórmulas acima, sabemos que o valor esperado e o erro padrão são:

```
n * (20-18)/38
#> [1] 52.6
sqrt(n) * 2 * sqrt(90)/19
#> [1] 31.6
```

Os valores teóricos acima correspondem aos obtidos com a simulação de Monte Carlo:

```
mean(S)
#> [1] 52.2
sd(S)
#> [1] 31.7
```

Usando o TLC, podemos pular a simulação de Monte Carlo e, em vez disso, calcular a probabilidade de o cassino perder dinheiro usando esta aproximação:

```
mu <- n * (20-18)/38
se <- sqrt(n) * 2 * sqrt(90)/19
pnorm(0, mu, se)
#> [1] 0.0478
```

o que também concorda muito bem com o resultado da simulação de Monte Carlo:

```
mean(S < 0)
#> [1] 0.0458
```

Qual é o tamanho do Teorema do Limite Central?

O Teorema do Limite Central funciona quando o número de eventos é amplo. Mas amplo é um termo relativo. Em muitas circunstâncias, apenas 30 eventos são suficientes para que o TLC seja útil. Em alguns casos específicos, apenas 10 eventos são suficientes. No entanto, essas não devem ser consideradas regras gerais. Por exemplo, quando a probabilidade de sucesso é muito pequena, precisamos de tamanhos de amostra muito maiores.

A título de ilustração, vamos considerar a loteria. Na loteria, a chance de ganhar é menor que 1 em um milhão. Milhares de pessoas jogam, então o número de apostas é muito grande. Já o número de vencedores, a soma dos sorteados, varia entre 0 a 4. A distribuição normal não é uma boa aproximação da soma, portanto o Teorema do Limite Central não se aplica, mesmo quando o tamanho da amostra é muito grande. Isso geralmente é verdade quando a probabilidade de sucesso é muito baixa. Nesses casos, a distribuição de Poisson é mais apropriada.

Você pode examinar as propriedades da distribuição de Poisson usando `dpois` ou `ppois`. Você pode gerar variáveis aleatórias seguindo esta distribuição com `rpois`. No entanto, não discutimos a teoria aqui. Para saber mais sobre a distribuição de Poisson, você pode consultar qualquer livro de probabilidade ou até mesmo a Wikipedia³.

14.8 Propriedades estatísticas das médias

Existem diversos resultados matemáticos úteis que usamos frequentemente para trabalhar com dados. Listamos alguns abaixo.

1. O valor esperado da soma das variáveis aleatórias é a soma do valor esperado de cada variável aleatória. Podemos escrever assim:

$$\mathbb{E}[X_1 + X_2 + \dots + X_n] = \mathbb{E}[X_1] + \mathbb{E}[X_2] + \dots + \mathbb{E}[X_n]$$

Se X são eventos independentes das urnas, então todos têm o mesmo valor esperado. Vamos chamar isso de μ . Logo:

$$\mathbb{E}[X_1 + X_2 + \dots + X_n] = n\mu$$

que é outra maneira de escrever o resultado mostrado acima para a soma dos eventos.

2. O valor esperado de uma constante não aleatória multiplicada por uma variável aleatória é a constante não aleatória multiplicada pelo valor esperado de uma variável aleatória. É mais fácil explicar com símbolos:

$$\mathbb{E}[aX] = a \times \mathbb{E}[X]$$

³https://en.wikipedia.org/w/index.php?title=Poisson_distribution

Para ver por que isso é intuitivo, considere alterar as unidades. Se mudarmos as unidades de uma variável aleatória, digamos de dólares para centavos, a expectativa deve mudar da mesma maneira. Uma consequência dos dois fatos anteriores é que o valor esperado da média de eventos independentes da mesma urna é o valor esperado da urna. Chame isso de μ novamente:

$$\text{E}[(X_1 + X_2 + \dots + X_n)/n] = \text{E}[X_1 + X_2 + \dots + X_n]/n = n\mu/n = \mu$$

3. O quadrado do erro padrão da soma das variáveis aleatórias **independentes** é a soma do quadrado do erro padrão de cada variável aleatória. É mais fácil entender matematicamente:

$$\text{SE}[X_1 + X_2 + \dots + X_n] = \sqrt{\text{SE}[X_1]^2 + \text{SE}[X_2]^2 + \dots + \text{SE}[X_n]^2}$$

O quadrado do erro padrão é chamado de variância (*variance*) nos livros didáticos de estatística. Observe que essa propriedade em particular não é tão intuitiva quanto as anteriores. Explicações mais detalhadas podem ser encontradas nos livros de estatística.

4. O erro padrão de uma constante não aleatória multiplicada por uma variável aleatória é a constante não aleatória multiplicada pelo erro padrão da variável aleatória. O mesmo que para o valor esperado:

$$\text{SE}[aX] = a \times \text{SE}[X]$$

Para ver por que isso é intuitivo, pense novamente nas unidades.

Uma consequência de 3 e 4 é que o erro padrão da média de eventos independentes para a mesma urna é o desvio padrão da urna dividido pela raiz quadrada de n (o número de eventos), chame isso de σ :

$$\begin{aligned}\text{SE}[(X_1 + X_2 + \dots + X_n)/n] &= \text{SE}[X_1 + X_2 + \dots + X_n]/n \\ &= \sqrt{\text{SE}[X_1]^2 + \text{SE}[X_2]^2 + \dots + \text{SE}[X_n]^2}/n \\ &= \sqrt{\sigma^2 + \sigma^2 + \dots + \sigma^2}/n \\ &= \sqrt{n\sigma^2}/n \\ &= \sigma/\sqrt{n}\end{aligned}$$

5. Se X é uma variável aleatória distribuída normalmente e se a e b são constantes não aleatórias, então $aX + b$ também é uma variável aleatória distribuída normalmente. Tudo o que estamos fazendo é alterar as unidades da variável aleatória multiplicando por a e, em seguida, movendo o centro b .

Note que os livros didáticos estatísticos usam as letras gregas μ e σ para indicar o valor esperado e o erro padrão, respectivamente. Isto ocorre porque μ é a letra grega para m , a primeira letra de *média*, que é outro termo usado para o valor esperado. Da mesma forma, σ é a letra grega para s , a primeira letra de *standard error* (erro padrão).

14.9 Lei dos grandes números

Uma implicação importante do resultado final é que o erro padrão da média se torna cada vez menor conforme n torna-se maior. Quando n é muito grande, então o erro padrão é praticamente 0 e a média dos eventos converge para a média da urna. Isso é conhecido nos livros estatísticos como a lei dos grandes números ou a lei das médias.

14.9.1 Interpretação incorreta da lei das médias

A lei das médias, às vezes, é mal interpretada. Por exemplo, se você jogar uma moeda cinco vezes e obter cara em todas, alguém pode argumentar que o próximo sorteio é provavelmente dará coroa devido à lei das médias: em média, devemos ver 50% cara e 50% coroa. Um argumento semelhante seria dizer que o resultado da roleta deve ser vermelho depois de ver que o preto apareceu cinco vezes seguidas. Esses eventos são independentes, portanto a probabilidade de se obter cara após girar uma moeda é de 50%, independentemente dos cinco resultados anteriores. Esse também é o caso do resultado da roleta. A lei das médias se aplica somente quando o número de eventos é muito grande e não em amostras pequenas. Após um milhão de jogadas, você definitivamente verá que cerca de 50% dos resultados serão cara, independentemente do resultado dos cinco primeiros lançamentos.

Outro interessante uso incorreto da lei das médias está nos esportes, quando, por exemplo, apresentadores de televisão prevêem que um jogador está prestes a ter sucesso porque falhou várias vezes seguidas.

14.10 Exercícios

- Nos jogos de roleta americanos, você também pode apostar no verde. Existem 18 vermelhos, 18 pretos e dois verdes (0 e 00). Quais são as chances de um verde sair?
- O pagamento caso você acerte no verde é de \$17 dólares. Isto significa que se você apostar um dólar e o resultado cair no verde, você ganha \$17. Crie um modelo de amostragem usando uma amostra para simular a variável aleatória X para seus ganhos. Dica: Veja o exemplo abaixo de como seria esse código para apostas no vermelho.

```
x <- sample(c(1,-1), 1, prob = c(9/19, 10/19))
```

- Qual é o valor esperado de X ?
- Qual é o erro padrão de X ?
- Agora crie uma variável aleatória S que deve receber a soma dos seus ganhos depois de apostar no verde 1.000 vezes. Dica: altere os argumentos `size` e `replace` na sua resposta da pergunta 2. Inicie seu código configurando a semente aleatória como 1 usando `set.seed(1)`.
- Qual é o valor esperado de S ?
- Qual é o erro padrão de S ?
- Qual é a probabilidade de você acabar ganhando dinheiro? Dica: use o TLC.
- Crie uma simulação de Monte Carlo que gere 1.000 resultados a partir de S . Calcule a

média e o desvio padrão da lista resultante para confirmar os resultados das questões 6 e 7. Inicie seu código configurando a semente aleatória como 1 usando `set.seed(1)`.

10. Agora verifique sua resposta da pergunta 8 usando o resultado da simulação de Monte Carlo.

11. O resultado da simulação de Monte Carlo e a aproximação TLC estão próximos, mas não tão próximos. O que poderia explicar isso?

- a. 1.000 simulações não são suficientes. Se fizermos mais, eles irão coincidir.
- b. O TLC não funciona tão bem quando a probabilidade de sucesso é pequena. Nesse caso, era 1/19. Se aumentarmos o número de jogos de roleta, eles coincidirão melhor.
- c. A diferença está dentro do erro de arredondamento.
- d. O TLC funciona apenas para médias.

12. Agora crie uma variável aleatória Y que deve armazenar sua média de ganhos por aposta depois de apostar 1.000 vezes no verde.

13. Qual é o valor esperado de Y ?

14. Qual é o erro padrão de Y ?

15. Qual é a probabilidade de que, quando você terminar de jogar, os ganhos por jogo sejam positivos? Dica: use o TLC.

16. Crie uma simulação de Monte Carlo que gere 2.500 resultados a partir de Y . Calcule a média e o desvio padrão da lista resultante para confirmar os resultados das questões 6 e 7. Inicie seu código configurando a semente aleatória como 1 usando `set.seed(1)`.

17. Agora verifique sua resposta da pergunta 8 usando o resultado da simulação de Monte Carlo.

18. O resultado da simulação de Monte Carlo e a aproximação TLC estão agora muito mais próximos. O que poderia explicar isso?

- a. Agora estamos calculando médias em vez de somas.
- b. 2.500 simulações de Monte Carlo não são melhores que 1.000.
- c. O TLC funciona melhor quando o tamanho da amostra é maior. Aumentamos de 1.000 para 2.500.
- d. Não está mais perto. A diferença está dentro do erro de arredondamento.

14.11 Estudo de caso: *The Big Short*

14.11.1 Taxas de juros explicadas com modelo de probabilidade

Versões mais complexas dos modelos de amostragem que discutimos anteriormente também são utilizadas por bancos para determinar suas taxas de juros. Suponha que você administre um pequeno banco que tenha um histórico de identificação de possíveis proprietários em que se pode confiar para fazer pagamentos. De fato, historicamente, em um determinado ano, apenas 2% de seus clientes não pagam o dinheiro que lhes foi emprestado. No entanto, o banco sabe que se você simplesmente emprestar dinheiro a todos os seus clientes sem

juros, você acabará perdendo dinheiro por causa desses 2%. Embora o banco saiba que provavelmente 2% de seus clientes não pagarão, eles não sabem quais serão esses clientes. Contudo, cobrando de todos um pouco mais de juros, eles podem compensar as perdas incorridas devido a esses 2% e também cobrir seus custos operacionais. Eles também podem obter lucro. Entretanto, se definirem taxas de juros muito altas, os clientes irão para um outro banco. Usaremos todos esses fatos e um pouco da teoria das probabilidades para determinar qual taxa de juros cobrar.

Suponha que seu banco faça 1.000 empréstimos de \$180.000 este ano. Além disso, depois de somar todos os custos, suponha que seu banco perca \$200.000 por cada execução hipotecária (quando o proprietário não paga as prestações e o banco retoma o imóvel). Para simplificar, assumimos que isso inclui todos os custos operacionais. Um modelo de amostragem para esse cenário pode ser codificado da seguinte maneira:

```
n <- 1000
loss_per_foreclosure <- -200000
p <- 0.02
defaults <- sample( c(0,1), n, prob=c(1-p, p), replace = TRUE)
sum(defaults * loss_per_foreclosure)
#> [1] -3400000
```

Observe que a perda total definida pela soma final é uma variável aleatória. Sempre que executar o código acima, você receberá uma resposta diferente. Podemos facilmente construir uma simulação de Monte Carlo para ter uma ideia da distribuição dessa variável aleatória.

```
B <- 10000
losses <- replicate(B, {
  defaults <- sample( c(0,1), n, prob=c(1-p, p), replace = TRUE)
  sum(defaults * loss_per_foreclosure)
})
```

Porém, realmente não precisamos de uma simulação de Monte Carlo. Usando o que aprendemos, o TLC nos diz que, como nossas perdas são uma soma de eventos independentes, sua distribuição é aproximadamente normal com o valor esperado e o erro padrão dados por:

```
n*(p*loss_per_foreclosure + (1-p)*0)
#> [1] -4e+06
sqrt(n)*abs(loss_per_foreclosure)*sqrt(p*(1-p))
#> [1] 885438
```

Agora podemos definir uma taxa de juros para garantir que, em média, alcancemos um ponto de equilíbrio. Basicamente, precisamos adicionar uma quantidade x para cada empréstimo, que neste caso é representado pelos eventos, para que o valor esperado seja 0. Se definirmos l para ser a perda por execução de hipoteca, precisamos:

$$lp + x(1 - p) = 0$$

implicando que x é:

```
- loss_per_foreclosure*p/(1-p)
#> [1] 4082
```

ou uma taxa de juros de 0.023.

No entanto, ainda temos um problema. Embora essa taxa de juros garanta que, em média,

eles atinjam um ponto de equilíbrio, há 50% de chance de perder dinheiro. Se o seu banco perder dinheiro, eles terão que fechá-lo. Portanto, devemos escolher uma taxa de juros que nos proteja disso. Ao mesmo tempo, se a taxa de juros for muito alta, seus clientes irão para outro banco; portanto, devemos estar dispostos a correr alguns riscos. Então, digamos que você queira que suas chances de perder dinheiro sejam de 1 em 100. O que a quantidade de x deve ser agora? Isso é um pouco mais difícil. Nós queremos que a soma S tenha:

$$\Pr(S < 0) = 0.01$$

Sabemos que S é aproximadamente normal. O valor esperado de S é:

$$\mathbb{E}[S] = \{lp + x(1 - p)\}n$$

com n o número de seleções, que neste caso representa empréstimos. O erro padrão é:

$$\text{SD}[S] = |x - l| \sqrt{np(1 - p)}.$$

Porque x é positivo e l negativo $|x - l| = x - l$. Observe que essas são apenas aplicações das fórmulas mostradas acima, mas usam símbolos mais compactos.

Agora vamos usar um “truque” matemático que é muito comum em estatística. Adicionamos e subtraímos os mesmos valores aos dois lados do evento $S < 0$. Isso é feito para que a probabilidade não mude e acabemos com uma variável aleatória normal padrão à esquerda, o que nos permitirá escrever uma equação com apenas x como desconhecido. Esse “truque” é o seguinte:

Se $\Pr(S < 0) = 0.01$, então:

$$\Pr\left(\frac{S - \mathbb{E}[S]}{\text{SE}[S]} < \frac{-\mathbb{E}[S]}{\text{SE}[S]}\right)$$

E lembre-se disso $\mathbb{E}[S]$ e $\text{SE}[S]$ são o valor esperado e o erro padrão de S , respectivamente. Tudo o que fizemos acima foi somar e dividir pela mesma quantia em ambos os lados. Fizemos isso porque agora o termo à esquerda é uma variável aleatória normal padrão, à qual iremos renomear como Z . Agora preenchemos os espaços em branco com a fórmula atual para o valor esperado e o erro padrão:

$$\Pr\left(Z < \frac{-\{lp + x(1 - p)\}n}{(x - l)\sqrt{np(1 - p)}}\right) = 0.01$$

Pode parecer complicado, mas lembre-se de que l , p e n todas são quantidades conhecidas, então em algum momento as substituiremos por números.

Agora como Z é uma variável aleatória normal com valor esperado 0 e erro padrão 1, significa que a quantidade no lado direito do sinal $<$ deve ser igual a:

```
qnorm(0.01)
#> [1] -2.33
```

para que a equação seja verdadeira. Lembre-se que $z = \text{qnorm}(0.01)$ nos dá o valor de z para qual:

$$\Pr(Z \leq z) = 0.01$$

Isso significa que o lado direito da complicada equação deve ser `z=qnorm(0.01)`:

$$\frac{-\{lp + x(1-p)\}n}{(x - l)\sqrt{np(1-p)}} = z$$

O truque funciona porque acabamos com uma expressão que contém x , que sabemos que deve ser igual a uma quantidade conhecida z . Resolver x agora é simplesmente álgebra:

$$x = -l \frac{np - z\sqrt{np(1-p)}}{n(1-p) + z\sqrt{np(1-p)}}$$

que é:

```
l <- loss_per_foreclosure
z <- qnorm(0.01)
x <- -l*( n*p - z*sqrt(n*p*(1-p)) ) / ( n*(1-p) + z*sqrt(n*p*(1-p)) )
x
#> [1] 6249
```

Nossa taxa de juros agora sobe para 0.035. Ainda é uma taxa de juros muito competitiva. Ao escolher essa taxa de juros, agora teremos um retorno esperado do empréstimo de:

```
loss_per_foreclosure*p + x*(1-p)
#> [1] 2124
```

que é um lucro total esperado de cerca de:

```
n*(loss_per_foreclosure*p + x*(1-p))
#> [1] 2124198
```

dólares!

Podemos executar uma simulação de Monte Carlo para verificar nossas aproximações teóricas:

```
B <- 100000
profit <- replicate(B, {
  draws <- sample( c(x, loss_per_foreclosure), n,
    prob=c(1-p, p), replace = TRUE)
  sum(draws)
})
mean(profit)
#> [1] 2126976
mean(profit<0)
#> [1] 0.0124
```

14.11.2 The Big Short

Um de seus funcionários ressalta que, como o banco está ganhando 2.124 dólares por empréstimo, o banco deveria fazer mais empréstimos! Por que apenas n ? Você explica que

encontrar aqueles n clientes foi difícil. Você precisa de um grupo que seja previsível e que mantenha as chances de inadimplência baixas. Seu funcionário então ressalta que, mesmo que a probabilidade de inadimplência seja maior, desde que o valor esperado seja positivo, o banco pode minimizar sua probabilidade de perda aumentando n e confiando na lei dos grandes números.

Seu funcionário afirma ainda que, mesmo que a taxa de inadimplência seja duas vezes mais alta, digamos 4%, se definirmos a taxa um pouco mais alta do que esse valor:

```
p <- 0.04
r <- (- loss_per_foreclosure*p/(1-p))/ 180000
r
#> [1] 0.0463
```

o banco irá lucrar. Em 5%, temos a garantia de um valor positivo esperado de:

```
r <- 0.05
x <- r*180000
loss_per_foreclosure*p + x * (1-p)
#> [1] 640
```

e podemos minimizar as chances de perder dinheiro simplesmente aumentando n já que:

$$\Pr(S < 0) = \Pr\left(Z < -\frac{\text{E}[S]}{\text{SE}[S]}\right)$$

com Z uma variável aleatória normal padrão, como mostrado acima. Se definirmos μ e σ como o valor esperado e o desvio padrão da urna, respectivamente (ou seja, de um único empréstimo), usando as fórmulas anteriores, temos: $\text{E}[S] = n\mu$ e $\text{SE}[S] = \sqrt{n}\sigma$. Então, se definirmos $z=\text{qnorm}(0.01)$, temos:

$$-\frac{n\mu}{\sqrt{n}\sigma} = -\frac{\sqrt{n}\mu}{\sigma} = z$$

o que implica que, se permitirmos:

$$n \geq z^2 \sigma^2 / \mu^2$$

temos uma probabilidade garantida inferior a 0,01. A implicação é que, desde que μ seja positivo, podemos encontrar um n que minimiza a probabilidade de uma perda. Esta é uma versão da lei dos grandes números: quando n é grande, nosso lucro médio por empréstimo converge para o ganho esperado μ .

Com x fixo, agora podemos perguntar qual valor de n precisamos para que a probabilidade ser 0,01? No nosso exemplo, se distribuirmos:

```
z <- qnorm(0.01)
n <- ceiling((z^2*(x-1)^2*p*(1-p))/(1*p + x*(1-p))^2)
n
#> [1] 22163
```

empréstimos, a probabilidade de perda é de aproximadamente 0,01 e esperamos ganhar um total de:

```
n*(loss_per_foreclosure*p + x * (1-p))
#> [1] 14184320
```

dólares! Podemos confirmar isso com uma simulação de Monte Carlo:

```
p <- 0.04
x <- 0.05*180000
profit <- replicate(B, {
  draws <- sample( c(x, loss_per_foreclosure), n,
  prob=c(1-p, p), replace = TRUE)
  sum(draws)
})
mean(profit)
#> [1] 14176656
```

Portanto, essa parece ser uma decisão óbvia. Como resultado, seu funcionário decide deixar o banco e abrir sua própria empresa de empréstimos hipotecários de alto risco. Alguns meses depois, o banco de seu ex-funcionário declara falência. Um livro é escrito e, eventualmente, um filme é feito contando o erro cometido por seu funcionário e muitos outros. O que aconteceu?

O esquema de seu ex-funcionário foi baseado principalmente nesta fórmula matemática:

$$\text{SE}[(X_1 + X_2 + \dots + X_n)/n] = \sigma/\sqrt{n}$$

Ao fazer n grande, minimizamos o erro padrão do nosso lucro por empréstimo. No entanto, para que esta regra seja seguida, os X s devem ser eventos independentes: a inadimplência de uma pessoa deve ser independente da inadimplência de outras pessoas. Observe que, no caso de calcular a média do **mesmo** evento repetidamente, um exemplo extremo de eventos que não são independentes, obtemos um erro padrão que é \sqrt{n} vezes maior:

$$\text{SE}[(X_1 + X_1 + \dots + X_1)/n] = \text{SE}[nX_1/n] = \sigma > \sigma/\sqrt{n}$$

Para criar uma simulação mais realista do que a simulação original executada por seu colega, vamos assumir que haja um evento global que afete todas as pessoas com hipotecas de alto risco e mude sua probabilidade de pagamento. Assumiremos que, com uma probabilidade de 50 a 50, todas as probabilidades aumentam ou diminuem levemente para algo entre 0,03 e 0,05. Mas isso acontece com todos de uma vez, não apenas com uma pessoa. Esses eventos não são mais independentes.

```
p <- 0.04
x <- 0.05*180000
profit <- replicate(B, {
  new_p <- 0.04 + sample(seq(-0.01, 0.01, length = 100), 1)
  draws <- sample( c(x, loss_per_foreclosure), n,
  prob=c(1-new_p, new_p), replace = TRUE)
  sum(draws)
})
```

Observe que o lucro esperado ainda é grande:

```
mean(profit)
#> [1] 14181574
```

Entretanto, a probabilidade de o banco ter ganhos negativos dispara:

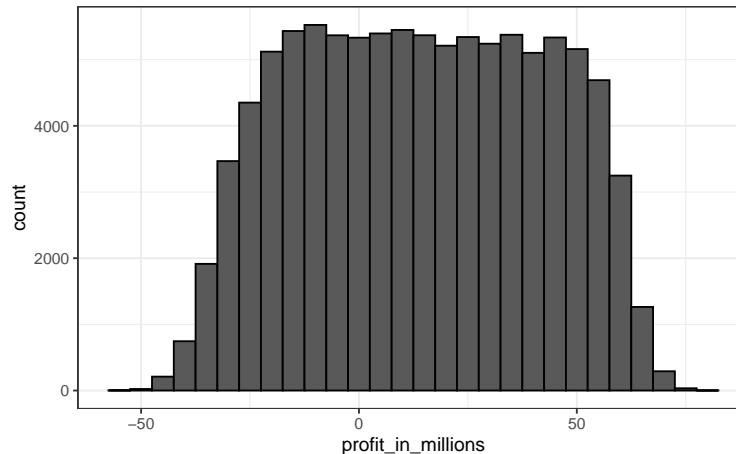
```
mean(profit < 0)
#> [1] 0.348
```

Ainda mais preocupante é que a probabilidade de perder mais de US\$ 10 milhões é:

```
mean(profit < -10000000)
#> [1] 0.24
```

Para entender como isso acontece, veja a distribuição:

```
data.frame(profit_in_millions=profit/10^6) %>%
ggplot(aes(profit_in_millions)) +
geom_histogram(color="black", binwidth = 5)
```



A teoria quebra completamente e a variável aleatória tem muito mais variabilidade do que o esperado. O colapso financeiro de 2007 ocorreu, dentre outras coisas, a “especialistas” financeiros que assumiram a independência quando não era esse o caso.

14.12 Exercícios

1. Crie uma variável aleatória S com os ganhos do seu banco, se você fizer 10.000 empréstimos, a uma taxa de juros de 0,3 e uma perda \$200.000 em cada execução de hipoteca (retomada de propriedade). Dica: use o código que mostramos na seção anterior, mas altere os parâmetros.
2. Execute uma simulação de Monte Carlo com 10.000 resultados para S . Faça um histograma dos resultados.
3. Qual é o valor esperado de S ?
4. Qual é o erro padrão de S ?
5. Suponha que façamos empréstimos de \$180.000. Qual deve ser a taxa de juros para o nosso valor esperado ser 0?
6. (Difícil) Qual deve ser a taxa de juros para que a probabilidade de perda de dinheiro seja de uma em 20? Em notação matemática, qual deve ser a taxa de juros para $\Pr(S < 0) = 0.05$?

7. Se o banco deseja minimizar as chances de perder dinheiro, qual das seguintes opções **não** faz com que as taxas de juros subam?

- a. Um grupo menor de empréstimos.
- b. Uma maior probabilidade de inadimplência.
- c. Uma menor probabilidade necessária de perda de dinheiro.
- d. O número de simulações de Monte Carlo.