

Jeidsan A. da C. Pereira

R para Data Science

Solução dos exercícios

To Shao Yong (邵雍),
for sharing a secret joy with simple words;

月到天心处，风来水面时。
一般清意味，料得少人知。

and

To Hongzhi Zhengjue (宏智禅师),
for sharing the peace of an ending life with simple words.

梦幻空华，六十七年；
白鸟淹没，秋水连天。

Conteúdo

Prefácio	xi
Prefácio	xi
Pendências	xi
I Explorar	1
1 Visualização de dados com <code>ggplot2</code>	3
1.1 Introdução	3
1.2 Primeiros passos	3
1.3 Mapeamentos estéticos	8
1.4 Problemas comuns	15
1.5 Facetas	15
1.6 Objetos geométricos	21
1.7 Transformações estatísticas	27
1.8 Ajustes de posição	33
1.9 Sistemas de coordenadas	37
1.10 A gramática em camadas de gráficos	39
2 Fluxo de trabalho: o básico	41
2.1 O básico de programação	41
2.2 O que há em um nome?	41
2.3 Chamando funções	41

3 Transformação de dados com dplyr	45
3.1 Introdução	45
3.2 Filtrar linhas com filter()	45
3.3 Comparações	45
3.4 Ordenar linhas com arrange()	52
3.5 Selecionar colunas com select()	56
3.6 Adicionar novas variáveis com mutate()	58
3.7 Resumos agrupados com summarize()	62
3.8 Mudanças agrupadas (e filtros)	70
4 Fluxo de trabalho: scripts	81
4.1 Executando códigos	81
4.2 Diagnósticos Rstudio	81
5 Análise exploratória de dados	83
5.1 Introdução	83
5.2 Perguntas	83
5.3 Variação	83
5.4 Valores faltantes	89
5.5 Covariação	92
5.6 Padrões e modelos	134
5.7 Chamadas ggplot2	134
5.8 Aprendendo mais	135
6 Fluxo de trabalho: projetos	137
6.1 O que é real?	137
6.2 Onde sua análise vive?	137
6.3 Caminhos e diretórios	137
6.4 Projetos RStudio	137
6.5 Resumo	137
II Wrangle	139

7 Tibbles com <code>tibble</code>	141
7.1 Introdução	141
7.2 Criando tibbles	141
7.3 Tibbles <i>versus</i> <code>data.frame</code>	141
7.4 Interagindo com códigos mais antigos	141
8 Importando dados com <code>readr</code>	151
8.1 Introdução	151
8.2 Começando	151
8.3 Analisando um vetor	155
8.4 Analisando um arquivo	159
8.5 Escrevendo em um arquivo	159
8.6 Outros tipos de dados	159
9 Arrumando dados com <code>tidyverse</code>	161
9.1 Introdução	161
9.2 Dados arrumados (Tidy Data)	161
9.3 Espalhando e reunindo	166
9.4 Separando e unindo	169
9.5 Valores faltantes	170
9.6 Estudo de caso	172
9.7 Dados desarrumados (não tidy)	176
10 Dados relacionais com <code>dplyr</code>	177
10.1 Introdução	177
10.2 <code>nycflights13</code>	177
10.3 Chaves (<code>keys</code>)	178
10.4 Mutating joins	179
10.5 Filtering joins	189
10.6 Problemas de joins	197
10.7 Operações de conjuntos	197

11 Strings com <code>stringr</code>	199
11.1 Introdução	199
11.2 O básico de <code>string</code>	199
11.3 Combinando padrões com expressões regulares	202
11.4 Ferramentas	205
11.5 Outros tipos de padrões	207
11.6 Outros usos para expressões regulares	207
11.7 <code>stringi</code>	208
12 Fatores com <code>forcats</code>	209
12.1 Introdução	209
12.2 Criando fatores	209
12.3 General Social Survey	209
12.4 Modificando a ordem dos fatores	217
12.5 Modificando níveis de fatores	222
13 Datas e horas com <code>lubridate</code>	227
13.1 Introdução	227
13.2 Criando data/horas	227
13.3 Componentes de data-hora	229
13.4 Intervalos de tempo	237
13.5 Fusos horários	239
III Programar	241
14 Pipes com <code>magrittr</code>	243
14.1 Introdução	243
14.2 Alternativas ao piping	243
14.3 Quando não usar o pipe	243
14.4 Outras ferramentas do <code>magrittr</code>	243

<i>Contents</i>	vii
15 Funções	245
15.1 Introdução	245
15.2 Quando você deveria escrever uma função?	245
15.3 Funções são para humanos e computadores	249
15.4 Execução condicional	251
15.5 Argumentos de funções	257
15.6 Retorno de valores	257
15.7 Ambiente	257
16 Vetores	259
16.1 Introdução	259
16.2 O Básico de vetores	259
16.3 Tipos importantes de vetores atômicos	259
16.4 Usando vetores atômicos	261
16.5 Vetores recursivos (listas)	263
16.6 Atributos	264
16.7 Vetores aumentados	264
17 Iteração com <code>purrr</code>	267
17.1 Introdução	267
17.2 Loops <code>for</code>	267
17.3 Variações do loop <code>for</code>	276
17.4 Loops <code>for</code> versus funcionais	280
17.5 As funções <code>map</code>	281
17.6 Lidando com falhas	282
17.7 Fazendo <code>map</code> com vários argumentos	282
17.8 Walk	282
17.9 Outros padrões para loops <code>for</code>	282
IV Modelar	283

18 O básico de modelos com <code>modelr</code>	285
18.1 Introdução	285
18.2 Um modelo simples	285
18.3 Visualizando modelos	289
18.4 Fórmulas e famílias de modelos	293
18.5 Valores faltantes	300
18.6 Outras famílias de modelos	300
19 Construção de modelos	301
19.1 Introdução	301
19.2 Por que diamantes de baixa qualidade são mais caros?	301
19.3 O que afeta o número de voos diários?	304
19.4 Aprendendo mais sobre modelos	308
20 Muitos modelos com <code>purrr</code> e <code>broom</code>	309
20.1 Introdução	309
20.2 <code>gapminder</code>	309
20.3 List-columns	309
20.4 Criando list-columns	309
20.5 Simplificando list-columns	309
20.6 Criando dados tidy com <code>broom</code>	309
V Comunicar	311
21 R Markdown	313
21.1 Introdução	313
21.2 O Básico de R Markdown	313
21.3 Formatação de texto com <code>markdown</code>	313
21.4 Trechos de código	313
21.5 Resolução de problemas	313
21.6 Header YAML	313
21.7 Aprendendo mais	313

<i>Contents</i>	ix
22 Gráficos para comunicação com <code>ggplot2</code>	315
22.1 Introdução	315
22.2 Rótulo	315
22.3 Anotações	315
22.4 Escalas	315
22.5 Dando zoom	315
22.6 Temas	315
22.7 Salvando seus gráficos	315
22.8 Aprendendo mais	315
23 Formatos R Markdown	317
23.1 Introdução	318
23.2 Opções de saída	318
23.3 Documentos	318
23.4 Notebooks	318
23.5 Apresentações	318
23.6 Dashboards	318
23.7 Interatividade	318
23.8 Sites	318
23.9 Outros formatos	318
23.10 Aprendendo mais	318
24 Fluxo de trabalho de R Markdown	319
Apêndice	319
A Apendice A - Estudo de caso sobre a arrumação de dados com <code>tidyverse</code>	321
Introdução	321
Estudo de caso	321



Prefácio

Esta página serviu para estudo e prática com o pacote R Bookdown e contém a solução encontrada por mim para os exercícios propostos no livro R para Data Sciente, de Hadley Wickham e Garret Grolemund, publicado no Brasil em 2019 pela Alta Books Editora [Wickham and Grolemund, 2019].

Por se tratar de um produto construído durante o processo de aprendizagem, o conteúdo pode conter erros, tanto no texto em si, como na lógica utilizada para solução dos exercícios.

Dúvidas ou sugestões de melhoria podem ser encaminhadas para o e-mail *jeidsan.pereira@gmail.com*¹.

Pendências

- No PDF, o prefácio está sendo exibido duas vezes no sumário;
- Exercício 1.7.4;
- Exercício 2.3.3;
- Exercício 3.5.1;
- Exercício 3.7.1;
- Exercício 3.8.1;
- Exercício 4.2.1;
- Exercício 4.2.2;
- Exercício 5.5.4;
- Exercício 5.5.8;
- Exercício 5.5.9;
- Exercício 5.5.12;
- Exercício 8.2.3;
- Exercício 8.3.6;
- Exercício 9.5.2;
- Exercício 10.2.1;
- Exercício 10.3.2;
- Exercício 10.3.3;

¹<mailto:jeidsan.pereira@gmail.com>

- Exercício 10.4.3;
- Exercícios do capítulo 11;
- Exercício 12.3.3;
- Exercício 13.3.3;
- Exercício 13.3.7;
- Exercício 15.2.7;
- Exercício 15.3.2;
- Exercício 15.3.3;
- Exercício 15.3.4;
- Exercícios da seção 15.5;
- Exercício 16.3.3;
- Exercício 16.4.3;
- Exercício 16.4.5;
- Exercício 16.5.1;
- Exercício 17.3.1;
- Exercício 17.4.1;
- Exercício 17.4.1;
- Exercícios da seção 17.5;
- Exercícios da seção 17.9;
- Avaliar melhor a resposta ao Exercício 18.2.1;
- Exercício 18.2.3;
- Exercício 18.3.4;
- Buscar uma explicação para o fato de não haver diferença no Exercício 18.4.1;
- Exercício 18.4.3;
- Exercício 18.4.4;
- Devo incluir os aspectos matemáticos para mostrar a relação entre as variáveis citadas no Exercício 19.2.2?
- Revisar Exercício 19.2.3;
- Revisar Exercício 19.3.4;
- Exercício 19.3.5;
- Exercício 19.3.6;
- Exercício 19.3.7;
- Exercício 19.3.8;

Parte I

Explorar



1

Visualização de dados com ggplot2

Para a correta execução dos códigos desse capítulo, utilizaremos algumas configurações específicas.

Inicialmente, precisaremos carregar o pacote `nycflights13`, que contém os dados de todos os voos da cidade de Nova York em 2013.

```
library(nycflights13)
library(gridExtra)

## 
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
## 
##     combine
```

1.1 Introdução

Não temos exercícios nesta seção.

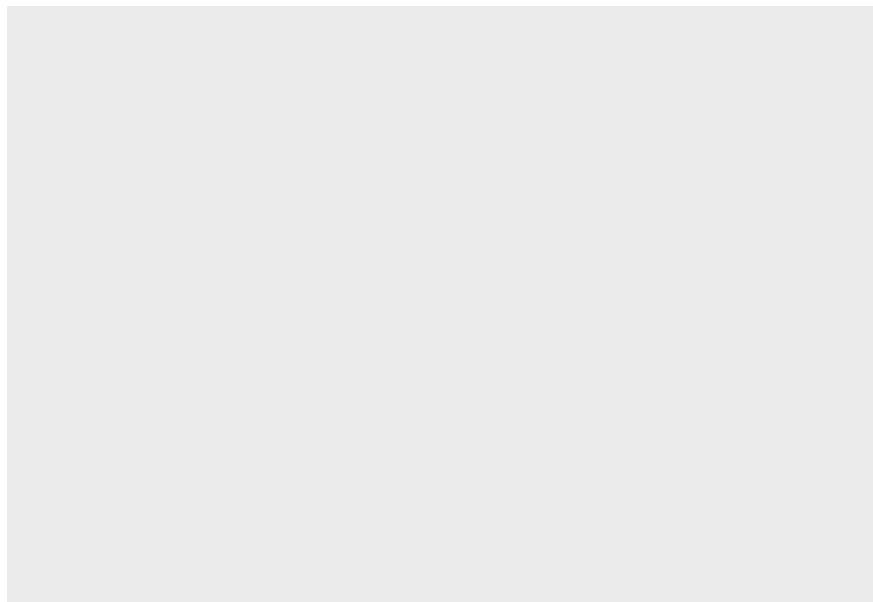
1.2 Primeiros passos

Exercício 1.2.1

Execute `ggplot(data=mpg);.` O que você vê?

Solução.

```
ggplot(data=mpg) +  
  tema
```



É exibido um quadro em branco. Este quadro contém o sistema de coordenadas sobre o qual serão desenhados os gráficos que pretendemos exibir.

Exercício 1.2.2

Quantas linhas existem em `mtcars`? Quantas colunas?

Solução.

```
dim(mtcars)
```

```
## [1] 32 11
```

R.: Existem 32 linhas e 11 colunas.

Exercício 1.2.3

O que a variável `drv` descreve?

Solução. Executamos o comando `?mpg` no console no R e a página de ajuda foi aberta. Nela encontramos o significado de cada variável do conjunto de dados.

A variável descreve o tipo de tração dos carros analisados, onde `f` significa tração dianteira, `r` significa tração traseira e `4` significa tração nas quatro rodas.

Exercício 1.2.4

Faça um gráfico de dispersão de `hwy` versus `cyl`.

Solução.

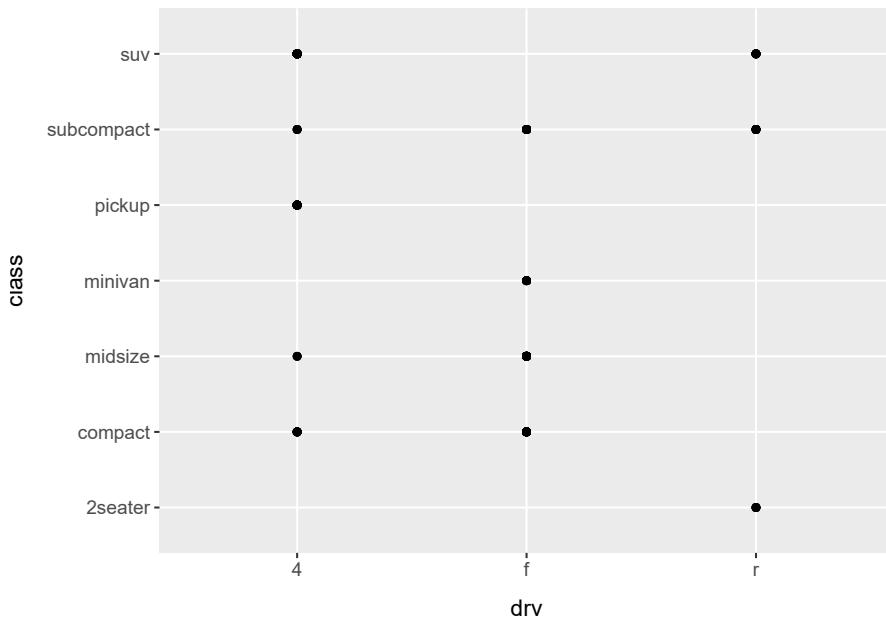
```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = hwy, y = cyl)) +  
  tema
```

**Exercício 1.2.5**

O que acontece se você fizer um gráfico de dispersão de `class` versus `drv`? Por que esse gráfico não é útil?

Solução.

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = drv, y = class)) +
  tema
```

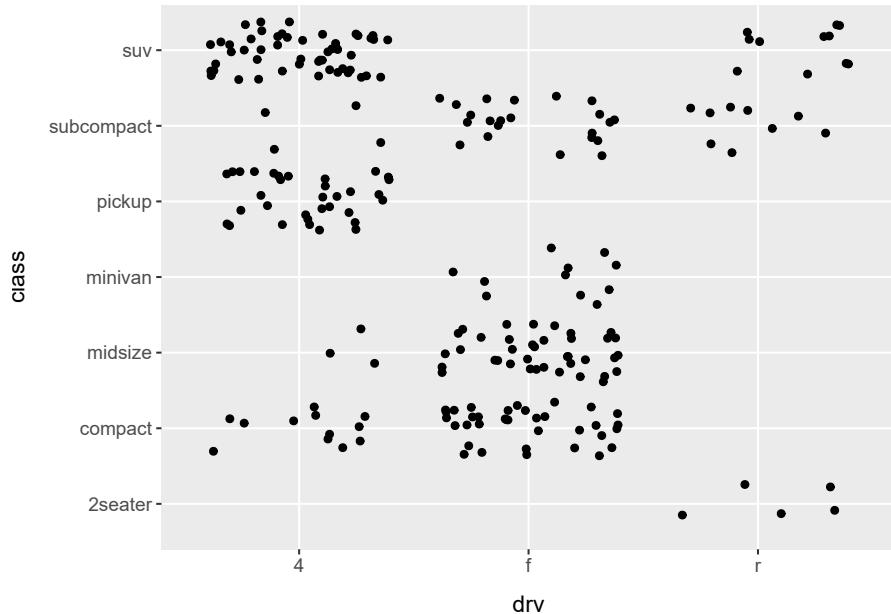


Apesar de serem exibidos dados no gráfico, nenhuma informação substancial é extraída, uma vez que o tipo de tração não está (a princípio) relacionado com a categoria do carro. Outro fator que torno o gráfico pouco informativo é que há, por exemplo, diversas SUVs com tração nas 4 rodas, contudo os valores ficam sobrepostos no gráfico, não dando dimensão do quanto de dados temos.

Abaixo seguem duas opções de como trazer mais informação ao gráfico:

- a primeira opção adiciona um ruído aos dados (`position = jitter` ou `geom_jitter()`) de modo que não haja sobreposição;

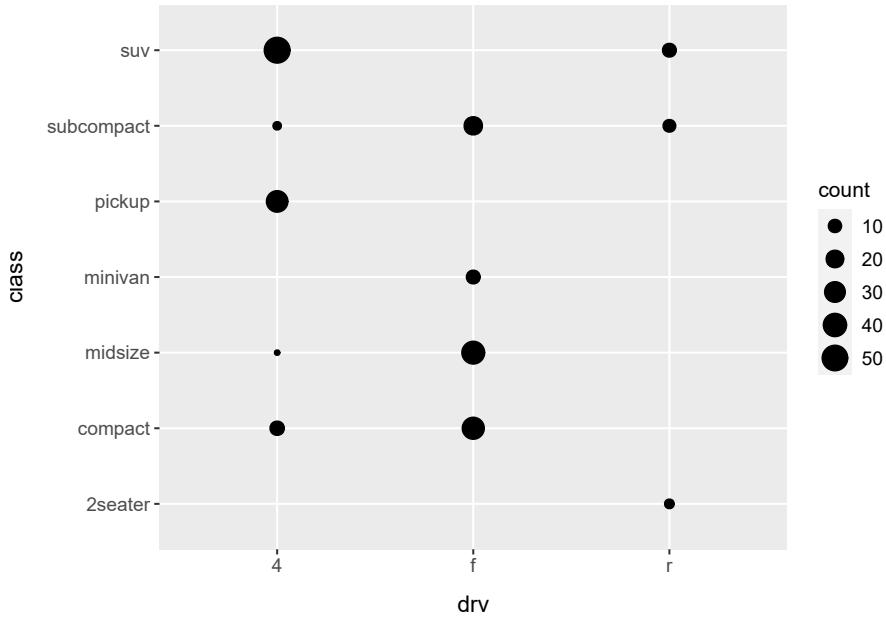
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = drv, y = class), position = "jitter") +
  tema
```



- a segunda opção, bem mais avançada, adiciona uma estética de `size` considerando a quantidade de registros.

```
mpg %>%
  group_by(class, drv) %>%
  summarise(count = n()) %>%
  ggplot(mapping = aes(x = drv, y = class, size = count)) +
  geom_point() +
  tema
```

```
## `summarise()` has grouped output by 'class'. You can override using the
## `.` argument.
```

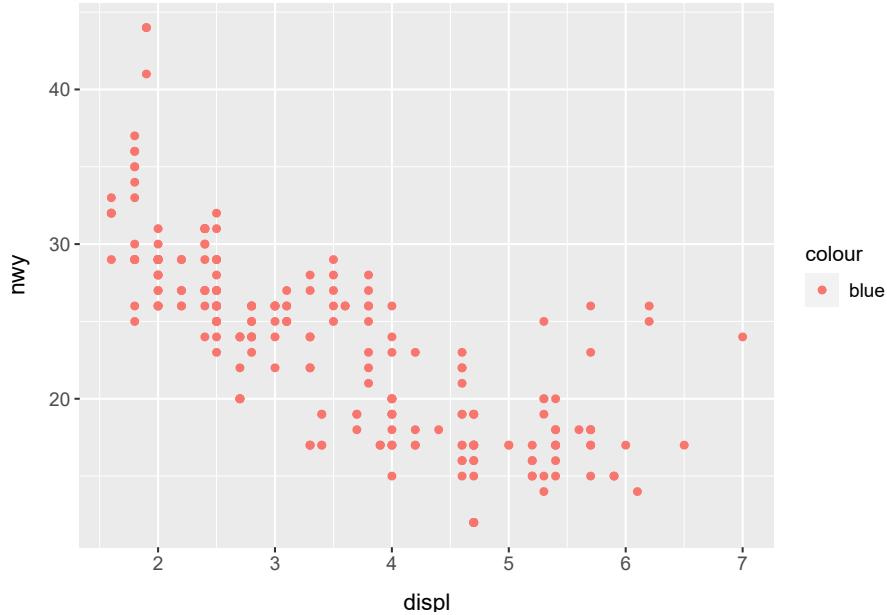


1.3 Mapeamentos estéticos

Exercício 1.3.1

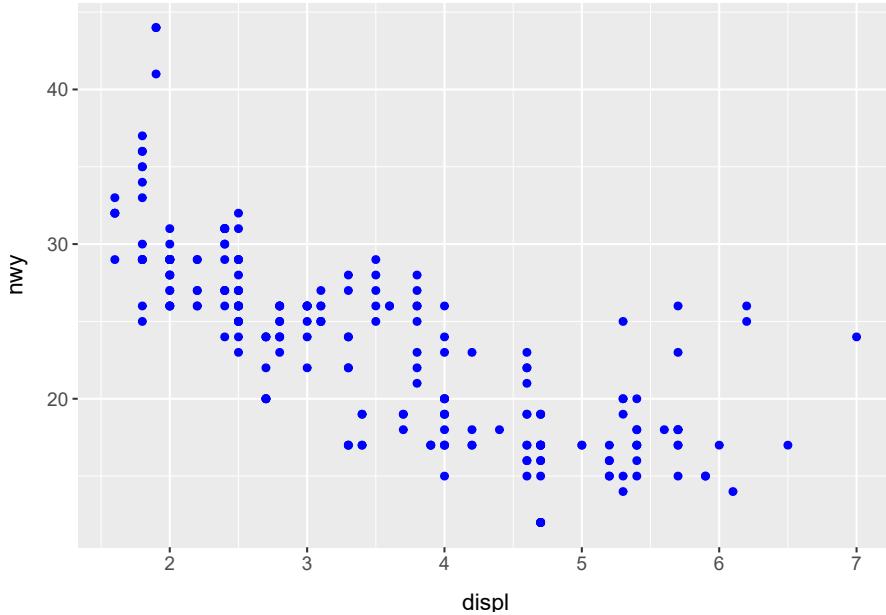
O que há de errado com este código? Por que os pontos não estão azuis?

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy, color = "blue")) +  
  tema
```



Solução. Ao invés de atribuir uma cor aos elementos de `geom_point`, o atributo `color` foi passado como uma estética. O gráfico deveria ser construído da seguinte maneira:

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy), color = "blue") +  
  tema
```



Exercício 1.3.2

Quais variáveis em `mpg` são categóricas? Quais variáveis são contínuas? Como você pode ver essa informação quando executa `mpg`?

Solução. Usando `?mpg` vemos que as variáveis categóricas são: `manufacturer`, `model`, `trans`, `drv`, `fl` e `class`. As variáveis contínuas são: `displ`, `cty`, `hwy`.

Exercício 1.3.3

Mapeie uma variável contínua para `color`, `size` e `shape`. Como essas estéticas se comportam de maneira diferente para variáveis categóricas e contínuas?

Solução.

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy, color = displ)) +  
  tema
```



```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy, size = displ)) +  
  tema
```



```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy, shape = displ)) +  
  tema
```

```
## Error in `geom_point()`:  
## ! Problem while computing aesthetics.  
## i Error occurred in the 1st layer.  
## Caused by error in `scale_f()`:  
## ! A continuous variable cannot be mapped to the shape aesthetic  
## i choose a different aesthetic or use `scale_shape_binned()`
```

Quando possível, a biblioteca *ggplot* apresenta a estética em um gradiente, como em `color` e `size`. Porém, nem sempre isso é possível, como vemos em `shape`, que só pode ser utilizada com variáveis discretas ou categóricas.

Exercício 1.3.4

O que acontece se você mapear a mesma variável a várias estéticas?

Solução.

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy, size = class, color = class, shape = class)) +  
  tema
```

```
## Warning: Using size for a discrete variable is not advised.
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values because  
## more than 6 becomes difficult to discriminate; you have 7. Consider  
## specifying shapes manually if you must have them.
```

```
## Warning: Removed 62 rows containing missing values (`geom_point()`).
```



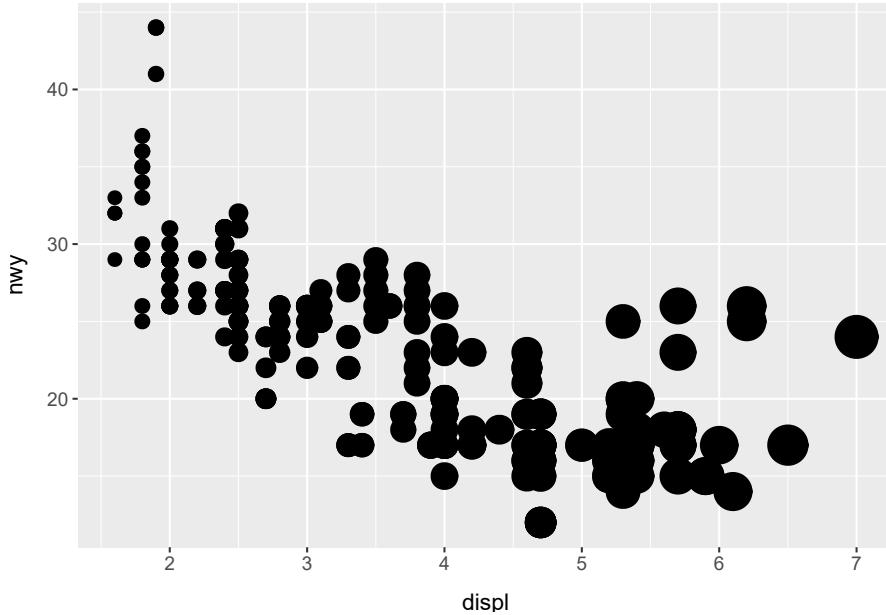
Os valores da variável serão representados de modo a atender todas as estéticas simultaneamente, por exemplo, no gráfico acima é dada uma cor, um formato e um tamanho específicos para cada classe de veículo. Os veículos de dois lugares são exibidos como um disco rosa pequeno.

Exercício 1.3.5

O que a estética `stroke` faz? com que formas ela trabalha?

Solução.

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, stroke = displ)) +
  tema
```



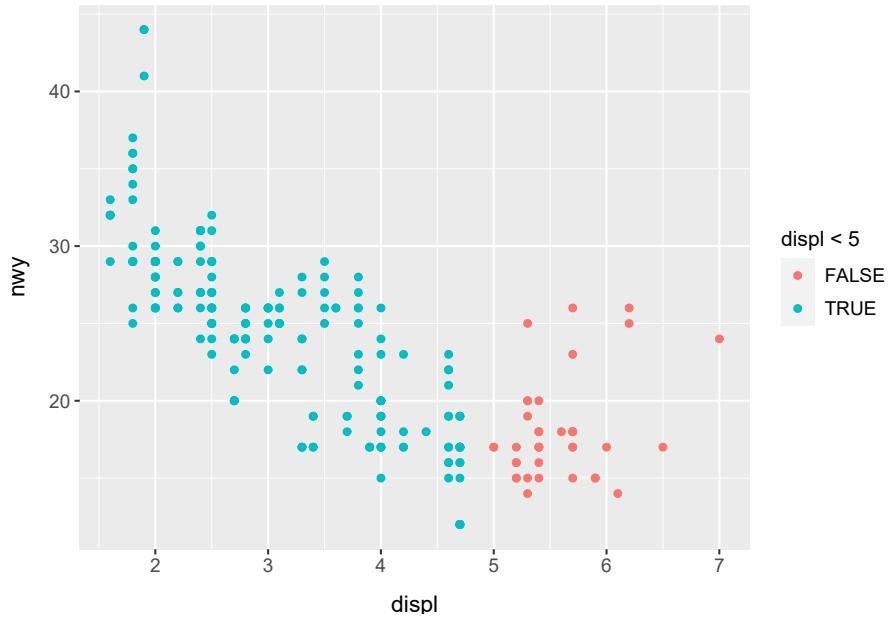
A estética `stroke` controla a espessura do ponto ou elemento a ser representado.

Exercício 1.3.6

O que acontece se você mapear uma estética a algo diferente de um nome de variável, como `aes(color = displ < 5)`?

Solução.

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy, color = displ < 5)) +  
  tema
```



A expressão é avaliada para cada um dos valores da variável e o resultado é utilizado para plotagem da estética no gráfico.

1.4 Problemas comuns

Não temos exercícios nessa seção.

1.5 Facetas

Exercício 1.5.1

O que acontece se você criar facetas em uma variável contínua?

Solução.

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~ displ) +
  tema
```

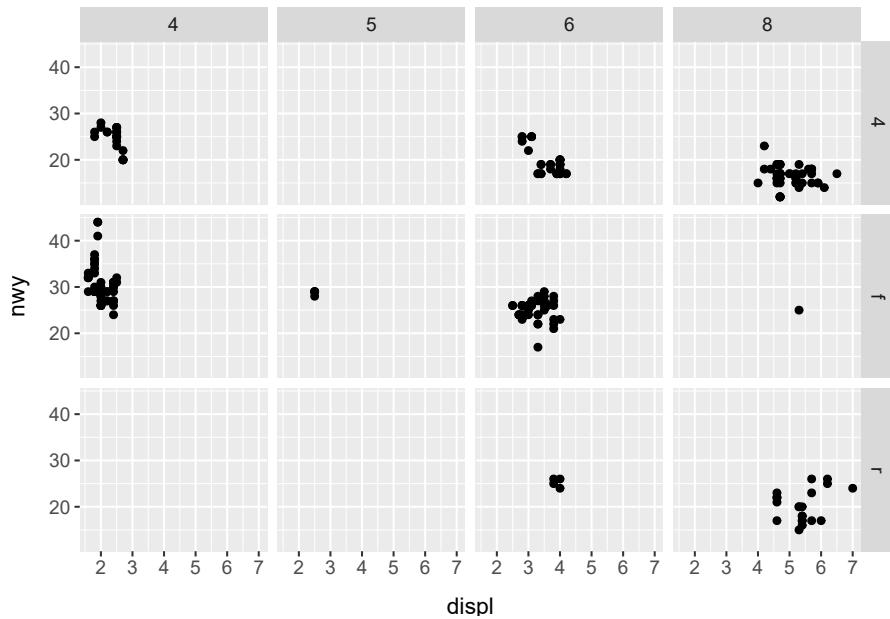


O *ggplot* se encarrega de dividir o conjunto em classes e toma o ponto médio de cada classe para realizar a quebra em facetas.

Exercício 1.5.2

O que significam as células em branco em um gráfico com `facet_grid(drv ~ cyl)`? Como elas se relacionam a este gráfico?

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_grid(drv ~ cyl) +
  tema
```



Solução. Significa que para aquela combinação de variáveis, não há nenhum valor observado. Por exemplo, não há nenhum veículo com 5 cilindros e tração nas quatro rodas.

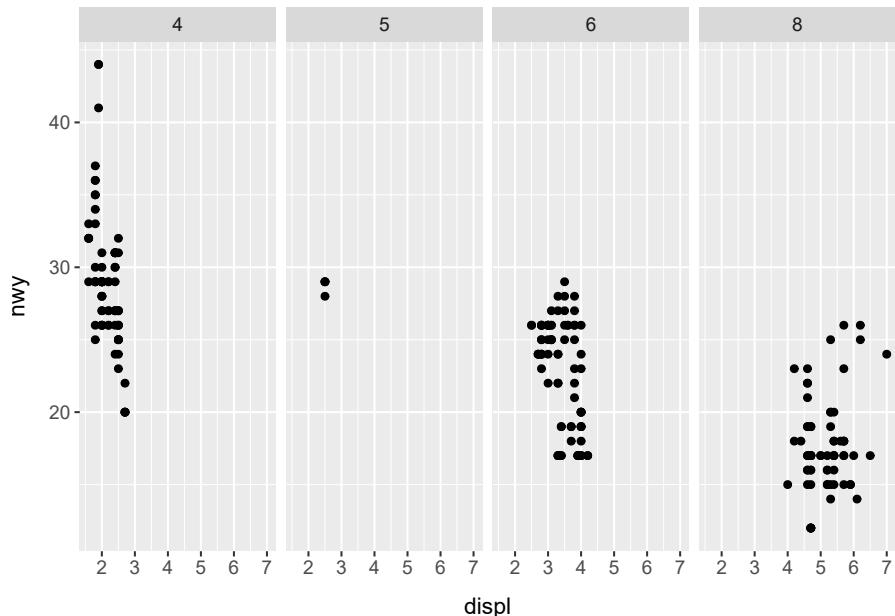
Exercício 1.5.3

Que gráficos o código a seguir faz? O que . faz?

```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  facet_grid(drv ~ .) +  
  tema
```



```
ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy)) +  
  facet_grid(. ~ cyl) +  
  tema
```

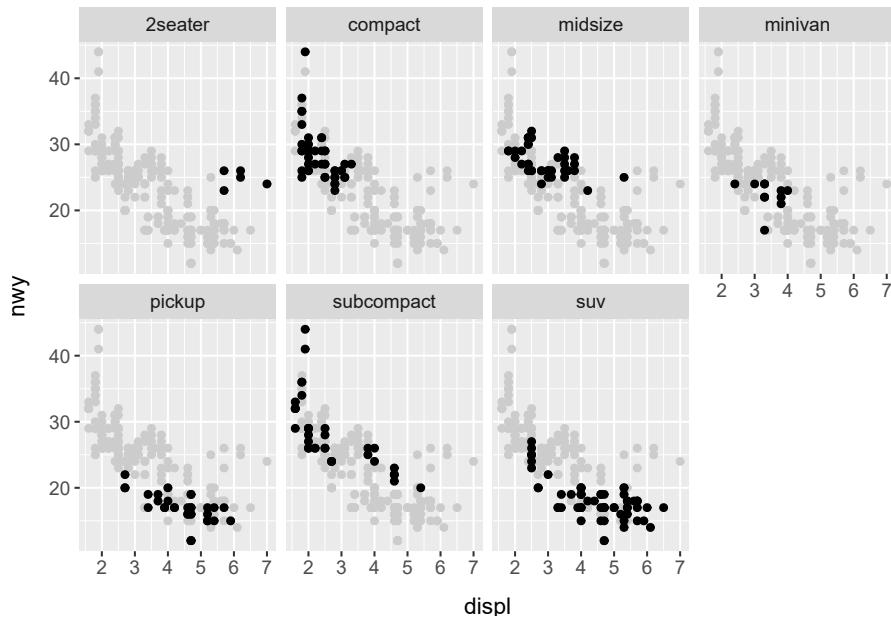


Solução. São gerados os gráficos de dispersão segregados pelas variáveis `drv` e `cyl`, respectivamente. O `.` indica que não queremos considerar nenhuma segregação naquela dimensão do `grid` (linha ou coluna).

Exercício 1.5.4

Pegue o primeiro gráfico em facetas dessa seção.

```
ggplot(data = mpg) +
  geom_point(data = transform(mpg, class = NULL), mapping = aes(x = displ, y = hwy), color = "gray80") +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~ class, nrow = 2) +
  tema
```



Quais são as vantagens de usar facetas, em vez de estética de cor? Quais são as desvantagens? Como o equilíbrio poderia mudar se você tivesse um conjunto de dados maior?

Solução. A principal vantagem no uso de facetas é que fica mais fácil analisar os dados quando eles estão separados em seu próprio contexto, contudo visualizá-los assim dificulta a comparação entre grupos.

Exercício 1.5.5

Leia `?facet_wrap`. O que `nrow` faz? o que `ncol` faz? Quais outras opções controlam o layout de painéis individuais? Por que `facet_grid()` não tem variáveis `nrow` e `ncol`?

Solução.

```
?facet_wrap
```

Os atributos `ncol` e `nrow` são utilizados pelo `facet_wrap` para determinar o número de colunas ou linhas (respectivamente) nas quais serão distribuídos os gráficos segregados. Esses atributos não figuram em `facet_grid()` pelo fato deste já organizar as facetas retangularmente.

Exercício 1.5.6

Ao usar `facet_grid()` você normalmente deveria colocar a variável com níveis mais singulares nas colunas. Por quê?

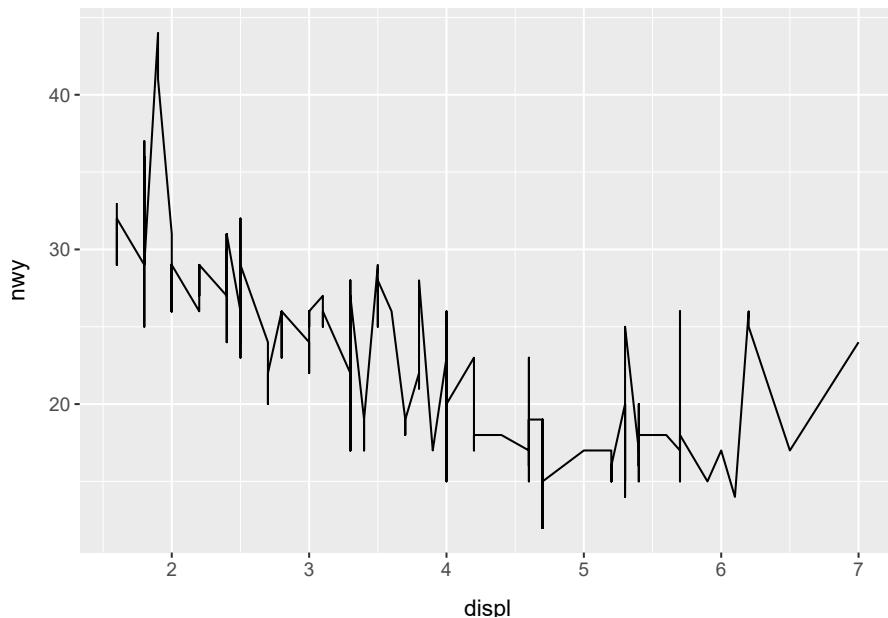
Solução. Para melhor aproveitamento do espaço em tela.

1.6 Objetos geométricos**Exercício 1.6.1**

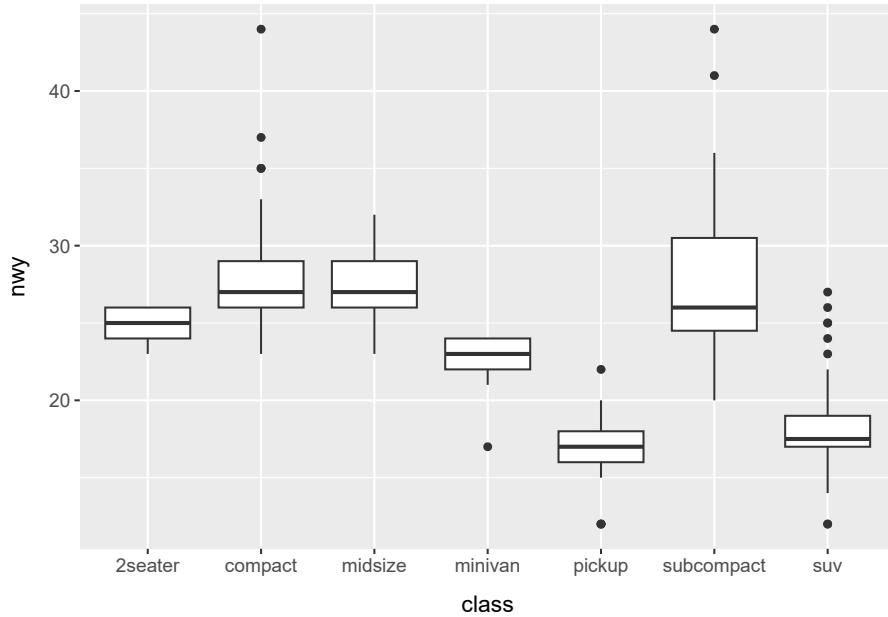
Que *geom* você usaria para desenhar um gráfico de linha? Um diagrama de caixas (*boxplot*)? Um histograma? Um gráfico de área?

Solução.

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +  
  geom_line() +  
  tema
```

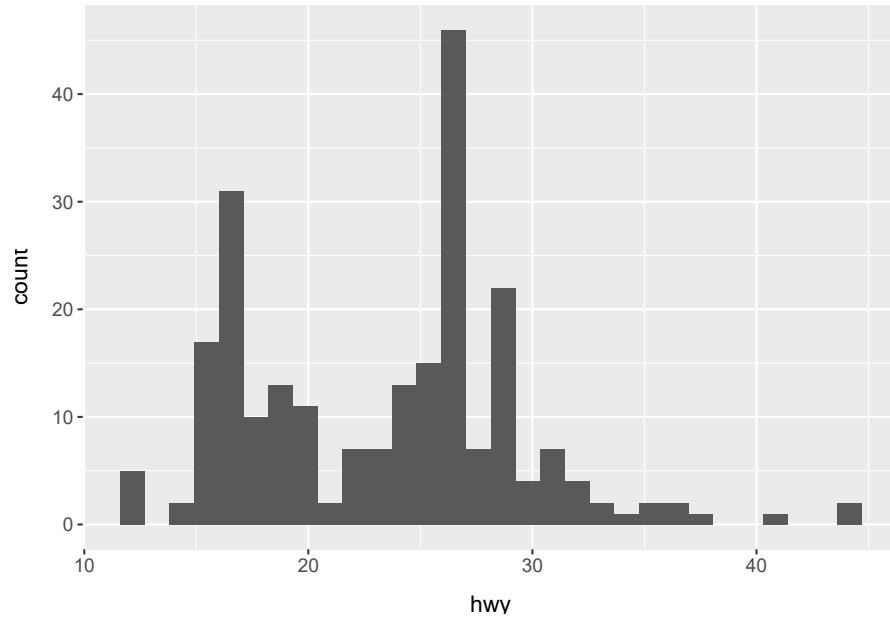


```
ggplot(data = mpg) +  
  geom_boxplot(mapping = aes(y = hwy, x = class)) +  
  tema
```

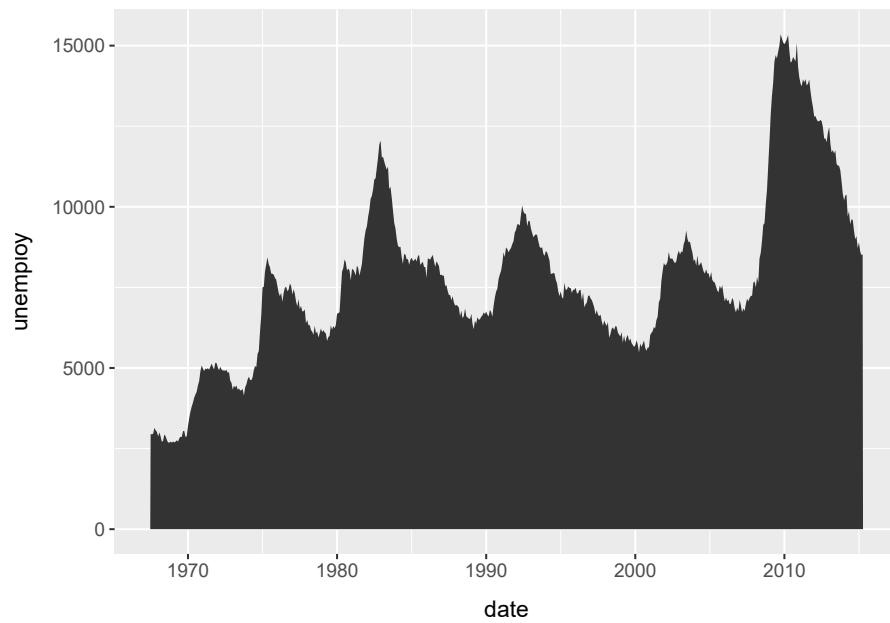


```
ggplot(data = mpg, mapping = aes(x = hwy)) +  
  geom_histogram() +  
  tema
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
ggplot(data = economics, mapping = aes(x = date, y = unemploy)) +  
  geom_area() +  
  tema
```



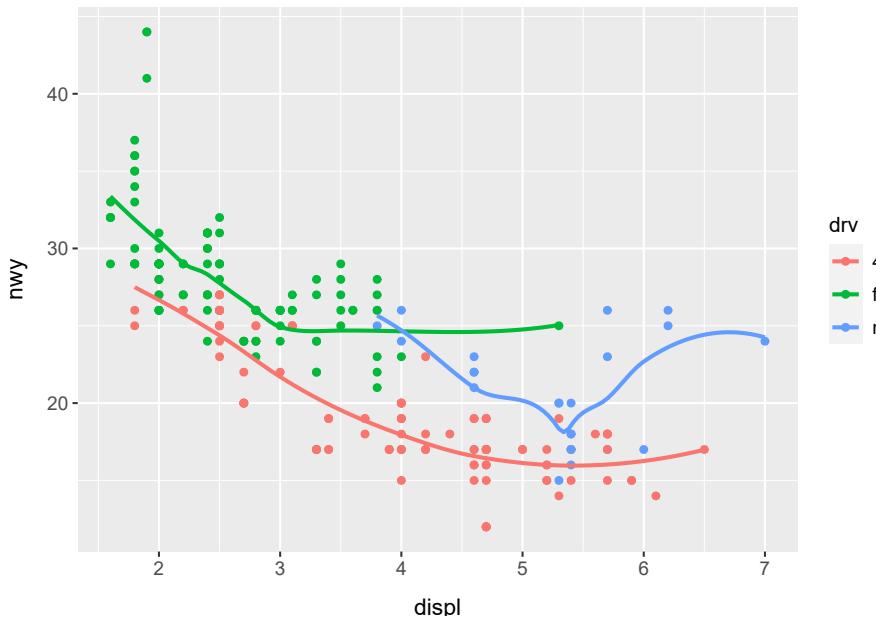
Podem ser utilizados, respectivamente as *geoms*: *line*, *boxplot*, *histogram* e *area*.

Exercício 1.6.2

Execute este código em sua cabeça e preveja como será o resultado. Depois execute o código no R e confira suas previsões:

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = drv)) +
  geom_point() +
  geom_smooth(se = FALSE) +
  tema
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Solução. O gráfico bateu com a expectativa.

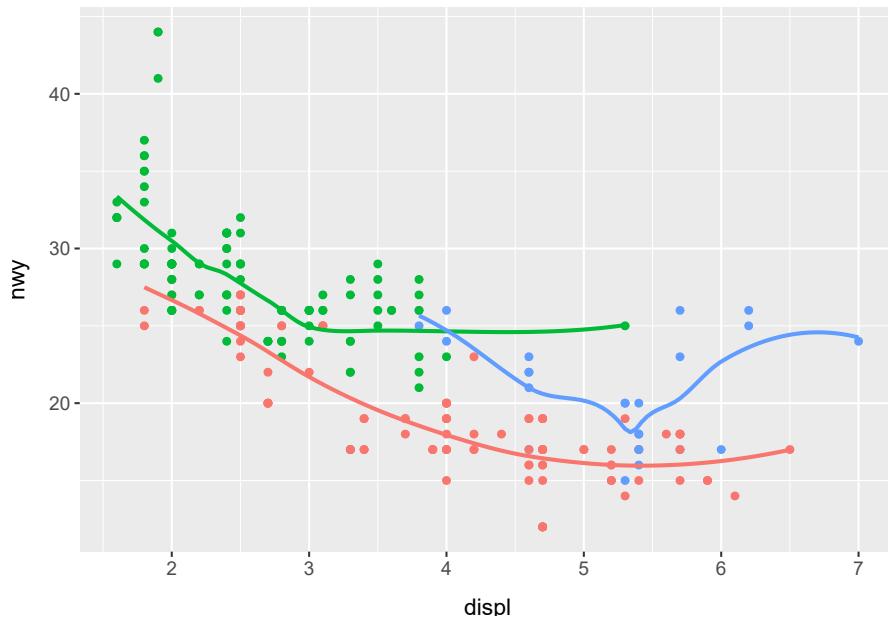
Exercício 1.6.3

O que o `show.legend = FALSE` faz? O que acontece se você removê-lo? Por que você acha que usei isso anteriormente no capítulo?

Solução.

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = drv)) +
  geom_point(show.legend = FALSE) +
  geom_smooth(se = FALSE, show.legend = FALSE) +
  tema
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Ele indica que, para a camada à qual se aplica, não serão geradas as legendas de identificação.

Exercício 1.6.4

O que o argumento `se` para `geom_smooth` faz?

Solução.

```
?geom_smooth
```

Esse argumento indica se o intervalo de confiança utilizado no processo de suavização da linha deve ou não ser exibido no gráfico.

Exercício 1.6.5

Esses dois gráficos serão diferentes? Por quê/por que não?

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth() +
  tema

ggplot() +
  geom_point(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_smooth(data = mpg, mapping = aes(x = displ, y = hwy)) +
  tema
```

Solução. Os gráficos serão iguais. Ao informar os parâmetros `data` e `mapping` na função `ggplot` essas atributos serão considerados como globais, sendo utilizado em todos as camadas do gráfico, a menos que alguma das camadas os sobrecreva. No segundo gráfico, não são definidos parâmetros globais, porém, o mesmo parâmetro é passado para ambas as camadas, sendo assim, a única diferença é o código estar duplicado.

Exercício 1.6.6

Recrie o código R necessário para gerar os seguintes gráficos:



Solução.

```
a <- ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +  
  geom_point() +  
  geom_smooth(se = FALSE) +  
  tema  
  
b <- ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +  
  geom_point() +  
  geom_smooth(mapping = aes(group = drv), se = FALSE) +  
  tema  
  
c <- ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color = drv)) +  
  geom_point() +  
  geom_smooth(se = FALSE) +  
  tema  
  
d <- ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +  
  geom_point(mapping = aes(color = drv)) +  
  geom_smooth(se = FALSE) +  
  tema  
  
e <- ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +  
  geom_point(mapping = aes(color = drv)) +  
  geom_smooth(mapping = aes(linetype = drv), se = FALSE) +  
  tema  
  
f <- ggplot(data = mpg, mapping = aes(x = displ, y = hwy, fill = drv)) +  
  geom_point(color = "white", shape = 21, size = 3, stroke = 2) +  
  tema
```

1.7 Transformações estatísticas

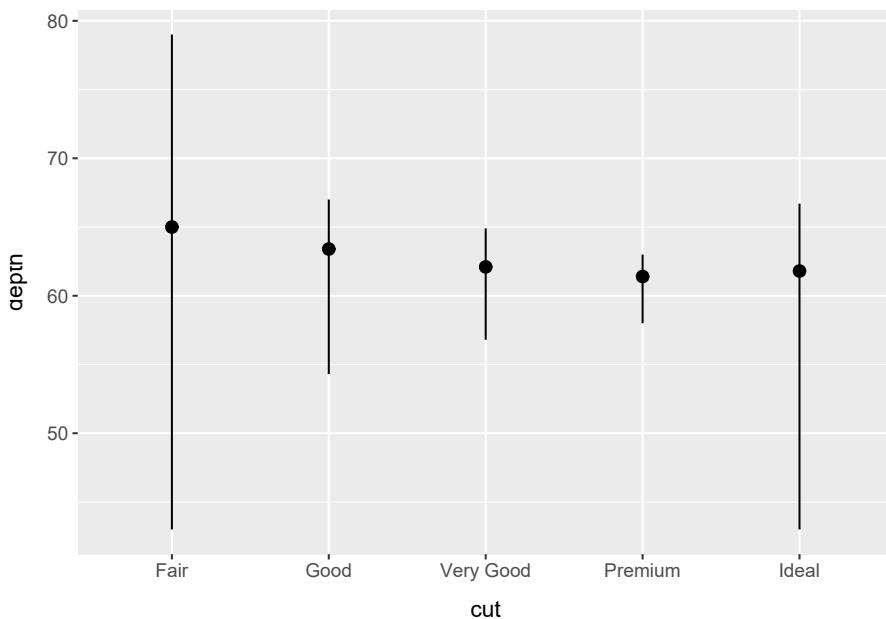
Exercício 1.7.1

Qual é o `geom` padrão associado ao `stat_summary()`? Como você poderia reescrever o gráfico anterior usando essa função `geom`, em vez da função `stat`?

Solução.

```
?stat_summary
```

```
ggplot(data = diamonds) +  
  stat_summary(  
    mapping = aes(x = cut, y = depth),  
    fun.min = min,  
    fun.max = max,  
    fun = median  
  ) +  
  tema
```



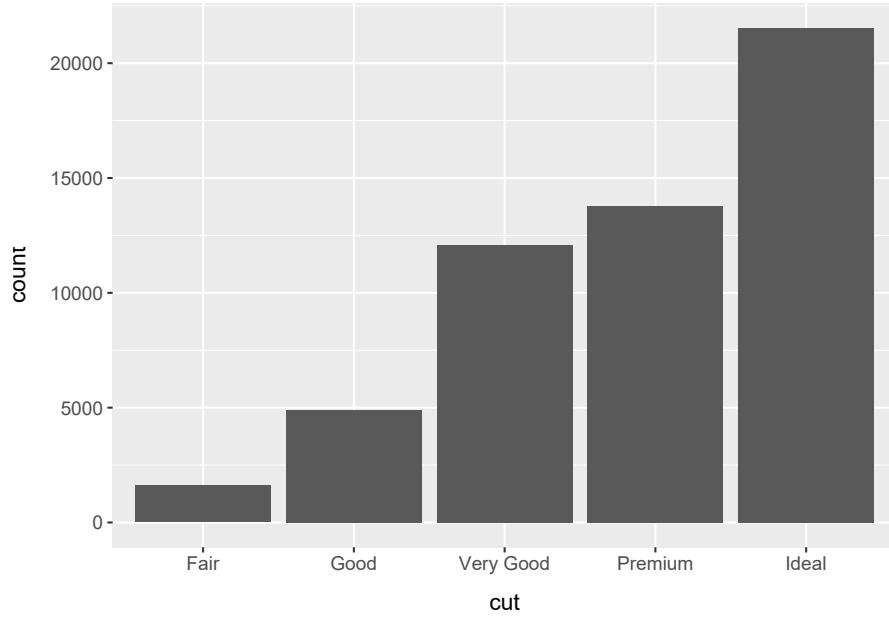
A geom associada é a `geom_pointrange` e o gráfico poderia ser reescrito da seguinte maneira.

Exercício 1.7.2

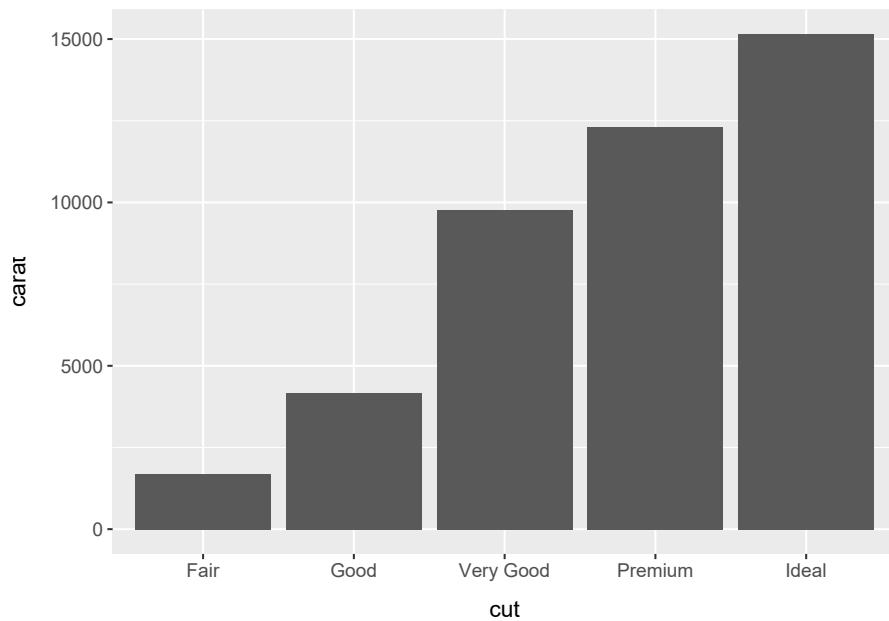
O que `geom_col()` faz? Qual é a diferença entre ele e `geom_bar()`?

Solução.

```
ggplot(data = diamonds, mapping = aes(x = cut)) +  
  geom_bar() +  
  tema
```



```
ggplot(data = diamonds, mapping = aes(x = cut, y = carat)) +  
  geom_col() +  
  tema
```



Enquanto no `geom_bar` a altura das barras representa uma transformação estatística relacionada às observações (como `count`, por exemplo), no `geom_col` podemos exibir o acumulado (soma) de uma variável para cada categoria exibida.

Exercício 1.7.3

A maioria dos `geoms` e `stats` vem em pares, que são quase sempre usados juntos. Leia a documentação e faça uma lista de todos os pares. O que eles têm em comum?

Solução.

#	Geom	Stat
01	Blank	Identity
02	Curve	Identity
03	Segment	Identity
04	Path	Identity
05	Line	Identity
06	Step	Identity
07	Poligon	Identity
08	Raster	Identity
09	Rect	Identity
10	Tile	Identity
11	Ribbon	Identity
12	Area	Identity
13	Align	?
14	ABLine	?
15	HLine	?
16	Density	Density
17	DotPlot	?
18	Freqpoly	Bin
19	Histogram	Bin
20	Col	Identity
21	Bar	Count
22	Label	Identity
23	Text	Identity
24	Jitter	Identity
25	Point	Identity
26	Quantile	Quantile
27	Rug	Identity
28	Boxplot	Boxplot
29	Violin	YDensity
30	Count	Sum
31	Bin 2D	Bin 2D
32	Density 2D	Density 2D

#	Geom	Stat
33	Hex	Bin Hex
34	Cross Bar	Identity
35	Error Bar	Identity
36	Line Range	Identity
37	Point Range	Identity
38	Map	Identity
39	Contour	Contour
40	Contour Filled	Contour Filled

Exercício 1.7.4

Quais variáveis `stat_smooth()` calcula? Quais parâmetros controlam seu comportamento?

Solução.

```
?stat_smooth
```

Exercício 1.7.5

Em nosso gráfico de barra de *proportion*, precisamos configurar `group = 1`. Por quê?
Em outras palavras, qual é o problema com esses dois gráficos?

```
ggplot(data = diamonds) +  
  geom_bar(mapping = aes(x = cut, y = after_stat(prop), group = 1)) +  
  tema
```



Solução.

```
ggplot(data = diamonds) +  
  geom_bar(mapping = aes(  
    x = cut,  
    fill = color,  
    y = after_stat(prop),  
    group = color  
  )) +  
  tema
```



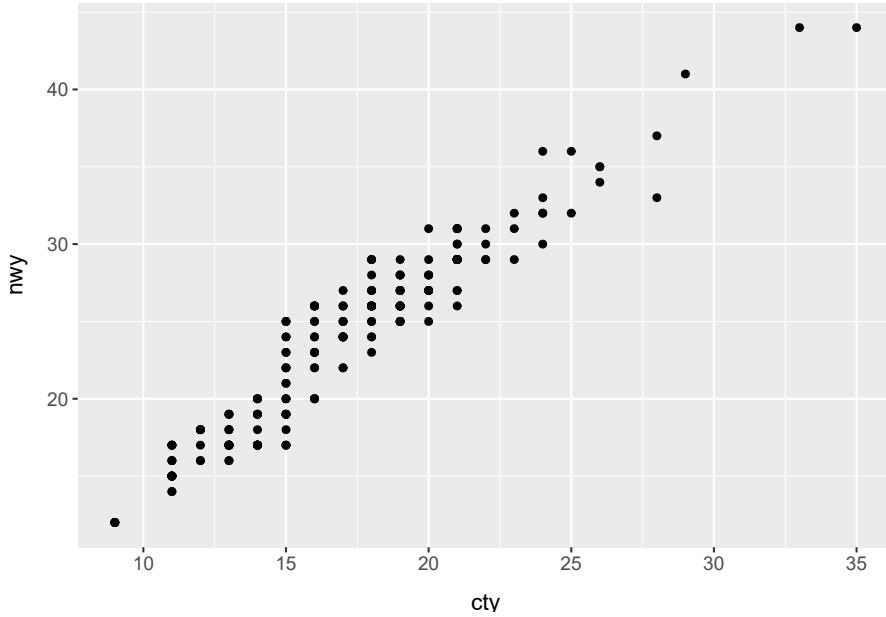
Quando estamos trabalhando com proporções (ou estatísticas em geral), é importante destacar para o `ggplot` qual agrupamento ele deve considerar, caso contrário ele irá considerar um único grupo e dará uma impressão incorreta ao gráfico. No primeiro exemplo, foi utilizado `group = 1` (e, na verdade, poderia ser qualquer valor) apenas para indicar que deveria ser realizado um agrupamento.

1.8 Ajustes de posição

Exercício 1.8.1

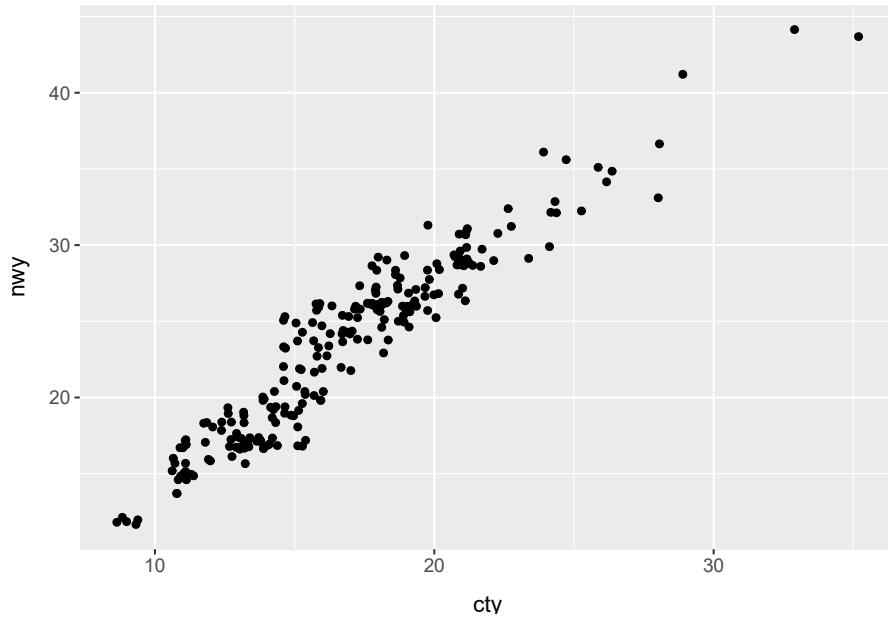
Qual é o problema com este gráfico? Como você poderia melhorá-lo?

```
ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +  
  geom_point() +  
  tema
```



Solução. Há pontos sobrepostos. Uma melhoria poderia ser usar `geom_jitter` em lugar de `geom_point`.

```
ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +  
  geom_jitter() +  
  tema
```



Exercício 1.8.2

Quais parâmetros para `geom_jitter` controlam a quantidade de oscilação?

Solução. Conforme a documentação disposta em `?geom_jitter`, são utilizados os parâmetros `width` e `height`.

Exercício 1.8.3

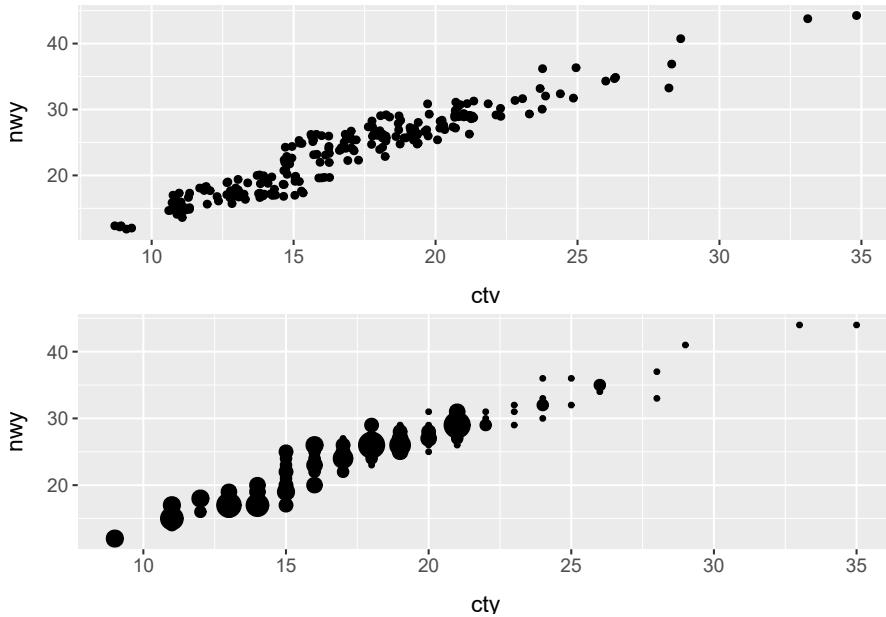
Compare o contraste entre `geom_jitter` e `geom_count`.

Solução.

```
a <- ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
  geom_jitter() +
  tema

b <- ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
  geom_count(show.legend = FALSE) +
  tema

grid.arrange(a, b, nrow = 2)
```



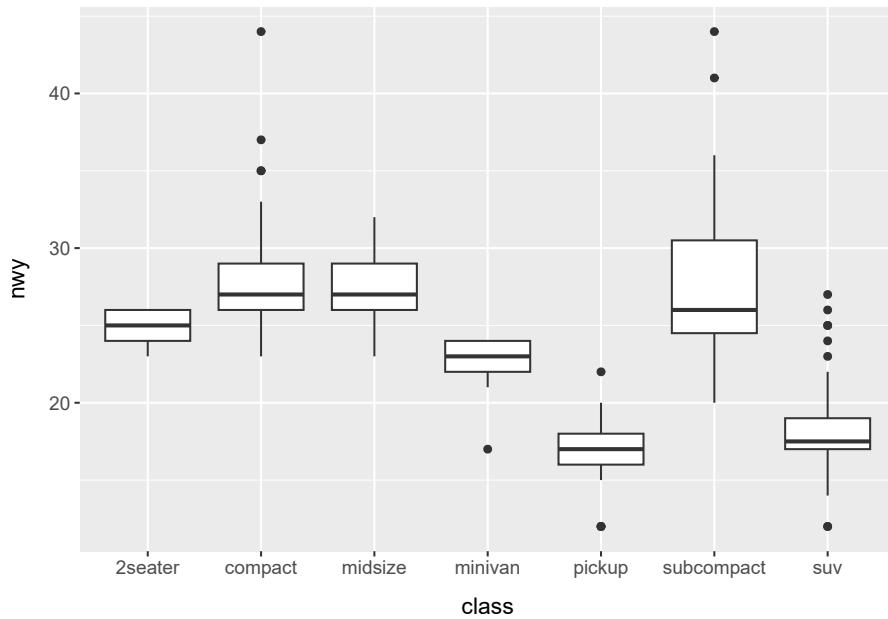
Para contornar o problema da sobreposição de pontos, `geom_jitter` adiciona um pequeno ruído aleatório aos dados, enquanto o `geom_count` contabiliza os pontos sobrepostos e altera o tamanho dos pontos conforme a quantidade.

Exercício 1.8.4

Qual é o ajuste de posição padrão para `geom_boxplot()`? Crie uma visualização do conjunto de dados `mpg` que demonstre isso.

Solução. Conforme pode ser visto em `?geom_boxplot`, a `position` padrão é a `dodge2`.

```
ggplot(data = mpg, mapping = aes(x = class, y = hwy)) +  
  geom_boxplot() +  
  tema
```



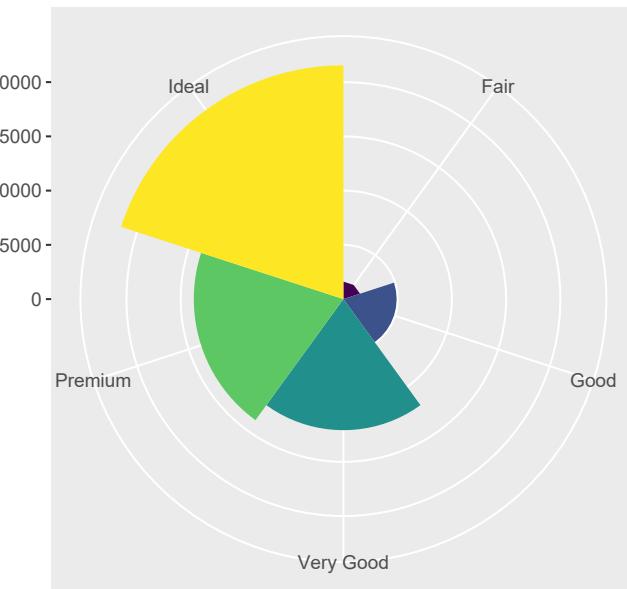
1.9 Sistemas de coordenadas

Exercício 1.9.1

Transforme um gráfico de barras empilhadas em um gráfico de pizza usando `coord_polar()`.

Solução.

```
ggplot(data = diamonds, mapping = aes(x = cut, fill = cut)) +
  geom_bar(show.legend = FALSE, width = 1) +
  coord_polar() +
  labs(x = NULL, y = NULL) +
  theme(aspect.ratio = 1) +
  tema
```



Exercício 1.9.2

O que `labs()` faz? Leia a documentação.

Solução. Usando o comando `?labs`, vimos que esta função é utilizada para definir labels do gráfico, como título, subtítulo, títulos de eixos, etc.

Exercício 1.9.3

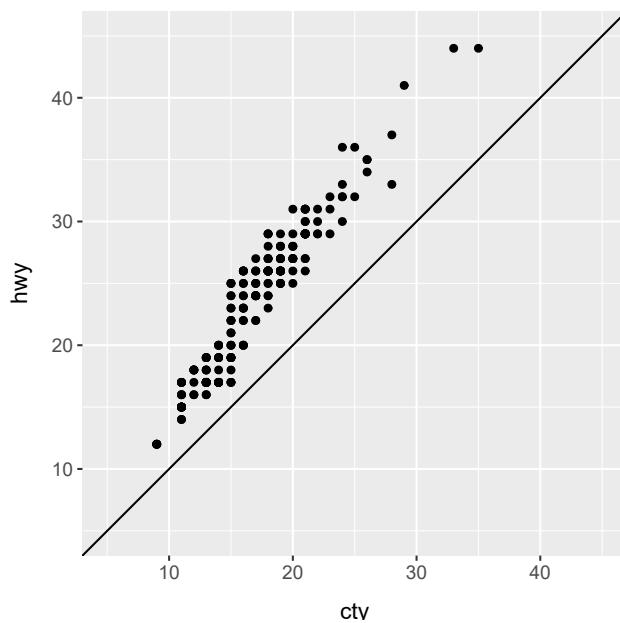
Qual é a diferença entre `coord_quickmap()` e `coord_map()`?

Solução. Usando o comando `?coord_map`, notamos que a diferença é que enquanto `coord_map()` não preserva linhas retas, sendo assim, mais custoso computacionalmente, o `coord_quickmap()` o faz.

Exercício 1.9.4

O que o gráfico a seguir lhe diz sobre a relação entre `mpg` de cidade e estrada? Por que `coord_fixed()` é importante? O que `geom_abline()` faz?

```
ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +  
  geom_point() +  
  geom_abline() +  
  coord_fixed(ratio = 1, xlim = c(5, 45), ylim = c(5, 45)) +  
  tema
```



Solução. O gráfico mostra a relação entre a eficiência na cidade e na estrada. O `coord_fixed()` força que seja mantida uma proporção entre os eixos x e y, isto é, garante que uma unidade no eixo y corresponda a um número determinado de unidades no eixo x. A razão padrão é 1. Já o `geom_abline()` define uma linha de referência diagonal ao gráfico, no nosso caso, a linha é a reta dada por $y - x = 0$.

1.10 A gramática em camadas de gráficos

Não temos exercícios nesta seção.



2

Fluxo de trabalho: o básico

2.1 O básico de programação

Não temos exercícios nesta seção.

2.2 O que há em um nome?

Não temos exercícios nesta seção.

2.3 Chamando funções

Exercício 2.3.1

Por que esse código não funciona?

```
my_variable <- 10  
my_varIable
```

Solução. Foi atribuído um valor à variável `my_variable`, contudo depois tentou-se utilizar essa variável, porém a escrita está incorreta e o R não reconheceu a variável. O R diferencia letras maiúsculas e minúsculas, isto é, as variáveis `my_variable` e `my_varIable` são distintas.

Exercício 2.3.2

Ajuste cada um dos seguintes comandos de R para que executem corretamente.

```
library(tidyverse)

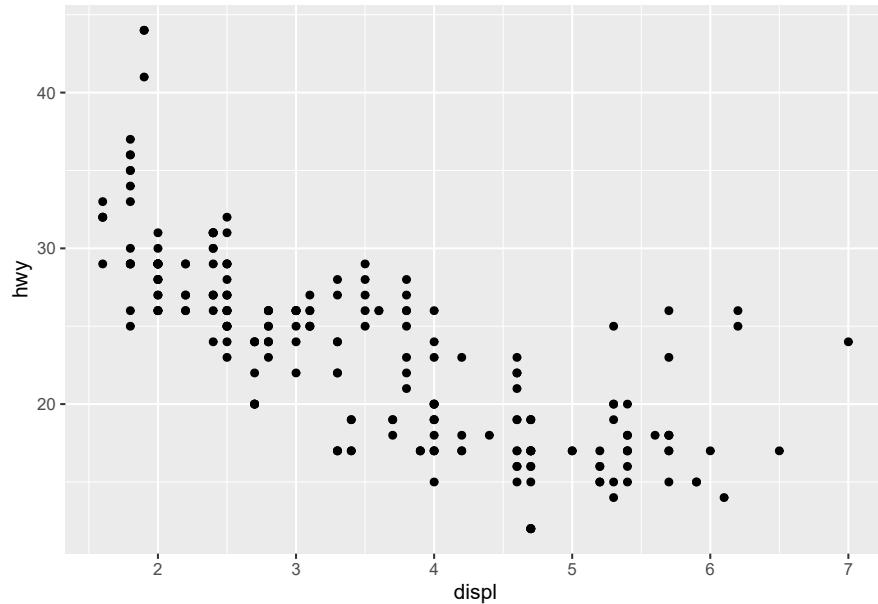
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))

filter(mpg, cyl = 8)
filter(diamond, carat > 3)
```

Solução.

```
library(tidyverse)

ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```



```
filter(mpg, cyl == 8)
```

```
## # A tibble: 70 x 11
##   manufacturer model      displ year  cyl trans drv   cty   hwy fl class
##   <chr>        <chr>     <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
## 1 audi         a6 quattro  4.2  2008     8 auto~ 4       16    23 p   mids~
## 2 chevrolet    c1500 sub~  5.3  2008     8 auto~ r       14    20 r   suv
```

```
## 3 chevrolet c1500 sub~ 5.3 2008 8 auto~ r 11 15 e suv
## 4 chevrolet c1500 sub~ 5.3 2008 8 auto~ r 14 20 r suv
## 5 chevrolet c1500 sub~ 5.7 1999 8 auto~ r 13 17 r suv
## 6 chevrolet corvette 5.7 1999 8 manu~ r 16 26 p 2sea~
## 7 chevrolet corvette 5.7 1999 8 auto~ r 15 23 p 2sea~
## 8 chevrolet corvette 6.2 2008 8 manu~ r 16 26 p 2sea~
## 9 chevrolet corvette 6.2 2008 8 auto~ r 15 25 p 2sea~
## 10 chevrolet corvette 6.2 2008 8 auto~ r 15 25 p 2sea~
## # i 60 more rows
```

```
filter(diamonds, carat > 3)
```

```
## # A tibble: 32 x 10
##   carat cut     color clarity depth table price     x     y     z
##   <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl>
## 1 3.01 Premium I     I1    62.7   58 8040  9.1   8.97  5.67
## 2 3.11 Fair      J     I1    65.9   57 9823  9.15  9.02  5.98
## 3 3.01 Premium F     I1    62.2   56 9925  9.24  9.13  5.73
## 4 3.05 Premium E     I1    60.9   58 10453  9.26  9.25  5.66
## 5 3.02 Fair      I     I1    65.2   56 10577  9.11  9.02  5.91
## 6 3.01 Fair      H     I1    56.1   62 10761  9.54  9.38  5.31
## 7 3.65 Fair      H     I1    67.1   53 11668  9.53  9.48  6.38
## 8 3.24 Premium H     I1    62.1   58 12300  9.44  9.4   5.85
## 9 3.22 Ideal     I     I1    62.6   55 12545  9.49  9.42  5.92
## 10 3.5  Ideal     H     I1    62.8   57 12587  9.65  9.59  6.03
## # i 22 more rows
```

Exercício 2.3.3

Pressione Alt-Shift-K. O que acontece? Como você pode chegar ao mesmo resultado usando os menus?

Solução. x



3

Transformação de dados com dplyr

Para este capítulo, necessitaremos das seguintes configurações iniciais:

```
not_cancelled <- flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay))
```

3.1 Introdução

Não temos exercícios nesta seção.

3.2 Filtrar linhas com filter()

Não temos exercícios nesta seção.

3.3 Comparações

Exercício 3.3.1

Encontre todos os voos que:

- a. Tiveram um atraso de duas horas ou mais na chegada.
- b. Foram para Houston (IAH ou HOU).
- c. Foram operados pela United, American ou Delta.

- d. Partiram em julho, agosto e setembro.
- e. Chegaram com mais de duas horas de atraso, mas não saíram atrasados.
- f. Atrasaram pelo menos uma hora, mas compensaram mais de 30 minutos durante o trajeto.
- g. Saíram entre meia-noite e 6h (incluindo esses horários).

Solução.

- a. Tiveram um atraso de duas horas ou mais na chegada.

```
filter(flights, arr_delay >= 120)
```

```
## # A tibble: 10,200 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>     <int>        <int>     <dbl>    <int>        <int>
## 1  2013     1     1      811        630     101 1047        830
## 2  2013     1     1      848       1835     853 1001       1950
## 3  2013     1     1      957       733     144 1056        853
## 4  2013     1     1     1114       900     134 1447       1222
## 5  2013     1     1     1505      1310     115 1638       1431
## 6  2013     1     1     1525      1340     105 1831       1626
## 7  2013     1     1     1549      1445      64 1912       1656
## 8  2013     1     1     1558      1359     119 1718       1515
## 9  2013     1     1     1732      1630      62 2028       1825
## 10 2013     1     1     1803      1620     103 2008       1750
## # i 10,190 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

- b. Foram para Houston (IAH ou HOU).

```
filter(flights, dest %in% c("IAH", "HOU"))
```

```
## # A tibble: 9,313 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>     <int>        <int>     <dbl>    <int>        <int>
## 1  2013     1     1      517        515      2     830        819
## 2  2013     1     1      533        529      4     850        830
## 3  2013     1     1      623        627     -4     933        932
## 4  2013     1     1      728        732     -4    1041       1038
```

```

## 5 2013 1 1 739 739 0 1104 1038
## 6 2013 1 1 908 908 0 1228 1219
## 7 2013 1 1 1028 1026 2 1350 1339
## 8 2013 1 1 1044 1045 -1 1352 1351
## 9 2013 1 1 1114 900 134 1447 1222
## 10 2013 1 1 1205 1200 5 1503 1505
## # i 9,303 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## # tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## # hour <dbl>, minute <dbl>, time_hour <dttm>

```

- c. Foram operados pela United, American ou Delta.

```
filter(flights, carrier %in% c("AA", "DL", "UA"))
```

```

## # A tibble: 139,504 x 19
##   year month day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>     <int>        <int>      <dbl>    <int>        <int>
## 1 2013 1 1 517 515 2 830 819
## 2 2013 1 1 533 529 4 850 830
## 3 2013 1 1 542 540 2 923 850
## 4 2013 1 1 554 600 -6 812 837
## 5 2013 1 1 554 558 -4 740 728
## 6 2013 1 1 558 600 -2 753 745
## 7 2013 1 1 558 600 -2 924 917
## 8 2013 1 1 558 600 -2 923 937
## 9 2013 1 1 559 600 -1 941 910
## 10 2013 1 1 559 600 -1 854 902
## # i 139,494 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## # tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## # hour <dbl>, minute <dbl>, time_hour <dttm>

```

- d. Partiram em julho, agosto e setembro.

```
filter(flights, month %in% c(7, 8, 9))
```

```

## # A tibble: 86,326 x 19
##   year month day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>     <int>        <int>      <dbl>    <int>        <int>

```

```

## 1 2013    7   1     1      2029    212    236    2359
## 2 2013    7   1     2      2359     3    344    344
## 3 2013    7   1    29     2245    104    151      1
## 4 2013    7   1    43     2130    193    322     14
## 5 2013    7   1    44     2150    174    300    100
## 6 2013    7   1    46     2051    235    304    2358
## 7 2013    7   1    48     2001    287    308    2305
## 8 2013    7   1    58     2155    183    335     43
## 9 2013    7   1   100     2146    194    327     30
## 10 2013   7   1   100     2245    135    337    135
## # i 86,316 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>

```

e. Chegaram com mais de duas horas de atraso, mas não saíram atrasados.

```
filter(flights, dep_delay <= 0, arr_delay > 120)
```

```

## # A tibble: 29 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>        <int>     <dbl>    <int>        <int>
## 1 2013    1    27    1419       1420      -1    1754       1550
## 2 2013    10    7    1350       1350      0    1736       1526
## 3 2013    10    7    1357       1359      -2    1858       1654
## 4 2013    10    16    657        700      -3    1258       1056
## 5 2013    11    1    658        700      -2    1329       1015
## 6 2013     3    18    1844       1847      -3      39       2219
## 7 2013     4    17    1635       1640      -5    2049       1845
## 8 2013     4    18    558        600      -2    1149       850
## 9 2013     4    18    655        700      -5    1213       950
## 10 2013    5    22    1827       1830      -3    2217       2010
## # i 19 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>

```

f. Atrasaram pelo menos uma hora, mas compensaram mais de 30 minutos durante o trajeto.

```
filter(flights, dep_delay >= 60 & dep_delay - arr_delay >= 30)
```

```
## # A tibble: 2,074 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>     <int>       <int>     <dbl>    <int>       <int>
## 1 2013     1     1     1716        1545      91    2140       2039
## 2 2013     1     1     2205        1720      285      46    2040
## 3 2013     1     1     2326        2130      116     131       18
## 4 2013     1     3     1503        1221      162    1803       1555
## 5 2013     1     3     1821        1530      171    2131       1910
## 6 2013     1     3     1839        1700      99    2056       1950
## 7 2013     1     3     1850        1745      65    2148       2120
## 8 2013     1     3     1923        1815      68    2036       1958
## 9 2013     1     3     1941        1759     102    2246       2139
## 10 2013    1     3     1950        1845      65    2228       2227
## # i 2,064 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

g. Saíram entre meia-noite e 6h (incluindo esses horários).

```
filter(flights, dep_time >= 0, dep_time <= 600)
```

```
## # A tibble: 9,344 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>     <int>       <int>     <dbl>    <int>       <int>
## 1 2013     1     1      517        515       2     830       819
## 2 2013     1     1      533        529       4     850       830
## 3 2013     1     1      542        540       2     923       850
## 4 2013     1     1      544        545      -1    1004      1022
## 5 2013     1     1      554        600      -6     812       837
## 6 2013     1     1      554        558      -4     740       728
## 7 2013     1     1      555        600      -5     913       854
## 8 2013     1     1      557        600      -3     709       723
## 9 2013     1     1      557        600      -3     838       846
## 10 2013    1     1      558        600      -2     753       745
## # i 9,334 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Exercício 3.3.2

Outro ajudante da filtragem do **dplyr** é `between()`. O que ele faz? Você consegue utilizá-lo para simplificar o código necessário para responder os desafios anteriores?

Solução. O `between` recebe três parâmetros e verifica se o primeiro está entre o segundo e o terceiro.

```
filter(flights, between(dep_time, 0, 600))
```

```
## # A tibble: 9,344 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>     <dbl>        <dbl>      <dbl>     <dbl>        <dbl>
## 1 2013     1     1      517         515        2       830        819
## 2 2013     1     1      533         529        4       850        830
## 3 2013     1     1      542         540        2       923        850
## 4 2013     1     1      544         545       -1      1004       1022
## 5 2013     1     1      554         600       -6      812        837
## 6 2013     1     1      554         558       -4      740        728
## 7 2013     1     1      555         600       -5      913        854
## 8 2013     1     1      557         600       -3      709        723
## 9 2013     1     1      557         600       -3      838        846
## 10 2013    1     1      558         600      -2      753        745
## # i 9,334 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Exercício 3.3.3

Quantos voos têm um `dep_time` faltante? Que outras variáveis estão faltando? O que essas linhas podem representar?

Solução.

```
count(flights, is.na(dep_time))
```

```
## # A tibble: 2 x 2
##   `is.na(dep_time)`     n
##   <lgl>                  <int>
## 1 FALSE                 328521
## 2 TRUE                  8255
```

```
summary(is.na(flights))
```

```
##      year       month       day     dep_time
##  Mode :logical  Mode :logical  Mode :logical  Mode :logical
##  FALSE:336776  FALSE:336776  FALSE:336776  FALSE:328521
##                                TRUE :8255
##      sched_dep_time   dep_delay     arr_time   sched_arr_time
##  Mode :logical  Mode :logical  Mode :logical  Mode :logical
##  FALSE:336776  FALSE:328521  FALSE:328063  FALSE:336776
##                                TRUE :8255  TRUE :8713
##      arr_delay     carrier      flight     tailnum
##  Mode :logical  Mode :logical  Mode :logical  Mode :logical
##  FALSE:327346  FALSE:336776  FALSE:336776  FALSE:334264
##                                TRUE :9430  TRUE :2512
##      origin       dest      air_time     distance
##  Mode :logical  Mode :logical  Mode :logical  Mode :logical
##  FALSE:336776  FALSE:336776  FALSE:327346  FALSE:336776
##                                TRUE :9430
##      hour        minute    time_hour
##  Mode :logical  Mode :logical  Mode :logical
##  FALSE:336776  FALSE:336776  FALSE:336776
##
```

São 8255 voos com `dep_time` faltante, o que pode indicar voos cancelados. As seguintes colunas também possuem dados faltantes: `dep_delay`, `arr_time`, `arr_delay`, `tailnum` e `air_time`.

Exercício 3.3.4

Por que `NA ^ 0` não é um valor faltante? Por que `NA | TRUE` não é um valor faltante? Por que `FALSE & NA` não é um valor faltante? Você consegue descobrir a regra geral? (`NA * 0` é um contraexemplo complicado!)

Solução. `NA ^ 0` resulta em um, pois qualquer número real satisfaz essa mesma condição. A regra geral parece ser que, ao avaliar a expressão, sempre que o valor que `NA` representaria for indiferente para o resultado da expressão, então será retornado um valor diferente de `NA`.

3.4 Ordenar linhas com `arrange()`

Exercício 3.4.1

Como você poderia usar `arrange()` para classificar todos os valores faltantes no começo? (dica: use `is.na()`.)

Solução.

```
arrange(
  flights,
  !is.na(year),
  !is.na(month),
  !is.na(day),
  !is.na(dep_time),
  !is.na(sched_dep_time),
  !is.na(dep_delay),
  !is.na(arr_time),
  !is.na(sched_arr_time),
  !is.na(arr_delay),
  !is.na(carrier),
  !is.na(flight),
  !is.na(tailnum),
  !is.na(origin),
  !is.na(dest),
  !is.na(air_time),
  !is.na(distance),
  !is.na(hour),
  !is.na(minute),
  !is.na(time_hour)
)

## # A tibble: 336,776 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>        <int>     <dbl>    <int>        <int>
## 1 2013     1     2      NA       1545       NA       NA      1910
## 2 2013     1     2      NA       1601       NA       NA      1735
## 3 2013     1     3      NA       857        NA       NA      1209
## 4 2013     1     3      NA       645        NA       NA      952
## 5 2013     1     4      NA       845        NA       NA     1015
## 6 2013     1     4      NA      1830       NA       NA      2044
## 7 2013     1     5      NA       840       NA       NA      1001
```

```
## 8 2013 1 7 NA 820 NA NA 958
## 9 2013 1 8 NA 1645 NA NA 1838
## 10 2013 1 9 NA 755 NA NA 1012
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## # tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## # hour <dbl>, minute <dbl>, time_hour <dttm>
```

Observação. Deve haver uma solução muito mais elegante para este problema.

Exercício 3.4.2

Ordene `flights` para encontrar os voos mais atrasados. Encontre os voos que saíram mais cedo.

Solução. Voos mais atrasados:

```
arrange(flights, desc(dep_delay))
```

```
## # A tibble: 336,776 x 19
##   year month day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>     <int>        <int>      <dbl>    <int>        <int>
## 1 2013     1     9       641         900     1301     1242        1530
## 2 2013     6    15      1432        1935     1137     1607        2120
## 3 2013     1    10      1121        1635     1126     1239        1810
## 4 2013     9    20      1139        1845     1014     1457        2210
## 5 2013     7    22       845        1600     1005     1044        1815
## 6 2013     4    10      1100        1900      960     1342        2211
## 7 2013     3    17      2321        810      911      135         1020
## 8 2013     6    27       959        1900      899     1236        2226
## 9 2013     7    22      2257        759      898      121         1026
## 10 2013    12     5       756        1700      896     1058        2020
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## # tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## # hour <dbl>, minute <dbl>, time_hour <dttm>
```

Voos que saíram mais cedo:

```
arrange(flights, dep_time)
```

```
## # A tibble: 336,776 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>     <int>      <int>     <dbl>     <int>     <int>
## 1 2013     1     13       1        2249      72     108     2357
## 2 2013     1     31       1        2100     181     124     2225
## 3 2013    11     13       1        2359      2     442     440
## 4 2013    12     16       1        2359      2     447     437
## 5 2013    12     20       1        2359      2     430     440
## 6 2013    12     26       1        2359      2     437     440
## 7 2013    12     30       1        2359      2     441     437
## 8 2013     2     11       1        2100     181     111     2225
## 9 2013     2     24       1        2245      76     121     2354
## 10 2013    3      8       1        2355      6     431     440
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Exercício 3.4.3

Ordene `flights` para encontrar os voos mais rápidos.

Solução.

```
arrange(flights, air_time)
```

```
## # A tibble: 336,776 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>     <int>      <int>     <dbl>     <int>     <int>
## 1 2013     1     16     1355      1315      40     1442     1411
## 2 2013     4     13      537       527      10     622      628
## 3 2013    12      6      922       851      31     1021      954
## 4 2013     2      3     2153      2129      24     2247     2224
## 5 2013     2      5     1303      1315     -12     1342     1411
## 6 2013     2     12     2123      2130      -7     2211     2225
## 7 2013     3      2     1450      1500     -10     1547     1608
## 8 2013     3      8     2026      1935      51     2131     2056
## 9 2013     3     18     1456      1329      87     1533     1426
## 10 2013    3     19     2226      2145      41     2305     2246
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Exercício 3.4.4

Quais voos viajaram por mais tempo? Quais viajaram por menos tempo?

Solução. Voos que viajaram por mais tempo:

```
arrange(flights, desc(air_time))
```

```
## # A tibble: 336,776 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>        <int>     <dbl>    <int>        <int>
## 1 2013     3     17    1337        1335      2.00  1937        1836
## 2 2013     2      6     853        900     -7.00  1542        1540
## 3 2013     3     15    1001       1000      1.00  1551        1530
## 4 2013     3     17    1006       1000      6.00  1607        1530
## 5 2013     3     16    1001       1000      1.00  1544        1530
## 6 2013     2      5     900        900      0.00  1555        1540
## 7 2013    11     12     936       930      6.00  1630        1530
## 8 2013     3     14     958       1000     -2.00  1542        1530
## 9 2013    11     20    1006       1000      6.00  1639        1555
## 10 2013    3     15    1342       1335      7.00  1924        1836
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Voos que viajaram por menos tempo:

```
arrange(flights, air_time)
```

```
## # A tibble: 336,776 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>        <int>     <dbl>    <int>        <int>
## 1 2013     1     16    1355       1315     40.0  1442        1411
## 2 2013     4     13     537        527     10.0   622         628
## 3 2013    12      6     922       851     31.0  1021         954
## 4 2013     2      3    2153      2129     24.0  2247        2224
## 5 2013     2      5    1303      1315    -12.0  1342        1411
## 6 2013     2     12    2123      2130     -7.0  2211        2225
## 7 2013     3      2    1450      1500    -10.0  1547        1608
## 8 2013     3      8    2026      1935      51.0  2131        2056
## 9 2013     3     18    1456      1329     87.0  1533        1426
```

```
## 10 2013     3   19    2226          2145      41    2305      2246
## # i 336,766 more rows
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

3.5 Selecionar colunas com `select()`

Exercício 3.5.1

Faça um *brainstorm* da maior quantidade possível de maneiras de selecionar `dep_time`, `dep_delay`, `arr_time` e `air_delay` de `flights`.

Solução. x

Exercício 3.5.2

O que acontece se você incluir o nome de uma variável varias vezes em uma chamada `select()`?

Solução.

```
select(flights, arr_time, arr_time, arr_time)
```

```
## # A tibble: 336,776 x 1
##       arr_time
##   <int>
## 1     830
## 2     850
## 3     923
## 4    1004
## 5     812
## 6     740
## 7     913
## 8     709
## 9     838
## 10    753
## # i 336,766 more rows
```

A variável em questão é selecionada apenas uma vez.

Exercício 3.5.3

O que a função `one_of()` faz? Por que poderia ser útil em conjunção com este vetor?

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

Solução.

```
vars <- c("year", "month", "day", "dep_delay", "arr_delay")
select(flights, one_of(vars)) # superseded in favor of `any_of()``
```

```
## # A tibble: 336,776 x 5
##   year month   day dep_delay arr_delay
##   <int> <int> <int>     <dbl>     <dbl>
## 1 2013    1     1      2       11
## 2 2013    1     1      4       20
## 3 2013    1     1      2       33
## 4 2013    1     1     -1      -18
## 5 2013    1     1     -6      -25
## 6 2013    1     1     -4       12
## 7 2013    1     1     -5       19
## 8 2013    1     1     -3      -14
## 9 2013    1     1     -3       -8
## 10 2013   1     1     -2        8
## # i 336,766 more rows
```

A função `one_of()`, substituída por `any_of()` serve para indicar que devem ser selecionadas todas as colunas cujos nomes estejam no `array`.

Exercício 3.5.4

O resultado ao executar o código a seguir lhe surpreende? Como as funções auxiliares lidam com o caso por padrão? Como você pode mudar esse padrão?

```
select(flights, contains("TIME"))
```

Solução.

```
select(flights, contains("TIME"))
```

```
## # A tibble: 336,776 x 6
##   dep_time sched_dep_time arr_time sched_arr_time air_time time_hour
##   <int>      <int>     <int>      <int>     <dbl> <dttm>
## 1     517        515     830       819    227 2013-01-01 05:00:00
## 2     533        529     850       830    227 2013-01-01 05:00:00
## 3     542        540     923       850    160 2013-01-01 05:00:00
## 4     544        545    1004      1022    183 2013-01-01 05:00:00
## 5     554        600     812       837    116 2013-01-01 06:00:00
## 6     554        558     740       728    150 2013-01-01 05:00:00
## 7     555        600     913       854    158 2013-01-01 06:00:00
## 8     557        600     709       723     53 2013-01-01 06:00:00
## 9     557        600     838       846    140 2013-01-01 06:00:00
## 10    558        600     753       745    138 2013-01-01 06:00:00
## # ... i 336,766 more rows
```

O caso não surpreende. São retornadas todas as colunas que possuem “TIME” em seus nomes, não diferenciando maiúsculas e minúsculas. O comportamento pode ser alterado da seguinte forma:

```
select(flights, contains("TIME", ignore.case = FALSE))
```

```
## # A tibble: 336,776 x 0
```

3.6 Adicionar novas variáveis com `mutate()`

Exercício 3.6.1

Atualmente, `dep_time` e `sched_dep_time` são convenientes para observar, mas difíceis de usar para calcular, porque não são realmente números contínuos. Converta-os para uma representação mais adequada do número de minutos desde a meia-noite.

Solução.

```
(flights_min <- mutate(
  flights,
  dep_time_minutes = 60 * (dep_time %/% 100) + (dep_time %% 100),
  sched_dep_time_minutes = 60 * (sched_dep_time %/% 100) + (sched_dep_time %% 100),
  arr_time_minutes = 60 * (arr_time %/% 100) + (arr_time %% 100)
))
```

```
## # A tibble: 336,776 x 22
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>      <int>     <dbl>    <int>      <int>
## 1 2013     1     1      517        515       2     830       819
## 2 2013     1     1      533        529       4     850       830
## 3 2013     1     1      542        540       2     923       850
## 4 2013     1     1      544        545      -1    1004      1022
## 5 2013     1     1      554        600      -6     812       837
## 6 2013     1     1      554        558      -4     740       728
## 7 2013     1     1      555        600      -5     913       854
## 8 2013     1     1      557        600      -3     709       723
## 9 2013     1     1      557        600      -3     838       846
## 10 2013    1     1      558        600      -2     753       745
## # i 336,766 more rows
## # i 14 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>, dep_time_minutes <dbl>,
## #   sched_dep_time_minutes <dbl>, arr_time_minutes <dbl>
```

Exercício 3.6.2

Compare `air_time` e `arr_time - dep_time`. O que você espera ver? O que você vê? O que você precisa fazer para corrigir isso?

Solução.

```
transmute(flights_min, air_time, arr_time_minutes - dep_time_minutes)
```

```
## # A tibble: 336,776 x 2
##   air_time `arr_time_minutes - dep_time_minutes`
##   <dbl>                <dbl>
## 1     227                  193
## 2     227                  197
## 3     160                  221
## 4     183                  260
## 5     116                  138
## 6     150                  106
## 7     158                  198
## 8      53                   72
## 9     140                  161
## 10    138                  115
## # i 336,766 more rows
```

Como os valores `arr_time` e `dep_time` não são números de fato, a diferença não faz sentido e assim o cálculo gera uma diferença muito grande. Para corrigir isso, primeiro teremos que converter os valores dessas duas variáveis para o número de minutos desde a meia noite e, depois, efetuar a diferença. Ainda assim, pode haver divergência entre esse valor e `air_time`, que pode ser explicada por chegada antecipada, saída atrasada ou porque um vôo chegou ao seu destino após a meia-noite.

Exercício 3.6.3

Compare `dep_time`, `sched_dep_time` e `dep_delay`. Como você espera que esses números estejam relacionados?

Solução.

```
select(flights_min, "dep_time", "sched_dep_time", dep_delay)
```

```
## # A tibble: 336,776 x 3
##   dep_time sched_dep_time dep_delay
##   <int>        <int>     <dbl>
## 1     517          515      2
## 2     533          529      4
## 3     542          540      2
## 4     544          545     -1
## 5     554          600     -6
## 6     554          558     -4
## 7     555          600     -5
## 8     557          600     -3
## 9     557          600     -3
## 10    558          600     -2
## # i 336,766 more rows
```

É esperado que `dep_time = sched_dep_time + dep_delay`.

Exercício 3.6.4

Encontre os 10 voos mais atrasados usando uma função de classificação. Como você quer lidar com empates? Leia cuidadosamente a documentação de `min_rank()`.

Solução.

```
filter(
  flights,
  between(rank(desc(flights$dep_delay), ties.method = "min"), 1, 10)
)

## # A tibble: 10 x 19
##   year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>        <int>     <dbl>    <int>        <int>
## 1 2013     1     9      641         900     1301     1242        1530
## 2 2013     1    10     1121        1635     1126     1239        1810
## 3 2013    12     5      756        1700      896     1058        2020
## 4 2013     3    17     2321        810      911      135         1020
## 5 2013     4    10     1100        1900      960     1342        2211
## 6 2013     6    15     1432        1935     1137     1607        2120
## 7 2013     6    27      959        1900      899     1236        2226
## 8 2013     7    22      845        1600     1005     1044        1815
## 9 2013     7    22     2257        759      898      121         1026
## 10 2013    9    20     1139        1845     1014     1457        2210
## # i 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Usei a função `rank` e os empates foram tratados com o parâmetro `ties.method` setado como `min`.

Exercício 3.6.5

O que `1:3 + 1:10` retorna? Por quê?

Solução.

```
1:3 + 1:10
```

```
## Warning in 1:3 + 1:10: comprimento do objeto maior não é múltiplo do
## comprimento do objeto menor
```

```
## [1] 2 4 6 5 7 9 8 10 12 11
```

Como os vetores têm tamanhos diferentes, a soma vai ser executada entre as posições e, quando o menor dos vetores tiver sido completamente consumido, será tomado novamente o primeiro elemento (como em um movimento circular).

Exercício 3.6.6

Quais funções trigonométricas o R fornece?

Solução. Utilizamos o comando `?cos` para chegar até a documentação do pacote `trig`, um dos componentes da base do R.

O R fornece as funções `cos(x)`, `sin(x)`, `tan(x)`, `acos(x)`, `asin(x)`, `atan(x)`, `atan2(y, x)` (arco tangente entre dois vetores), `cosp(x)`, `sinp(x)` e `tanp(x)`.

3.7 Resumos agrupados com `summarize()`

Exercício 3.7.1

Faça um *brainstorming* de pelo menos cinco maneiras diferentes de avaliar as características do atraso típico de um grupo de voos. Considere os seguintes cenários:

- Um voo está 15 minutos adiantado em 50% do tempo e 15 minutos atrasado em 50% do tempo.
- Um voo está sempre 10 min atrasado.
- Um voo está 30 minutos adiantado em 50% do tempo e 30 minutos atrasado em 50% do tempo.
- Em 99% do tempo um voo está no horário. Em 1% do tempo, está 2 horas atrasado.

O que é mais importante: atrasado na chegada ou atraso na partida?

Solução. x

Exercício 3.7.2

Crie outra abordagem que lhe dará o mesmo resultado que `not_cancelled %>% count(dest) & not_cancelled %>% count(tailnum, wt = distance)` (sem usar `count()`).

Solução.

```
not_cancelled %>%
  group_by(dest) %>%
  summarise(n = n())
```

```
## # A tibble: 104 x 2
```

```
##   dest      n
##   <chr> <int>
## 1 ABQ     254
## 2 ACK     264
## 3 ALB     418
## 4 ANC      8
## 5 ATL    16837
## 6 AUS    2411
## 7 AVL     261
## 8 BDL     412
## 9 BGR     358
## 10 BHM    269
## # i 94 more rows
```

```
not_cancelled %>%
  group_by(tailnum) %>%
  summarise(n = sum(distance))
```

```
## # A tibble: 4,037 x 2
##   tailnum      n
##   <chr>     <dbl>
## 1 D942DN    3418
## 2 N0EGMQ  239143
## 3 N10156   109664
## 4 N102UW    25722
## 5 N103US    24619
## 6 N104UW    24616
## 7 N10575   139903
## 8 N105UW    23618
## 9 N107US    21677
## 10 N108UW   32070
## # i 4,027 more rows
```

Exercício 3.7.3

Nossa definição de voos cancelados (`is.na(dep_delay) | is.na(arr_delay)`) é ligeiramente insuficiente. Por quê? Qual é a coluna mais importante?

Solução. As variáveis `dep_delay` e `arr_delay` se referem ao atraso na partida ou na chegada dos voos. Caso um voo tenha saído e chegado no horário exato, esses valores podem estar `NA`, ou seja, o voo não foi cancelado, apenas partiu e chegou no horário planejado. Nesse caso, o mais correto seria considerar como cancelados os voos `dep_time` é `NA`.

Exercício 3.7.4

Veja o número de voos cancelados por dia. Existe um padrão? A proporção de voos cancelados está relacionado ao atraso médio?

Solução.

```
cancelled_by_day <- flights %>%
  group_by(year, month, day) %>%
  summarise(
    date = as.Date(paste(year, month, day, sep='-')),
    count = n(),
    count_cancelled = sum(is.na(dep_time)),
    count_not_cancelled = sum(!is.na(dep_time)),
    mean_dep_delay = mean(dep_delay, na.rm = TRUE),
    mean_arr_delay = mean(arr_delay, na.rm = TRUE),
  )
## Warning: Returning more (or less) than 1 row per `summarise()`' group was deprecated in
## dplyr 1.1.0.
## i Please use `reframe()` instead.
## i When switching from `summarise()` to `reframe()`, remember that `reframe()`'
##   always returns an ungrouped data frame and adjust accordingly.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

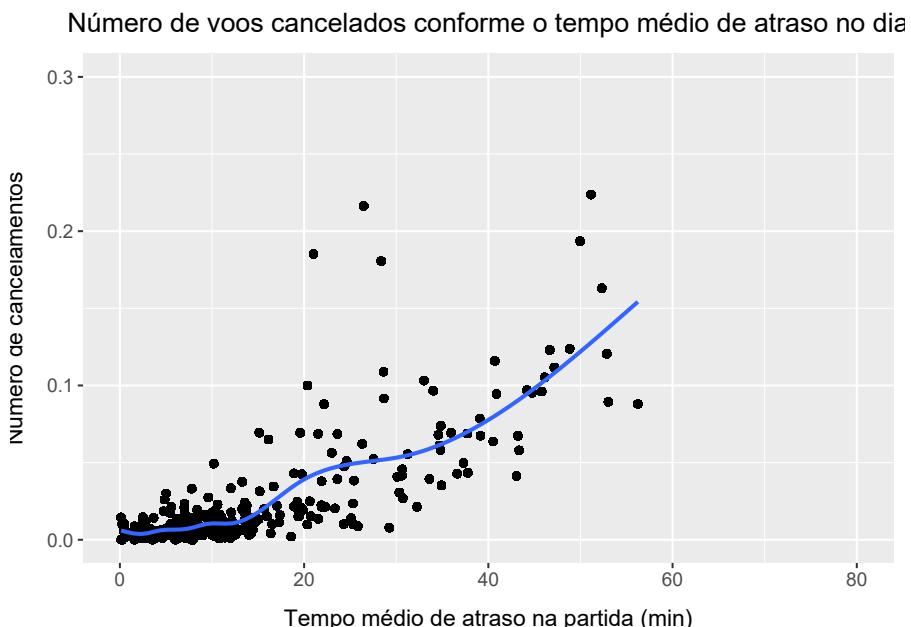
## `summarise()` has grouped output by 'year', 'month', 'day'. You can override
## using the `.`groups` argument.
```

```
cancelled_by_day %>%
  ggplot(aes(mean_dep_delay, count_cancelled / count)) +
  geom_point() +
  geom_smooth(se = FALSE) +
  labs(
    title = "Número de voos cancelados conforme o tempo médio de atraso no dia",
    x = "Tempo médio de atraso na partida (min)",
    y = "Número de cancelamentos"
  ) +
  xlim(0, 80) +
  ylim(0, 0.3) +
  tema
```

```
## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'

## Warning: Removed 12409 rows containing non-finite values (`stat_smooth()`).

## Warning: Removed 12409 rows containing missing values (`geom_point()`).
```



Parece existir uma relação entre o número de voos cancelados no dia e a média de atraso nos voos desse mesmo dia. Caso haja alguma condição desfavorável (tempo ruim, problemas na pista de decolagem/pouso, etc), o intervalo entre uma decolagem/pouso e outro pode aumentar significativamente gerando atrasos que se acumulam a ponto de alguns voos terem que ser cancelados (esse comportamento é real?).

Exercício 3.7.5

Qual companhia tem os piores atrasos? Desafio: você consegue desembaralhar o efeito dos aeroportos ruins *versus* companhias ruins? Por quê/Por que não? (Dica: pense em `flights %>% group_by(cartier, dest) %>% summarize(n())`)

Solução. Para verificar qual companhia tem os piores atrasos, vamos calcular o atraso médio por companhia.

```

flights %>%
  group_by(carrier) %>%
  summarize(
    mean_delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  arrange(desc(mean_delay))

## # A tibble: 16 x 2
##   carrier mean_delay
##   <chr>     <dbl>
## 1 F9         21.9
## 2 FL         20.1
## 3 EV         15.8
## 4 YV         15.6
## 5 OO         11.9
## 6 MQ          10.8
## 7 WN          9.65
## 8 B6          9.46
## 9 9E          7.38
## 10 UA          3.56
## 11 US          2.13
## 12 VX          1.76
## 13 DL          1.64
## 14 AA          0.364
## 15 HA         -6.92
## 16 AS         -9.93

```

Podemos notar que a companhia com o maior atraso médio é a F9 (Frontier Airlines Inc).

Para tentar desembaralhar o efeito de aeroportos ruins e companhias ruins, vamos:

- filtrar apenas os voos com atraso;
- agrupar os voos conforme as rotas e companhias;
- calcular o atraso médio e o total de voos por companhia no trecho (`arr_delay` e `flights`);
- calcular o atraso médio e o total de voos do trecho de todas as companhias (`arr_delay_total` e `flights_total`);
- calcular o atraso médio por voo da companhia (`arr_delay_mean <- arr_delay / flights`);
- calcular o atraso “médio” das demais companhias (`arr_delay_others <- (arr_delay_total - arr_delay) / (flights_total - flights)`);
- calcular a diferença entre o atraso médio da companhia e o atraso médio das outras companhias juntas (`arr_delay_diff <- arr_delay_mean - arr_delay_others`);

- remover valores cuja diferença não faça sentido (`is.finite(arr_delay_diff)`);
- agrupar por companhia;
- calcular a média das diferenças de atraso da companhia (`arr_delay_diff`);

```
(atrasos <- flights %>%
  filter(!is.na(arr_delay)) %>%
  group_by(origin, dest, carrier) %>%
  summarise(
    arr_delay = sum(arr_delay),
    flights = n()
  ) %>%
  group_by(origin, dest) %>%
  mutate(
    arr_delay_total = sum(arr_delay),
    flights_total = sum(flights)
  ) %>%
  ungroup() %>%
  mutate(
    arr_delay_mean = arr_delay / flights, # atraso médio da companhia
    arr_delay_others = (arr_delay_total - arr_delay) / (flights_total - flights), # atraso médio das demais
    arr_delay_diff = arr_delay_mean - arr_delay_others # diferença do atraso em relação às demais
  ) %>%
  filter(is.finite(arr_delay_diff)) %>%
  group_by(carrier) %>%
  summarise(
    arr_delay_diff = mean(arr_delay_diff)
  ) %>%
  arrange(desc(arr_delay_diff)))
```

```
## `summarise()` has grouped output by 'origin', 'dest'. You can override using
## the `groups` argument.
```

```
## # A tibble: 15 x 2
##   carrier arr_delay_diff
##   <chr>      <dbl>
## 1 00        27.3
## 2 F9        17.3
## 3 EV        11.0
## 4 B6        6.41
## 5 FL        2.57
## 6 VX       -0.202
## 7 AA       -0.970
```

```
## 8 WN          -1.27
## 9 UA          -1.86
## 10 MQ         -2.48
## 11 YV         -2.81
## 12 9E         -3.54
## 13 US         -4.14
## 14 DL        -10.2
## 15 AS        -15.8
```

Desconsiderando o efeito de trechos e aeroportos ruins, a companhia com maior atraso é a OO (SkyWest Airlines Inc.).

```
atrasos %>%
  left_join(airlines, by = "carrier") %>%
  ggplot(aes(
    arr_delay_diff,
    reorder(name, desc(arr_delay_diff)))
  )) +
  geom_col() +
  labs(
    title = "Atrasos por companhia aérea",
    y = "Companhia aérea",
    x = "Tempo médio de atraso (em min.)"
  ) +
  tema
```



Exercício 3.7.6

Para cada avião, conte o número de voos antes do primeiro atraso de mais de uma hora.

Solução. Utilizando a variável `flight` para identificar o voo e a variável `arr_delay` como parâmetro para determinar o tempo de atraso:

- ordenamos o data-frame conforme a hora agendada para decolagem;
- agrupamos pelo número do voo;
- utilizamos as funções `first()` e `which()` para buscar a posição do primeiro elemento que é `NA` ou o atraso é maior do que 60 min.

Obs.: `NA` indica que aquele voo não teve nenhum atraso superior a 60 min.

```
flights %>%
  arrange(time_hour) %>%
  group_by(flight) %>%
  summarise(
    first_delay_pos = first(which(is.na(arr_delay) | arr_delay > 60)) - 1
  )
```

```
## # A tibble: 3,844 x 2
##   flight first_delay_pos
##   <int>          <dbl>
## 1     1            47
## 2     2            NA
## 3     3             9
## 4     4            77
## 5     5            11
## 6     6            23
## 7     7            17
## 8     8            15
## 9     9            12
## 10    10           24
## # i 3,834 more rows
```

Exercício 3.7.7

O que o argumento `sort` para `count()` faz? Quando você pode usá-lo?

Solução. Utilizando o comando `?count`, identificamos que o argumento `sort` organiza a contagem em ordem decrescente.

3.8 Mudanças agrupadas (e filtros)

Exercício 3.8.1

Volte à tabela de funções de mudança e filtragem úteis. Descreva como cada operação muda quando você as combina com o agrupamento.

Solução. x

Exercício 3.8.2

Qual avião (`tailnum`) tem o pior registro de pontualidade?

Solução. Vamos inicialmente considerar que um voo é pontual se o tempo de atraso na chegada (`arr_delay`) é igual ou inferior a zero e, para considerar um avião como mais ou menos pontual, levaremos em consideração a proporção de voos pontuais que ele realizou.

```

flights %>%
  # Considerar apenas os registros que tem a informação sobre o voo,
  # hora de chegada e atraso na chegada
  filter(!is.na(tailnum), !is.na(arr_time), !is.na(arr_delay)) %>%
  # Criar uma variável booleana (0 ou 1) que indica se o voo foi pontual
  mutate(
    on_time = !is.na(arr_time) & arr_delay <= 0
  ) %>%
  # Calcular a proporção de voos pontuais e o número de voos por voo
  group_by(tailnum) %>%
  summarise(
    n = n(),
    arr_delay = mean(arr_delay),
    on_time = mean(on_time)
  ) %>%
  # Descartar aviões que voaram 20 vezes ou menos
  filter(n > 20) %>%
  # Ordenar por percentual de voos pontuais
  arrange(desc(on_time)) %>%
  head()

```

```

## # A tibble: 6 x 4
##   tailnum      n arr_delay on_time
##   <chr>     <int>     <dbl>    <dbl>
## 1 N382HA     26     -23.5    0.885
## 2 N553AA     51     -6.33    0.863
## 3 N423AS     29     -22.3    0.862
## 4 N538AA     35     -9.6     0.857
## 5 N548AA     49     -15.5    0.857
## 6 N5EJAA     21     -12.5    0.857

```

Com base na configuração acima, o avião N382HA é o mais pontual, com 88,46% dos 26 voos sendo executados com pontualidade.

Exercício 3.8.3

A que horas você deverá voar se quiser evitar atrasos ao máximo.

Solução. O problema depende de encontrar o horário em que ocorrem menos atrasos. Consideraremos a hora inteira como parâmetro para a busca (`hour`) e utilizaremos a média dos tempos de atraso dos voos.

```

flights %>%
  filter(!is.na(hour)) %>%
  group_by(hour) %>%
  summarise(
    arr_delay = mean(arr_delay, na.rm = T)
  ) %>%
  arrange(arr_delay) %>%
  head()

```

```

## # A tibble: 6 x 2
##   hour arr_delay
##   <dbl>     <dbl>
## 1     7     -5.30
## 2     5     -4.80
## 3     6     -3.38
## 4     9     -1.45
## 5     8     -1.11
## 6    10      0.954

```

Exercício 3.8.4

Para cada destino, calcule os minutos totais de atraso. Para cada voo, calcule a proporção de atraso total par seu destino.

Solução. R.: Para calcular o atraso total (em minutos) por destino, somaremos os valores da variável `arr_delay` de todos os voos para cada destino (`group_by(dest)`). Em seguida, para calcular a proporção com a qual cada voo colabora para o atraso total do destino, utilizaremos a razão entre o atraso do voo e o total do grupo ao qual pertence.

```

flights %>%
  filter(arr_delay > 0) %>%
  group_by(dest) %>%
  mutate(
    arr_delay_total = sum(arr_delay),
    arr_delay_prop = arr_delay / arr_delay_total
  ) %>%
  select (dest, flight, dep_time, arr_delay, arr_delay_total, arr_delay_prop) %>%
  arrange(dest, desc(arr_delay_prop)) %>%
  head()

```

```

## # A tibble: 6 x 6
##   dest flight dep_time arr_delay arr_delay_total arr_delay_prop
##   <dbl>   <dbl>     <dbl>     <dbl>           <dbl>           <dbl>
## 1     1       1       10.0      1.00            1.00            1.00
## 2     1       2       10.0      1.00            1.00            1.00
## 3     1       3       10.0      1.00            1.00            1.00
## 4     1       4       10.0      1.00            1.00            1.00
## 5     1       5       10.0      1.00            1.00            1.00
## 6     1       6       10.0      1.00            1.00            1.00

```

```
## # Groups:  dest [1]
##   dest flight dep_time arr_delay arr_delay_total arr_delay_prop
##   <chr> <int>    <int>     <dbl>          <dbl>        <dbl>
## 1 ABQ    1505    2145      153       4487  0.0341
## 2 ABQ     65     2223      149       4487  0.0332
## 3 ABQ     65     2146      138       4487  0.0308
## 4 ABQ    1505    2206      137       4487  0.0305
## 5 ABQ     65     2220      136       4487  0.0303
## 6 ABQ    1505    2025      126       4487  0.0281
```

Exercício 3.8.5

Atrasos são normalmente temporariamente correlacionados: mesmo quando o problema que causou o atraso inicial foi resolvido, , voos posteriores atrasam para permitir que os voos anteriores decolem. Usando `lag()`, explore como o atraso de um voo está relacionado com o atraso imediatamente anterior.

Solução. Considerando o atraso na decolagem, vamos inicialmente ordenar os voos por aeroporto, data e hora da decolagem. Em seguida, agrupando pelo aeroporto, coletamos o atraso do voo anterior (note que o `mutate` irá atuar sobre o grupo apenas). Não faz sentido considerar o voo anterior em outro aeroporto!. Na sequência, podemos agrupar pelo tempo de atraso do voo anterior para calcular a média dos atrasos dos voos. Por fim, é exibido o gráfico.

Avaliando a imagem, podemos notar a tendência de que, quanto maior o atraso do voo imediatamente anterior, maior será o atraso do voo atual. O padrão crescente segue até atrasos de aproximadamente 435 minutos. Depois passa a decrescer, o que deve ser analisado mais aprofundadamente.

```
flights %>%
  arrange(origin, month, day, dep_time) %>%
  group_by(origin) %>%
  mutate(prev_dep_delay = lag(dep_delay)) %>%
  filter(!is.na(dep_delay), !is.na(prev_dep_delay)) %>%
  group_by(origin, prev_dep_delay) %>%
  summarise(dep_delay_mean = mean(dep_delay)) %>%
  ggplot(aes(prev_dep_delay, dep_delay_mean)) +
  geom_point() +
  geom_smooth(se = FALSE) +
  scale_x_continuous(breaks = seq(0, 1300, by = 60)) +
  scale_y_continuous(breaks = seq(0, 450, by = 60)) +
  labs(
    title = "Atraso médio na decolagem em função do atraso na decolagem anterior.",
    x = "Atraso na decolagem anterior (min.)",
```

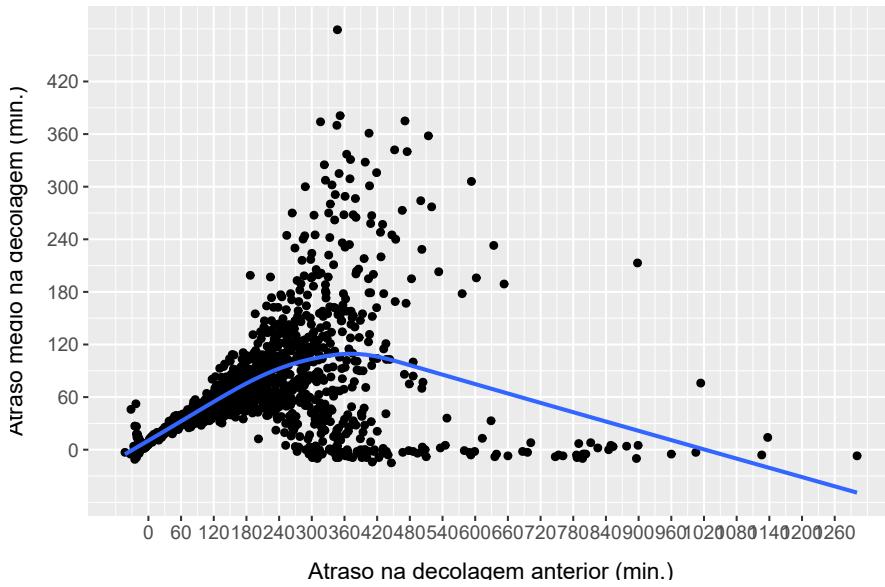
```

y = "Atraso médio na decolagem (min.)"
) +
  tema
}

## `summarise()` has grouped output by 'origin'. You can override using the
## `.groups` argument.
## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'

```

Atraso médio na decolagem em função do atraso na decolagem anterior



É importante notar que o padrão se repete se avaliarmos cada aeroporto individualmente.

```

flights %>%
  arrange(origin, month, day, dep_time) %>%
  group_by(origin) %>%
  mutate(prev_dep_delay = lag(dep_delay)) %>%
  filter(!is.na(dep_delay), !is.na(prev_dep_delay)) %>%
  group_by(origin, prev_dep_delay) %>%
  summarise(dep_delay_mean = mean(dep_delay)) %>%
  ggplot(aes(prev_dep_delay, dep_delay_mean)) +
  geom_point() +
  geom_smooth(se = FALSE) +
  facet_wrap(~ origin, ncol = 1) +

```

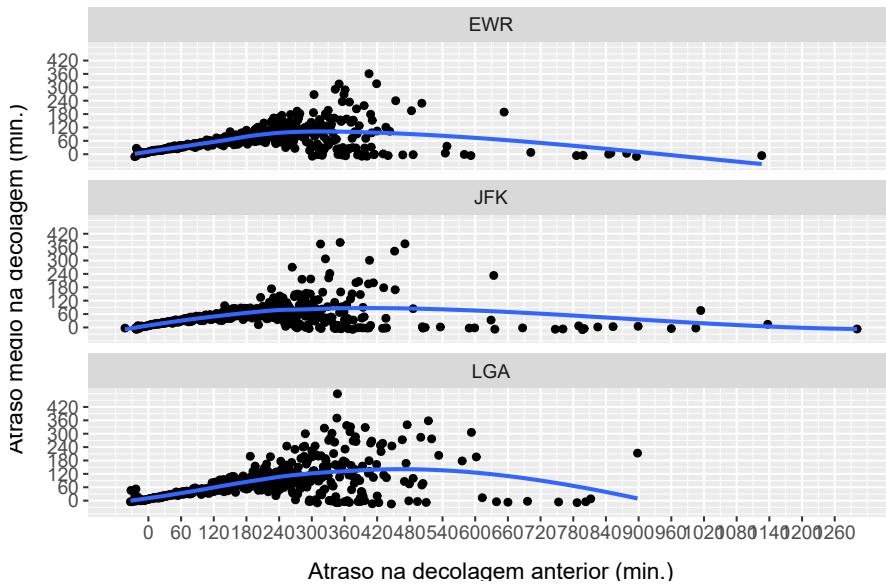
```

  scale_x_continuous(breaks = seq(0, 1300, by = 60)) +
  scale_y_continuous(breaks = seq(0, 450, by = 60)) +
  labs(
    title = "Atraso médio na decolagem em função do atraso na decolagem anterior.",
    x = "Atraso na decolagem anterior (min.)",
    y = "Atraso médio na decolagem (min.)"
  ) +
  tema
}

## `summarise()` has grouped output by 'origin'. You can override using the
## `.` argument.
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'

```

Atraso médio na decolagem em função do atraso na decolagem anterio



Exercício 3.8.6

Veja cada destino. Você consegue encontrar os voos que são suspeitamente rápidos? (Ou seja, voos que representam um erro de entrada de dados em potencial). Calcule o tempo de viagem de um voo relativo ao voo mais curto para aquele destino. Quais voos ficaram mais atrasados no ar?

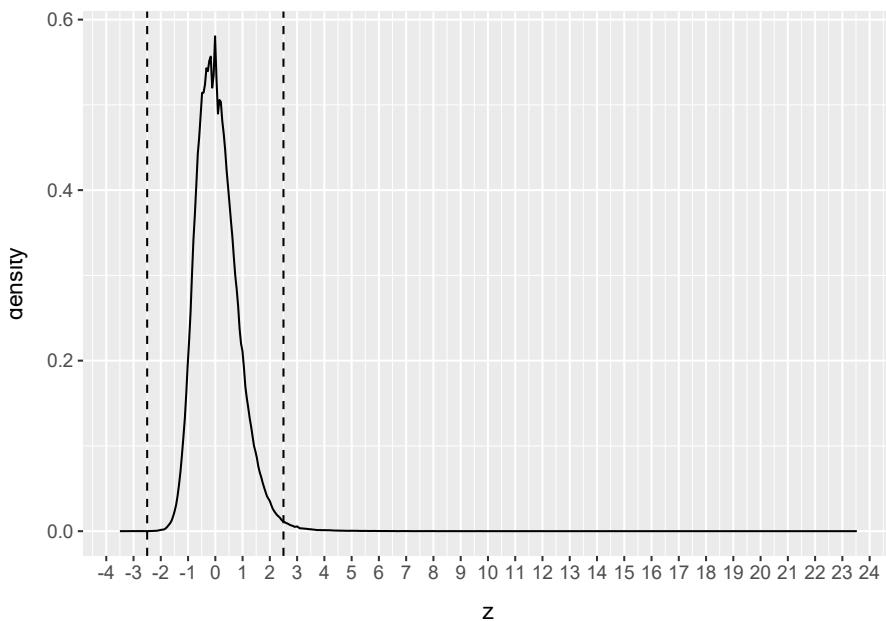
Solução. Inicialmente calcularemos a média e o desvio padrão para cada rota (`origin`, `dest`) e, na sequência, calcularemos o *z-score* para avaliar a distribuição dos tempos

de voo. Usaremos a mediana e o intervalo interquartílico para escapar do efeito de outliers.

```
standardized <- flights %>%
  filter(!is.na(air_time)) %>%
  group_by(origin, dest) %>%
  mutate(
    median = median(air_time),
    iqr = IQR(air_time),
    n = n(),
    z = (air_time - median) / iqr
  ) %>%
  ungroup()

standardized %>%
  ggplot(aes(x = z)) +
  geom_density() +
  geom_vline(aes(xintercept = -2.5), linetype = "dashed") +
  geom_vline(aes(xintercept = 2.5), linetype = "dashed") +
  scale_x_continuous(breaks = seq(-10, 30, by = 1)) +
  tema
```

Warning: Removed 4 rows containing non-finite values (`stat_density()`).



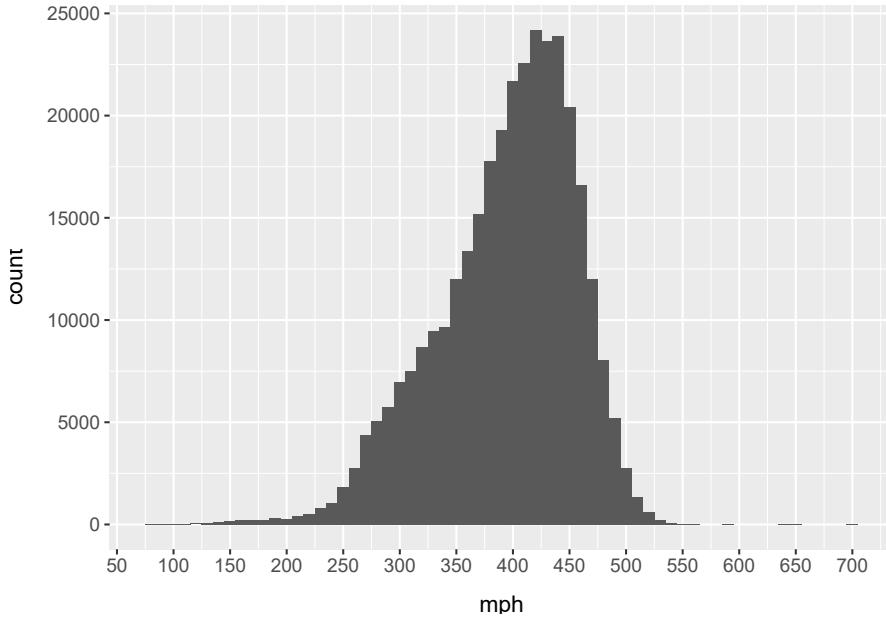
Os voos com z-score muito baixo, são aqueles cujo tempo de voo foi muito menor do que a média, ou seja, os mais rápidos.

```
standardized %>%
  arrange(z) %>%
  select(carrier, flight, origin, dest, month, day, air_time, median, iqr, z) %>%
  head(10)

## # A tibble: 10 x 10
##   carrier flight origin dest  month   day air_time median   iqr     z
##   <chr>    <int> <chr>  <chr> <int> <dbl>    <dbl> <dbl> <dbl>
## 1 EV        4667 EWR    MSP      7     2     93     149     16 -3.5
## 2 DL        1499 LGA    ATL      5    25     65     112     14 -3.36
## 3 US        2132 LGA    BOS      3     2     21      37      5 -3.2
## 4 B6         30 JFK    ROC      3    25     35      51      5 -3.2
## 5 B6         2002 JFK   BUF     11    10     38      57      6 -3.17
## 6 EV        4292 EWR    GSP      5    13     55      92     12 -3.08
## 7 EV        4249 EWR    SYR      3    15     30      39      3 -3
## 8 EV        4580 EWR    BTV      6    29     34      46      4 -3
## 9 EV        3830 EWR    RIC      7     2     35      53      6 -3
## 10 EV       4687 EWR    CVG      9    29     62      95     11 -3
```

Adicionalmente, vamos considerar também a velocidade do voo (`mph <- distance / (air_time / 60)`).

```
standardized %>%
  mutate(
    mph = distance / (air_time / 60)
  ) %>%
  ggplot(aes(x = mph)) +
  geom_histogram(binwidth = 10) +
  scale_x_continuous(breaks = seq(0, 700, by = 50)) +
  tema
```



```
standardized %>%
  mutate(
    mph = distance / (air_time / 60)
  ) %>%
  arrange(desc(mph)) %>%
  select(carrier, flight, origin, dest, month, day, mph) %>%
  head(10)
```

```
## # A tibble: 10 x 7
##   carrier flight origin dest  month   day   mph
##   <chr>    <int> <chr>  <chr> <dbl> <dbl> <dbl>
## 1 DL        1499 LGA    ATL      5     25  703.
## 2 EV        4667 EWR    MSP      7     2   650.
## 3 EV        4292 EWR    GSP      5    13   648
## 4 EV        3805 EWR    BNA      3    23   641.
## 5 DL        1902 LGA    PBI      1    12   591.
## 6 DL         315  JFK    SJU     11    17   564
## 7 B6         707  JFK    SJU      2    21   557.
## 8 AA         936  JFK    STT     11    17   556.
## 9 DL        347  JFK    SJU     11    16   554.
## 10 B6       1503  JFK    SJU     11    16   554.
```

Algum conhecimento prévio nos indica que a velocidade superior a 550 milhas por hora são suspeitamente altas.

Note que, em ambas as análises, coicidiram quase todos os voos. Poderíamos fazer análises mais acuradas, se tivéssemos mais conhecimento sobre o domínio de negócio, contudo já podemos concluir que aqueles são os voos suspeitos.

Exercício 3.8.7

Encontre todos os destinos que são feitos por pelo menos duas companhias. Use essa informação para classificar as companhias.

Solução.

```

flights %>%
  filter(!is.na(arr_delay)) %>%
  group_by(origin, dest) %>%
  mutate(
    carrier_count = n_distinct(carrier),
    arr_delay_mean = mean(arr_delay),
    arr_delay_percent = arr_delay / arr_delay_mean
  ) %>%
  filter(carrier_count > 1) %>%
  group_by(origin, dest, carrier) %>%
  summarise(
    arr_delay = mean(arr_delay_percent)
  ) %>%
  arrange(origin, dest, desc(arr_delay)) %>%
  head(25)

## `summarise()` has grouped output by 'origin', 'dest'. You can override using
## the ` `.groups` argument.

## # A tibble: 25 x 4
##   origin dest   carrier arr_delay
##   <chr>  <chr> <chr>      <dbl>
## 1 EWR    ATL    EV        1.48 
## 2 EWR    ATL    UA        0.793
## 3 EWR    ATL    DL        0.755
## 4 EWR    ATL    9E       -0.472
## 5 EWR    AUS    WN        23.7 
## 6 EWR    AUS    UA        -9.02
## 7 EWR    BDL    UA        3.20 
## 8 EWR    BDL    EV        0.962
## 9 EWR    BNA    EV        1.39 

```

```
## 10 EWR     BNA      WN      -0.168
## # i 15 more rows
```

4

Fluxo de trabalho: scripts

4.1 Executando códigos

Não temos exercícios neste seção.

4.2 Diagnósticos Rstudio

Exercício 4.2.1

Vá para a conta RStudio Tips no Twitter, em ?, e escolha uma dica que pareça interessante. Pratique o uso dessa dica!

Solução. x

Exercício 4.2.2

Quais outros erros comuns o diagnóstico do RStudio reportará? Leia <http://bit.ly/RStudiocodediag> para descobrir.

Solução. x



5

Análise exploratória de dados

5.1 Introdução

Não temos exercícios nesta seção.

5.2 Perguntas

Não temos exercícios nesta seção.

5.3 Variação

Exercício 5.3.1

Explore a distribuição de cada variável x , y e z em `diamonds`. O que você aprende? Pense em um diamante e como você pode determinar qual dimensão é o comprimento, a largura e a profundidade.

Solução. Por se tratar de variáveis continuas, vamos utilizar um gráfico de densidade (ou histograma) para visualizar os dados.

Como x e y possuem distribuição mais parecida, acredita-se que tratam-se do comprimento e da largura do diamante, sendo z a profundidade (por ter média menor).

```
plot <- diamonds %>%
  ggplot() +
  coord_cartesian(
    xlim = c(0, 10),
```

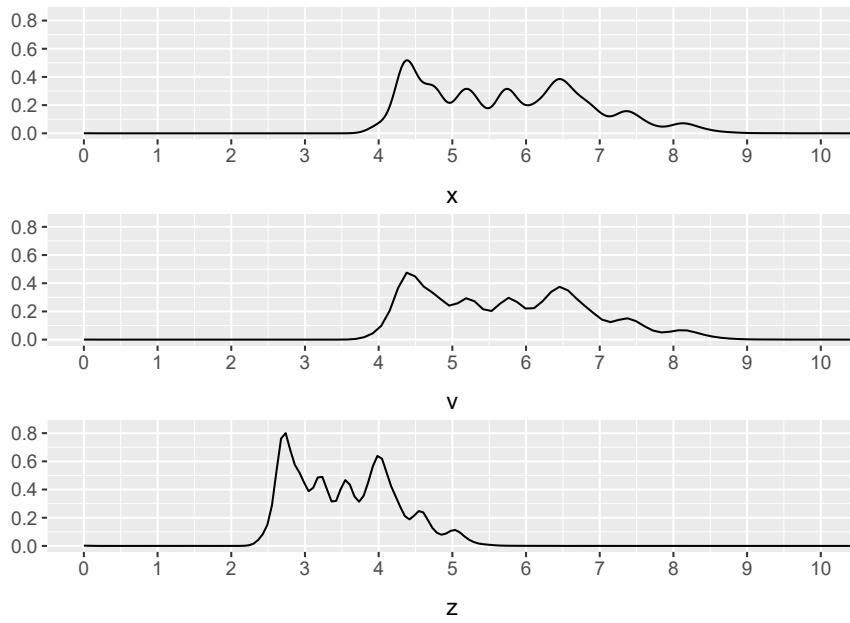
```

        ylim = c(0, .85)
    ) +
  scale_x_continuous(breaks = seq(0, 10, by = 1)) +
  labs(
    y = ""
  ) +
  tema

x <- plot + geom_density(aes(x))
y <- plot + geom_density(aes(y))
z <- plot + geom_density(aes(z))

grid.arrange(x, y, z, nrow = 3)

```

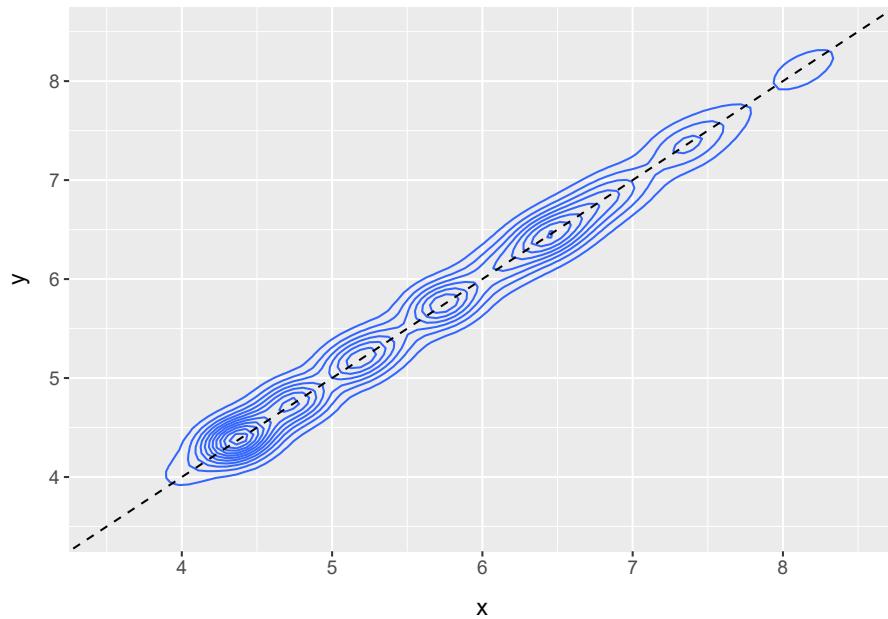


```

diamonds %>%
  # Remover os outliers
  filter(0 < x, x <= 10, 0 < y, y <= 10) %>%
  ggplot(aes(x, y)) +
    # Mostrar a densidade de x e y em conjunto
    geom_density2d() +
    # Mostrar uma linha guia para visualizar se x e Y crescem de forma
    # proporcional, isto é, se os diamantes são quadrados/redondos

```

```
geom_abline(  
  aes(intercept = 0, slope = 1),  
  linetype = "dashed"  
) +  
# Arrumar a proporção do gráfico  
coord_cartesian(  
  xlim = c(3.5, 8.5),  
  ylim = c(3.5, 8.5)  
) +  
# Aplicar o tema padrão  
tema
```



Exercício 5.3.2

Explore a distribuição de `price`. Você identifica algo incomum ou surpreendente? (Dica: pense cuidadosamente sobre `binwidth` e certifique-se de experimentar uma ampla gama de valores).

Solução.

```
summary(diamonds)
```

```
##      carat           cut       color     clarity      depth
##  Min.   :0.2000   Fair    : 1610   D: 6775   SI1    :13065   Min.   :43.00
##  1st Qu.:0.4000  Good   : 4906   E: 9797   VS2    :12258   1st Qu.:61.00
##  Median :0.7000  Very Good:12082  F: 9542   SI2    : 9194   Median :61.80
##  Mean   :0.7979  Premium :13791   G:11292   VS1    : 8171   Mean   :61.75
##  3rd Qu.:1.0400  Ideal   :21551   H: 8304   VVS2   : 5066   3rd Qu.:62.50
##  Max.   :5.0100                    I: 5422   VVS1   : 3655   Max.   :79.00
##                               J: 2808   (Other): 2531
##      table          price         x         y
##  Min.   :43.00   Min.   : 326   Min.   : 0.000   Min.   : 0.000
##  1st Qu.:56.00  1st Qu.: 950   1st Qu.: 4.710   1st Qu.: 4.720
##  Median :57.00  Median : 2401   Median : 5.700   Median : 5.710
##  Mean   :57.46  Mean   : 3933   Mean   : 5.731   Mean   : 5.735
##  3rd Qu.:59.00  3rd Qu.: 5324   3rd Qu.: 6.540   3rd Qu.: 6.540
##  Max.   :95.00  Max.   :18823   Max.   :10.740   Max.   :58.900
##
##      z
##  Min.   : 0.000
##  1st Qu.: 2.910
##  Median : 3.530
##  Mean   : 3.539
##  3rd Qu.: 4.040
##  Max.   :31.800
##
```

```
diamonds %>%
  ggplot(aes(x = price)) +
  geom_histogram(binwidth = 100) +
  tema
```



Exercício 5.3.3

Quantos diamantes têm 0,99 quilates? Quantos têm 1 quilate? Qual você acha que é a causa dessa diferença?

Solução. Existem 23 diamantes com 0,99 quilates, contra 1558 diamantes com 1 quilate. Provavelmente a concentração de diamantes de 1 quilate se deve a arredondamento.

```
## # A tibble: 2 x 2
##   carat     n
##   <dbl> <int>
## 1 0.99     23
## 2 1        1558
```

Exercício 5.3.4

Compare e contraste `coord_cartesian` versus `xlim()` ou `ylim()` ao dar zoom em um histograma. O que acontece se você não configurar `binwidth`? O que acontece se você tentar dar zoom para que apenas meia barra seja mostrada?

Solução. Ao utilizar `coord_cartesian()` a restrição nos eixos `x` e `y` ocorrem após calculados os valores do gráfico e desenhados os geoms, dessa forma, o cálculo não é afetado pelos limites, apenas é feito o zoom. Já para `xlim` e `ylim`, os filtros são aplicados antes da construção do gráfico e as restrições são levadas em consideração, dessa forma, temos pontos que serão realmente descartados, e o leiaute do gráfico acaba ficando bem diferente.

```
diamonds %>%
  ggplot(aes(carat)) +
  geom_histogram() +
  xlim(0,1) +
  tema

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## Warning: Removed 17502 rows containing non-finite values (`stat_bin()`).

## Warning: Removed 2 rows containing missing values (`geom_bar()`).
```



```
library(ggplot2)
library(dplyr)

diamonds %>%
  ggplot(aes(carat)) +
  geom_histogram() +
  coord_cartesian(xlim = c(0,1)) +
  tema

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



5.4 Valores faltantes

Exercício 5.4.1

O que acontece com valores faltantes em um histograma? O que ocorre com valores faltantes em um gráfico de barras? Por que há uma diferença?

Solução. A construção de um histograma considera valores continuos, dessa forma os valores faltantes são descartados, uma vez que não é possível dispor valores faltantes na ordenação dos valores (`NA > 0` não resulta em um valor lógico). Já para o gráfico de barras, como são considerados valores categóricos, os valores faltantes são exibidos como uma nova categoria.

```

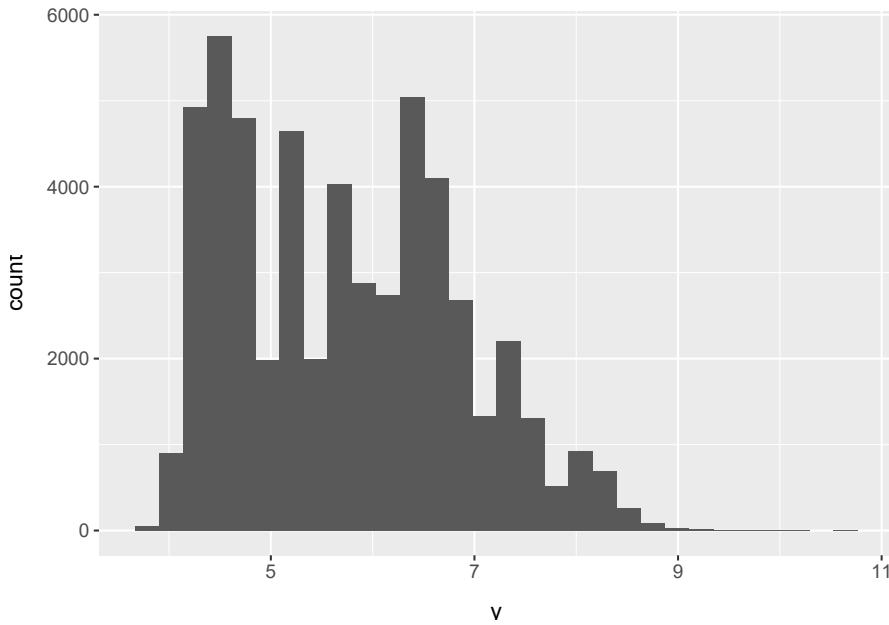
diamonds2 <- diamonds %>%
  mutate(y = ifelse(y < 3 | y > 20, NA, y))

diamonds2 %>%
  ggplot(aes(x = y)) +
  geom_histogram() +
  tema

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

## Warning: Removed 9 rows containing non-finite values (`stat_bin()`).

```

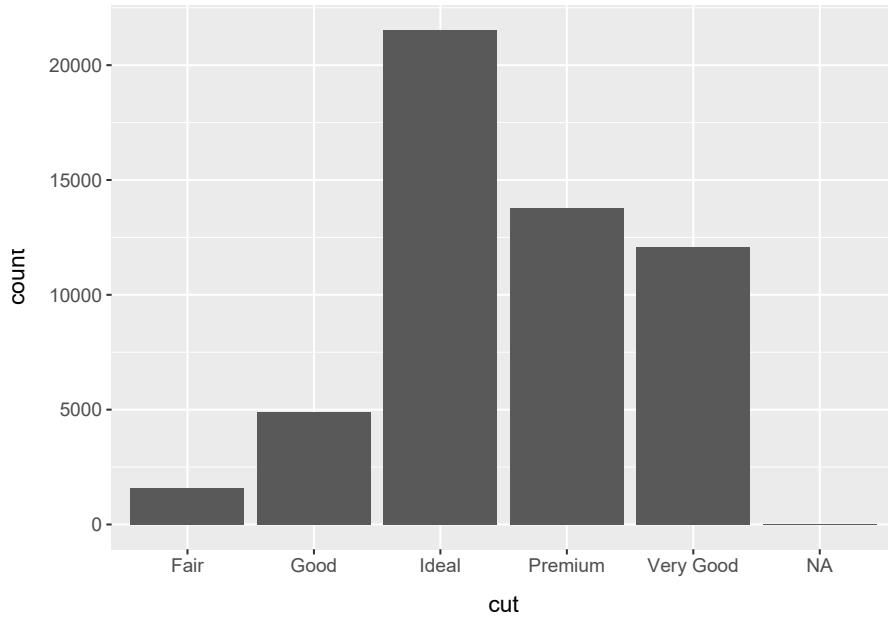


```

diamonds3 <- diamonds2 %>%
  mutate(cut = ifelse(is.na(y), NA, as.character(cut)))

diamonds3 %>%
  ggplot(aes(x = cut)) +
  geom_bar() +
  tema

```



Exercício 5.4.2

O que `na.rm = TRUE` faz em `mean()` e `sum()`?

Solução. O parâmetro `na.rm` serve para indicar que devem ser excluídos da soma ou da média os valores faltantes.

```
a <- c(1, 2, 3, 4, NA, 6, 7, 8, 9, 10)
mean(a)
```

```
## [1] NA
```

```
mean(a, na.rm = TRUE)
```

```
## [1] 5.555556
```

```
sum(a)
```

```
## [1] NA

sum(a, na.rm = TRUE)

## [1] 50
```

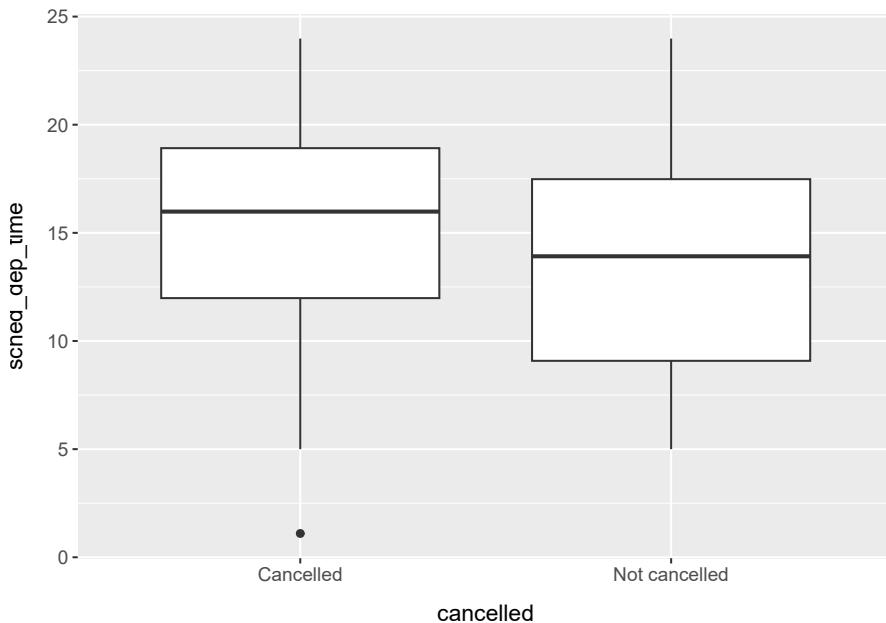
5.5 Covariação

Exercício 5.5.1

Use o que você aprendeu para melhorar a visualização dos tempos de decolagem dos voos cancelados *versus* não cancelados.

Solução.

```
flights %>%
  mutate(
    cancelled = ifelse(is.na(dep_time), "Cancelled", "Not cancelled"),
    sched_hour = sched_dep_time %% 100,
    sched_min = sched_dep_time %% 100,
    sched_dep_time = sched_hour + (sched_min / 60)
  ) %>%
  ggplot(aes(cancelled, sched_dep_time)) +
  geom_boxplot() +
  tema
```



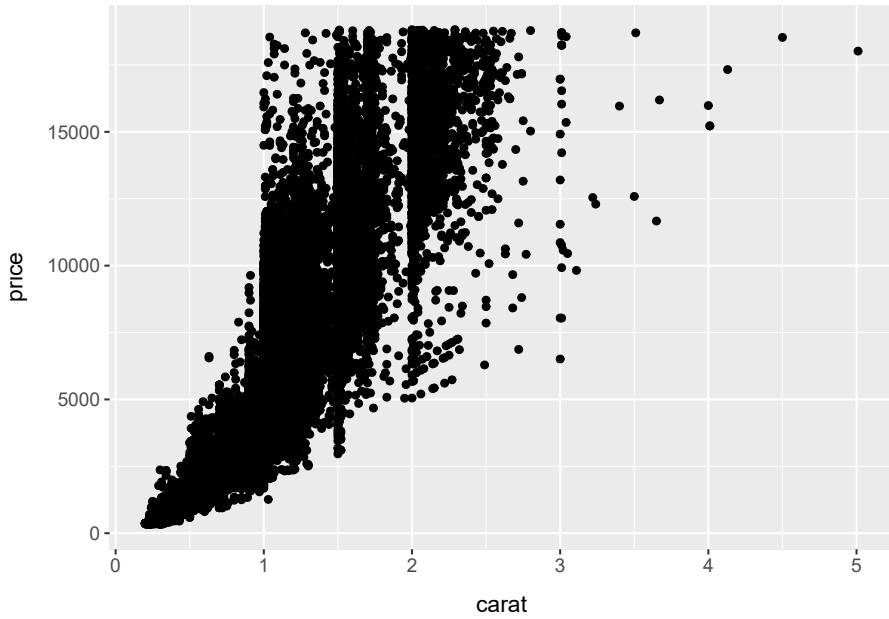
Exercício 5.5.2

Qual variável no conjunto de dados dos diamantes é mais importante para prever o preço de um diamante? Como essa variável está correlacionada ao corte (cut)? Por que a combinação desses dois relacionamentos leva a diamantes de menor qualidade serem mais caros?

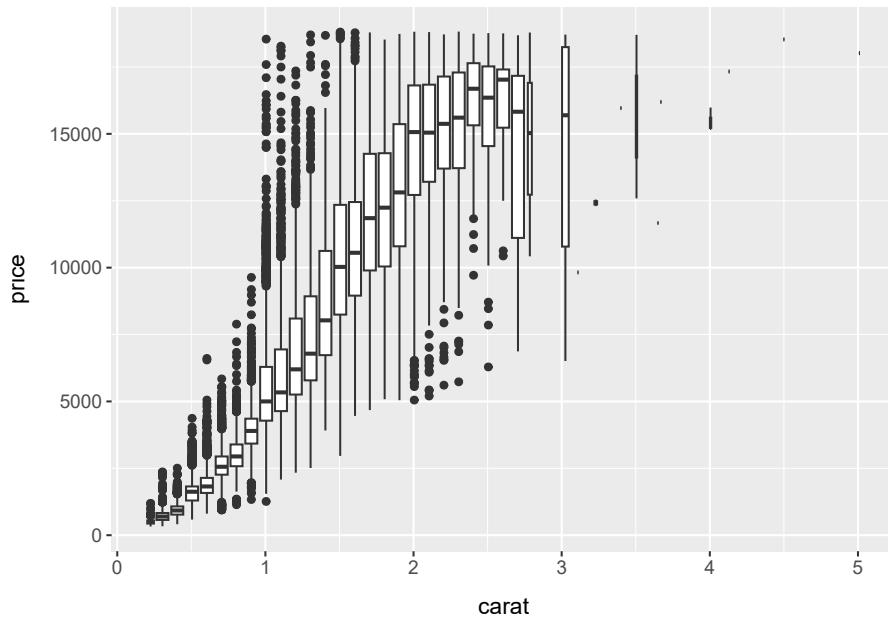
Solução. Vamos considerar na análise as seguintes variáveis: carat, cut, color e clarity.

Para avaliar a correlação entre carat e price (duas variáveis contínuas), podemos usar `geom_point()`, `geom_boxplot()` com a variável independente discretizada ou `geom_quantile()`. Vamos avaliar o melhor dos cenários:

```
diamonds %>%
  ggplot(aes(carat, price)) +
  geom_point() +
  tema
```



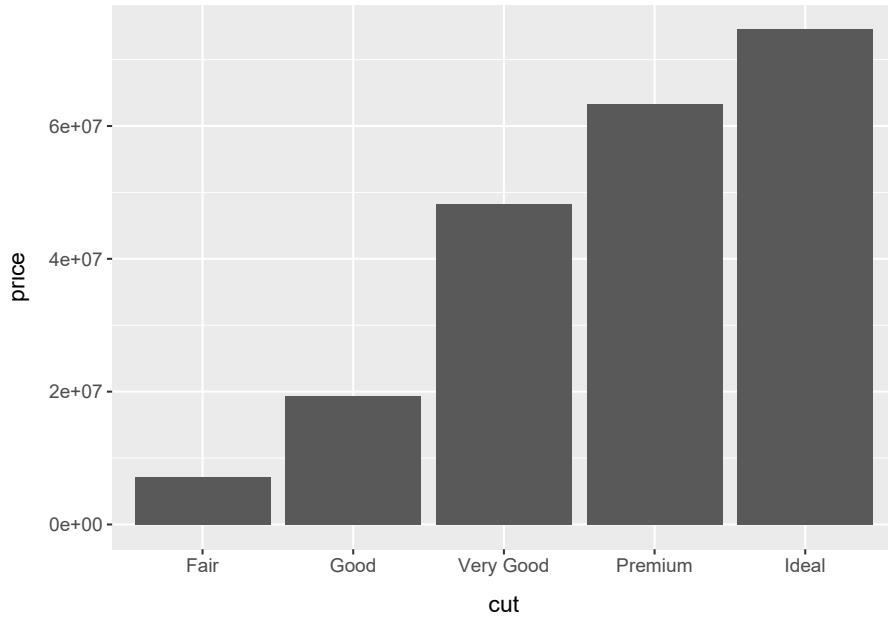
```
diamonds %>%
  ggplot(aes(carat, price)) +
  geom_boxplot(aes(group = cut_width(carat, .1))) +
  tema
```



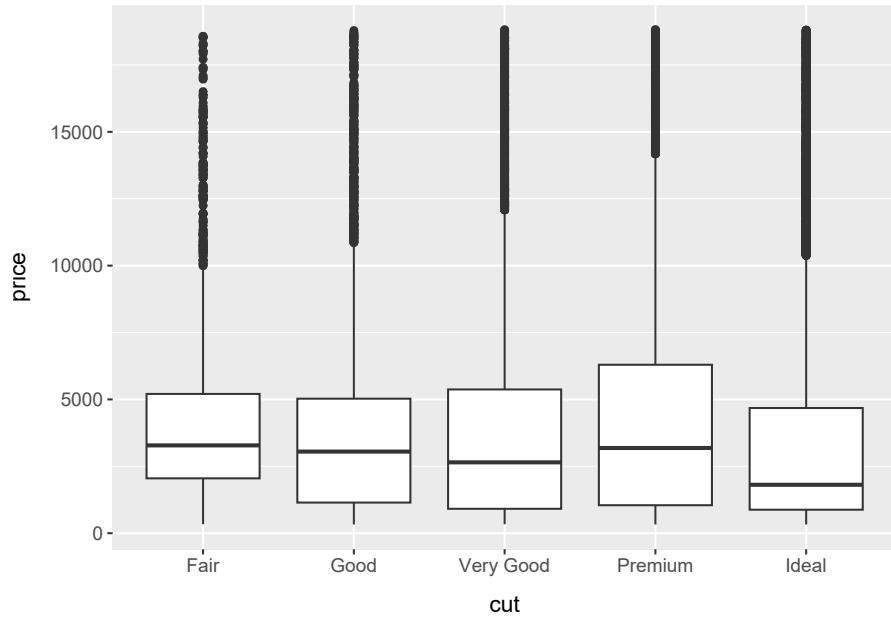
Com base nas imagens acima, podemos ver que existe uma relação positiva entre `carat` e `price`, o que indica que possivelmente essas duas variáveis estão bem correlacionadas. Note ainda que a representação via boxplot ficou um pouco melhor do que a representação por pontos.

Vamos agora avaliar a variável `cut`. Por se tratar de uma variável discreta, podemos utilizar `geom_col()`, `geom_boxplot()`, `geom_dotplot()` ou `geom_violin()`. Vamos avaliar cada um deles.

```
diamonds %>%
  ggplot(aes(cut, price)) +
  geom_col() +
  tema
```

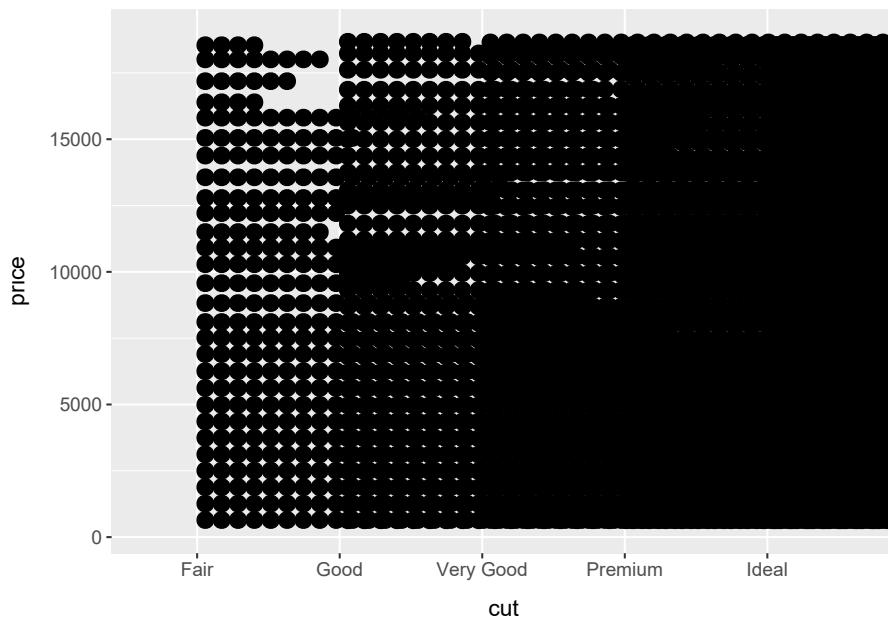


```
iamonds %>%
  ggplot(aes(cut, y = price)) +
  geom_boxplot() +
  tema
```



```
iamonds %>%
  ggplot(aes(cut, price)) +
  geom_dotplot(binaxis = "y") +
  tema
```

```
## Bin width defaults to 1/30 of the range of the data. Pick better value with
## `binwidth`.
```



```
diamonds %>%
  ggplot(aes(cut, price)) +
  geom_violin(scale = "area") +
  tema
```



Notamos que os gráficos gerados por `geom_col()` e `geom_dotplot()` não geraram nenhum resultado interessante. Este devido à poluição visual e aquele devido a mostrar uma contagem dos elementos em cada grupo, e não a associação entre as variáveis.

Tanto com `geom_boxplot()`, quanto com `geom_violin()`, podemos perceber que há uma correlação negativa muito fraca entre as variáveis e, desta forma, podemos considerar que `cut` não é interessante para predizer os preços dos diamantes.

Sigamos para a variável `color`:

```
library(ggplot2)
diamonds %>%
  ggplot(aes(color, price)) +
  geom_boxplot() +
  theme
```



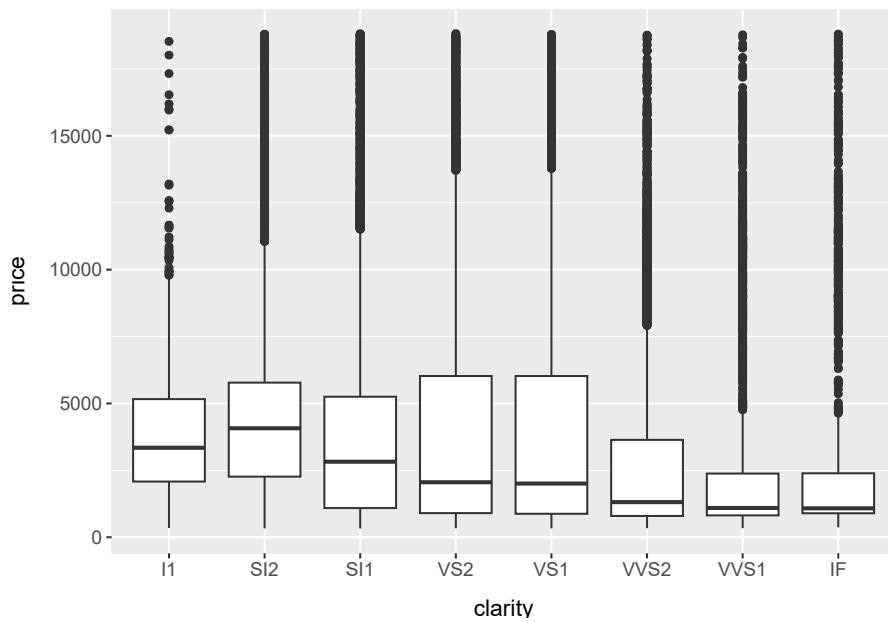
```
iamonds %>%
  ggplot(aes(color, price)) +
  geom_violin(scale = "area") +
  tema
```



Como podemos perceber, a relação entre as variáveis não é significativa, portanto, descartaremos `color`.

Seguindo em frente, precisamos avaliar a variável `clarity`:

```
diamonds %>%
  ggplot(aes(clarity, price)) +
  geom_boxplot() +
  tema
```



```
iamonds %>%
  ggplot(aes(color, price)) +
  geom_violin(scale = "area") +
  tema
```



Notamos também que a variável não tem correção com o preço.

Concluímos, portanto, que a melhor variável para predizer o preço do diamante é carat.

Agora, avaliaremos a relação entre carat e cut.

```
diamonds %>%
  ggplot(aes(cut, carat)) +
  geom_boxplot() +
  tema
```



Há uma relação negativa muito leve entre `cut` e `carat`, mas isso não é suficiente para dizer que uma impacta na outra. Há grande variabilidade de `carat` dentro de cada tipo de corte (`cut`) e, nota-se, os diamantes de grande quilate (provavelmente pedras grandes), tem um corte apenas justo. Isso pode se dar ao fato de que, quanto menor o diamante, melhor precisa ser o corte para que se consiga um bom valor. Além disso, é presumível que é mais fácil vender um diamante pequeno do que um grande, por isso talvez o preço não seja tão alto.

Exercício 5.5.3

Instale o pacote `ggstance` e crie um boxplot horizontal. Como isso se compara a usar `coord_flip()`?

Solução.

```
library(ggstance)

## 
## Attaching package: 'ggstance'

## The following objects are masked from 'package:ggplot2':
## 
##     geom_errorbarh, GeomErrorbarh
```

```
diamonds %>%
```

```
  ggplot(aes(carat, cut)) +
```

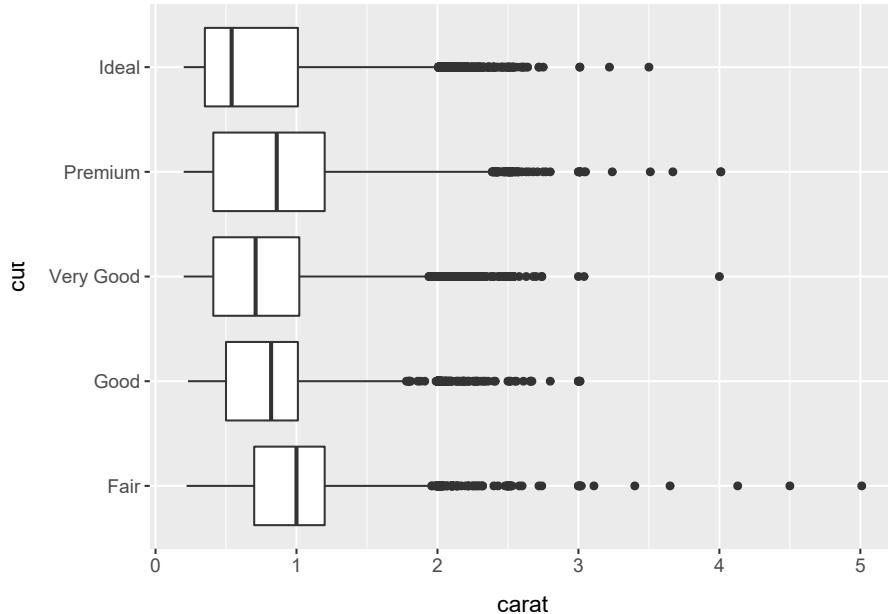
```
    geom_boxplot() +
```

```
    tema
```

```
## Warning: The following aesthetics were dropped during statistical transformation: x
## i This can happen when ggplot fails to infer the correct grouping structure in
##   the data.
## i Did you forget to specify a `group` aesthetic or to convert a numerical
##   variable into a factor?
```

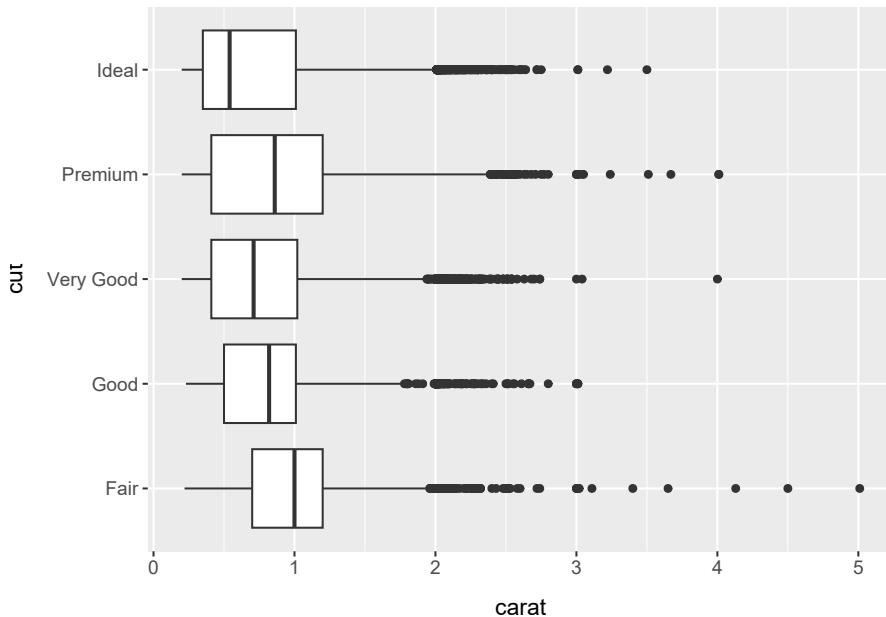
```
## Warning: Using the `size` aesthetic with geom_segment was deprecated in ggplot2 3.4.0.
## i Please use the `linewidth` aesthetic instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## Warning: Using the `size` aesthetic with geom_polygon was deprecated in ggplot2 3.4.0.
## i Please use the `linewidth` aesthetic instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



```
library(ggplot2)
library(dplyr)

diamonds %>%
  ggplot(aes(cut, carat)) +
  geom_boxplot() +
  coord_flip() +
  tema
```



A diferença está apenas no mapeamento.

Exercício 5.5.4

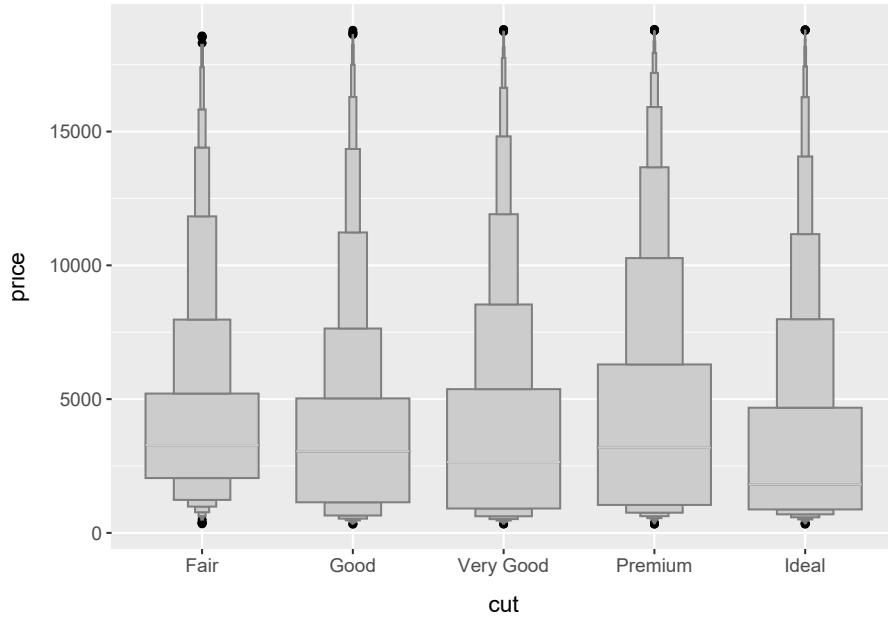
Um problema com boxplots é que eles foram desenvolvidos em uma era de conjuntos de dados muito menores e tendem a exibir um número proibitivamente grande de “valores fora da curva”. Uma abordagem para remediar esse problema é o *letter value plot*. Instale o `lvplot` e tente usar `geom_lv()` para exibir a distribuição de preço versus corte. O que você aprendeu? Como você interpreta os gráficos?

Solução. ???

```
library(lvplot)

diamonds %>%
  ggplot(aes(cut, price)) +
```

```
geom_lv(width.method = "height") +  
tema
```

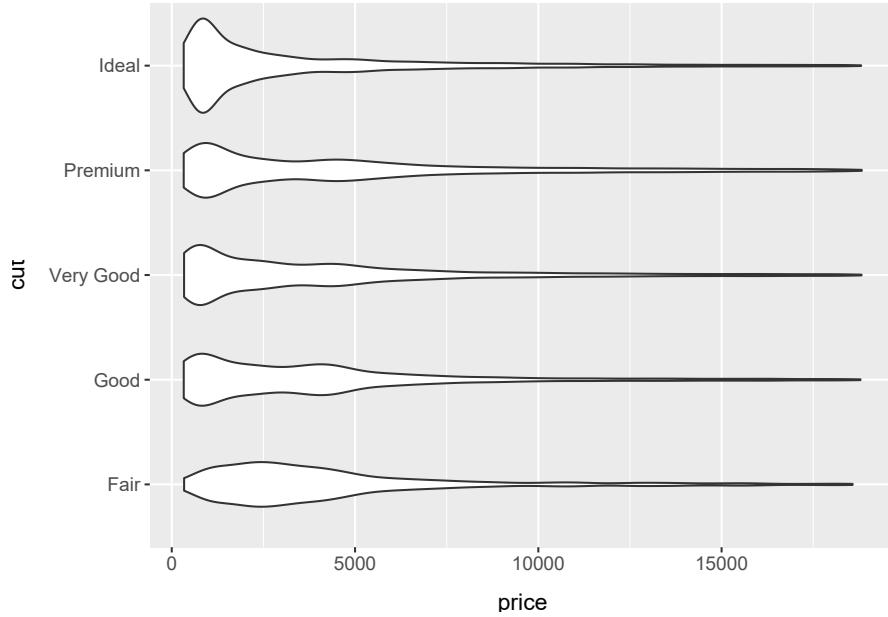


Exercício 5.5.5

Compare e contraste `geom_violin()` com `geom_histogram()` facetado, ou um `geom_freqpoly()` colorido. Quais são os prós e contras de cada método?

Solução. Com o polígono de frequência é mais fácil comparar os grupos, uma vez que as linhas são sobrepostas, contudo muitas vezes pode se tornar complicado visualizar o comportamento/variação de cada grupo individualmente. O violino e o histograma permitem visualizar a distribuição em cada grupo, contudo fica mais complicado fazer a comparação.

```
# Violin  
diamonds %>%  
  ggplot(aes(cut, price)) +  
    geom_violin() +  
    coord_flip() +  
    tema
```



```
# Histogram
diamonds %>%
  ggplot(aes(price)) +
  geom_histogram() +
  facet_wrap(~ cut, ncol = 1, scale = "free_y") +
  tema

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
#Frequency Polygon
diamonds %>%
  ggplot(aes(price, ..density.., color = cut)) +
  geom_freqpoly(binwidth = 500) +
  tema
```

```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(density)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



Exercício 5.5.6

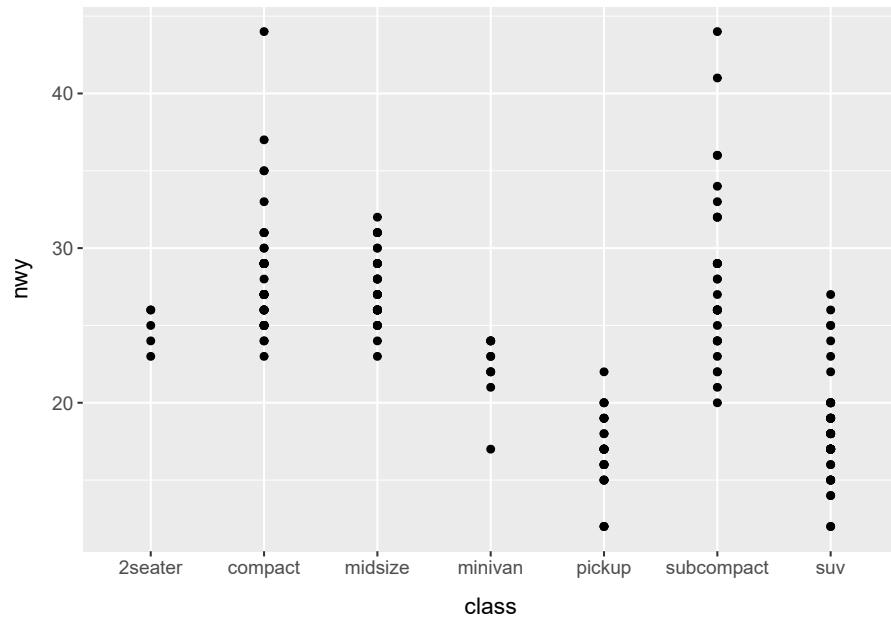
Se você tem um conjunto de dados pequeno, às vezes é útil usar `geom_jitter()` para ver a relação entre uma variável contínua e uma categórica. O pacote **ggbeeswarm** fornece alguns métodos similares a `geom_jitter()`. Liste-os e descreva brevemente o que cada um faz.

Solução. O pacote oferece duas geoms. A primeira, `geom_quasirandom()` mistura o `jitter` com a aparência do gráfico violino. A segunda, `geom_beeswarm()` produz gráficos parecidos com violinos, mas com alguma sobreposição.

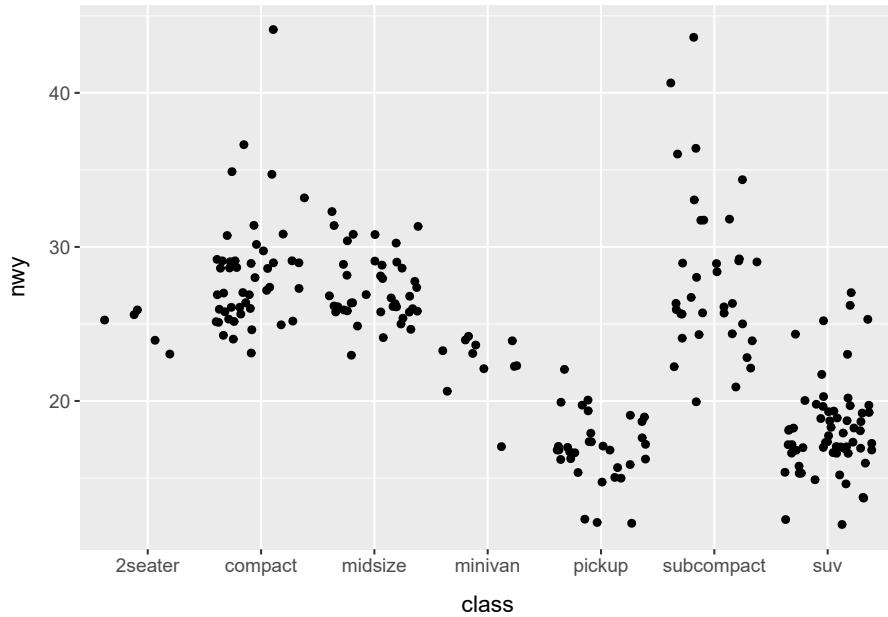
Em comparação ao `geom_jitter()`, o pacote `ggbeeswarm()` permite uma melhor visualização dos clusteres, caso existam. No nosso exemplo, os clusters seriam as próprias classes, ele evita que os pontos de uma classe se aproximem demais da classe ao lado, gerando uma melhor visualização. Também mantém mais ou menos agrupados as “alturas” no eixo y, isto é, o erro é colocado apenas em uma das direções.

```
library(ggbeeswarm)

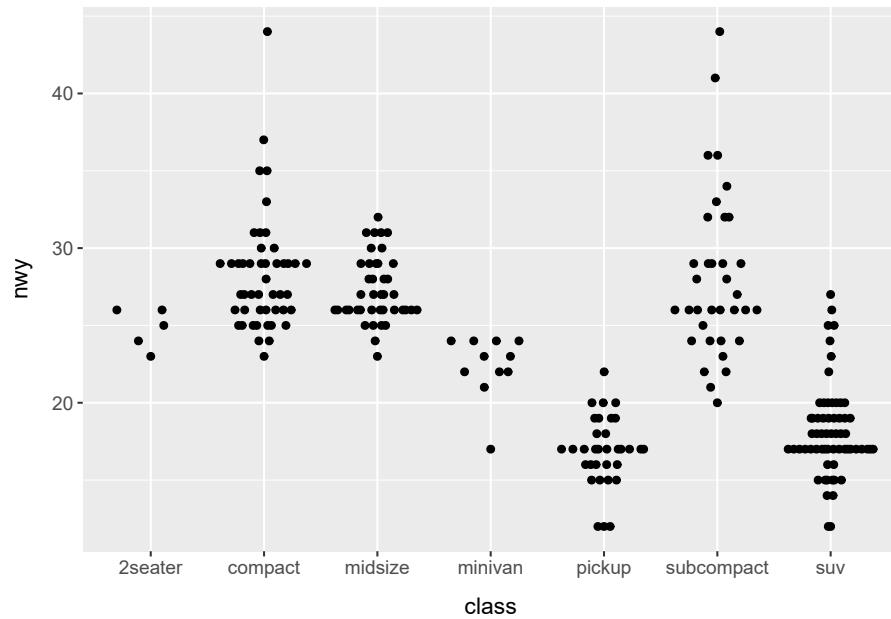
mpg %>%
  ggplot(aes(class, hwy)) +
  geom_point() +
  tema
```



```
mpg %>%
  ggplot(aes(class, hwy)) +
  geom_jitter() +
  tema
```



```
mpg %>%
  ggplot(aes(class, hwy)) +
  geom_quasirandom() +
  tema
```



```
mpg %>%
  ggplot(aes(class, hwy)) +
  geom_beeswarm() +
  tema
```

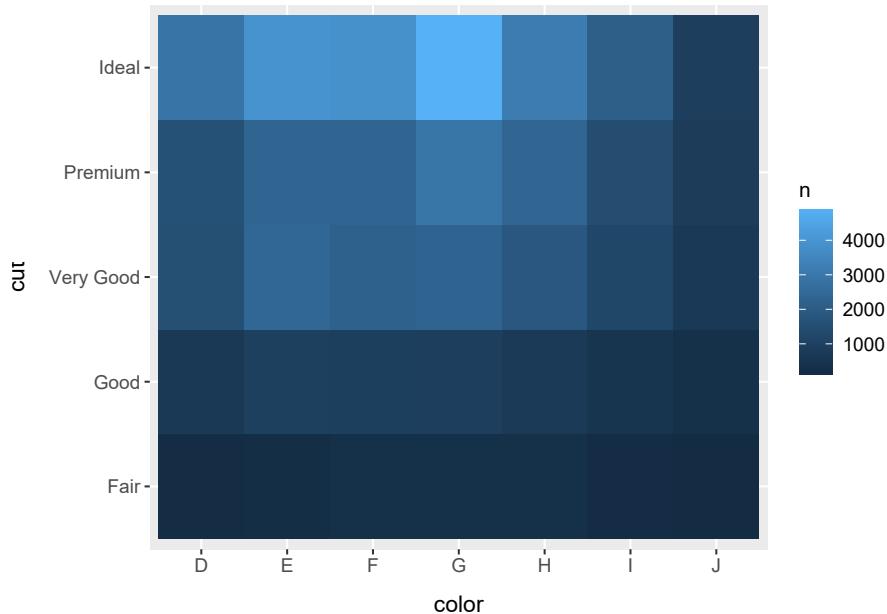


Exercício 5.5.7

Como você alteraria a escala do conjunto de dados diamonds para mostrar mais claramente a distribuição de corte dentro de cor ou cor dentro de corte?

Solução. Pode-se melhorar usando a proporção de cut dentro de color, porém a mudança não foi tão significativa.

```
iamonds %>%
  count(color, cut) %>%
  ggplot(aes(color, cut)) +
  geom_tile(mapping = aes(fill = n)) +
  tema
```



```
library(dplyr)
library(ggplot2)

diamonds %>%
  count(color, cut) %>%
  group_by(color) %>%
  mutate(
    prop = n / sum(n)
  ) %>%
  ggplot(aes(color, cut)) +
  geom_tile(mapping = aes(fill = prop)) +
  tema
```



Exercício 5.5.8

Use `geom_tile()` junto de `dplyr` para explorar como os atrasos médios dos voos variam por destino e mês. O que dificulta a leitura do gráfico? Como você melhorá-lo?

Solução. ???

```
flights %>%
  group_by(month, dest) %>%
  summarise(
    dep_delay = mean(dep_delay, na.rm = TRUE) # Agrupar por mês e destino
  ) %>%
  group_by(dest) %>%
  filter(n() == 12) %>%
  ungroup() %>%
  ggplot(aes(factor(month), dest, fill = dep_delay)) +
  geom_tile()

## `summarise()` has grouped output by 'month'. You can override using the
## `.groups` argument.
```



Exercício 5.5.9

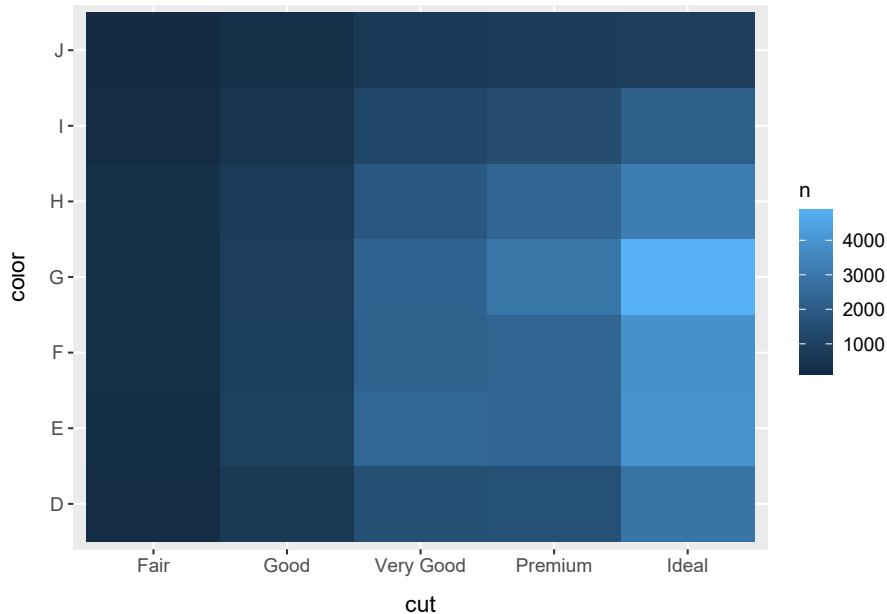
Por que é um pouco melhor usar `aes(x = color, y = cut)` em vez de `aes(c = cut, y = color)` no exemplo anterior?

Solução. ???

```
iamonds %>%
  count(color, cut) %>%
  ggplot(aes(color, cut)) +
  geom_tile(mapping = aes(fill = n)) +
  tema
```



```
iamonds %>%
  count(cut, color) %>%
  ggplot(aes(cut, color)) +
  geom_tile(mapping = aes(fill = n)) +
  tema
```



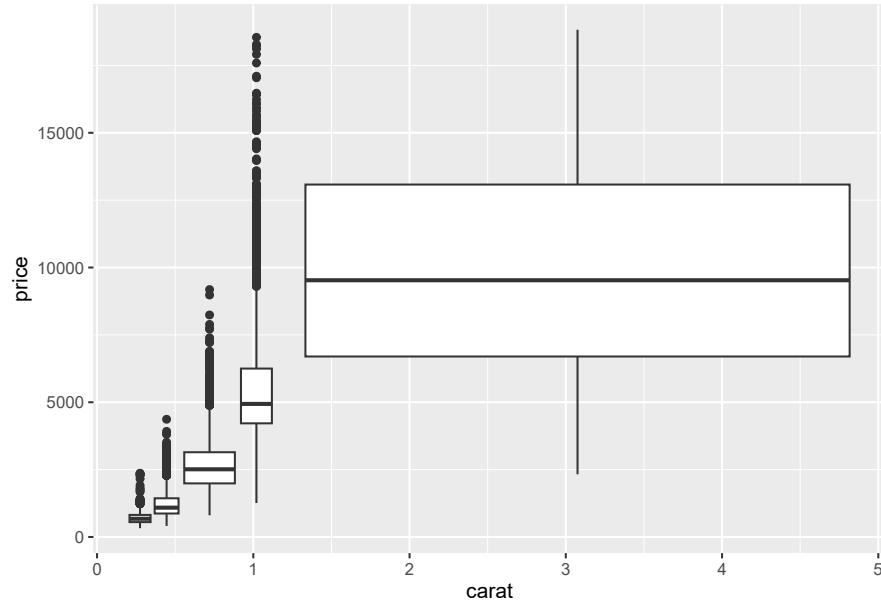
Exercício 5.5.10

Em vez de resumir a distribuição condicional com um boxplot, pode-se usar um polígono de frequência. O que você precisa considerar ao usar `cut_width()` versus `cut_number()`? Como isso impacta uma visualização da distribuição 2D de `carat` e `price`?

Solução. Utilizamos `cut_width()` quando queremos determinar o tamanho das classes que serão exibidas e `cut_number()` quando queremos determinar o número de classes.

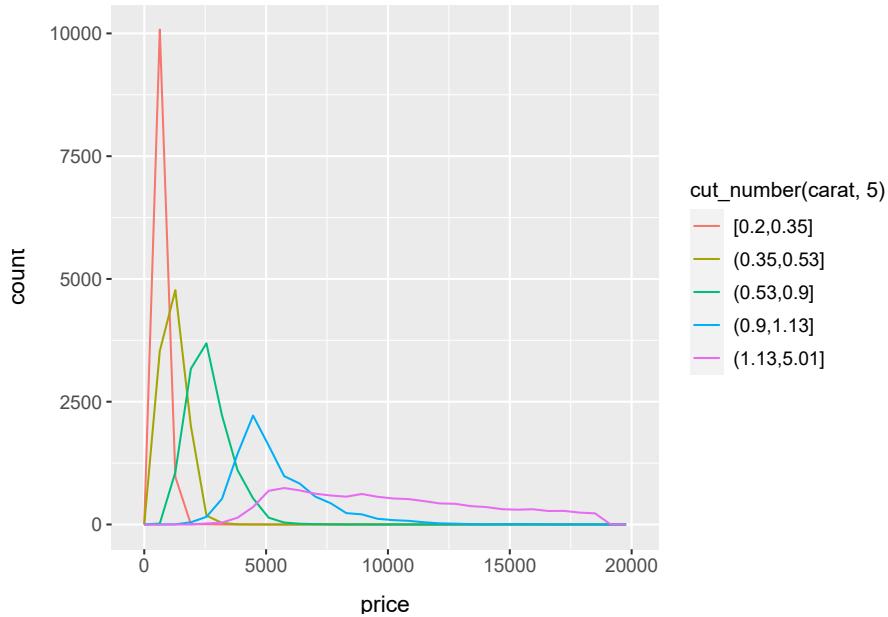
Se o número total de classes for muito grande, a visualização fica comprometida. Se for pequeno demais, não captaremos comportamentos importantes.

```
iamonds %>%
  ggplot(aes(carat, price)) +
  geom_boxplot(aes(group = cut_number(carat, 5)))
```



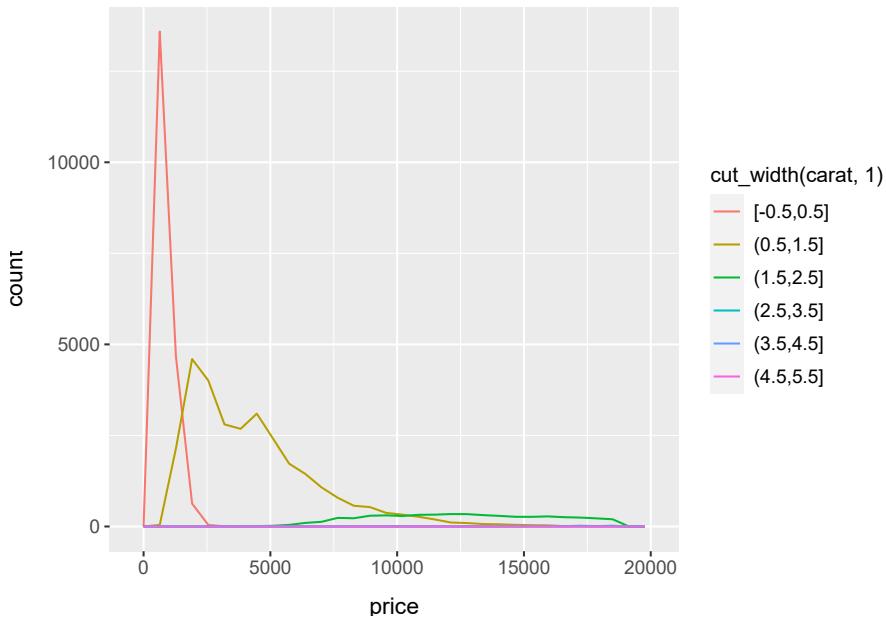
```
diamonds %>%
  ggplot(aes(price)) +
  geom_freqpoly(aes(color = cut_number(carat, 5))) +
  tema
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
diamonds %>%
  ggplot(aes(price)) +
  geom_freqpoly(aes(color = cut_width(carat, 1))) +
  tema
```

`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.

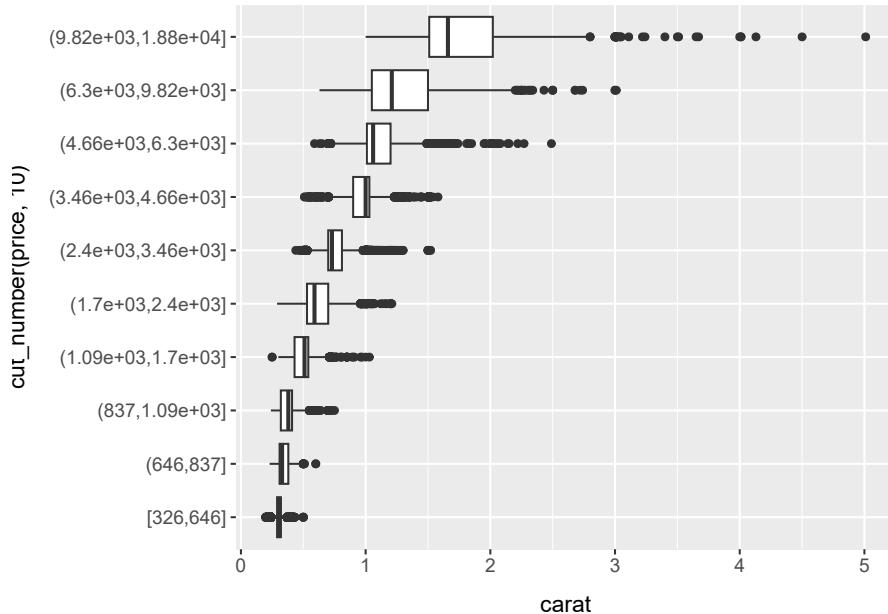


Exercício 5.5.11

Visualize a distribuição de carat, particionada por price.

Solução. Inicialmente vamos utilizar boxplot para visualizar a distribuição. Vamos dividir os preços em 10 grupos.

```
diamonds %>%
  ggplot(aes(cut_number(price, 10), carat)) +
  geom_boxplot() +
  coord_flip() +
  tema
```



Note que, `cut_number()` dividiu os grupos de modo a ter a mesma quantidade de registros em cada classe e isso acaba gerando classes de larguras diferentes. É mais fácil comparar classes de mesma largura, por isso vamos refazer a visualização utilizando `cut_width()`:

```
diamonds %>%
  ggplot(aes(cut_width(price, 2000), carat)) +
  geom_boxplot() +
  coord_flip() +
  tema
```



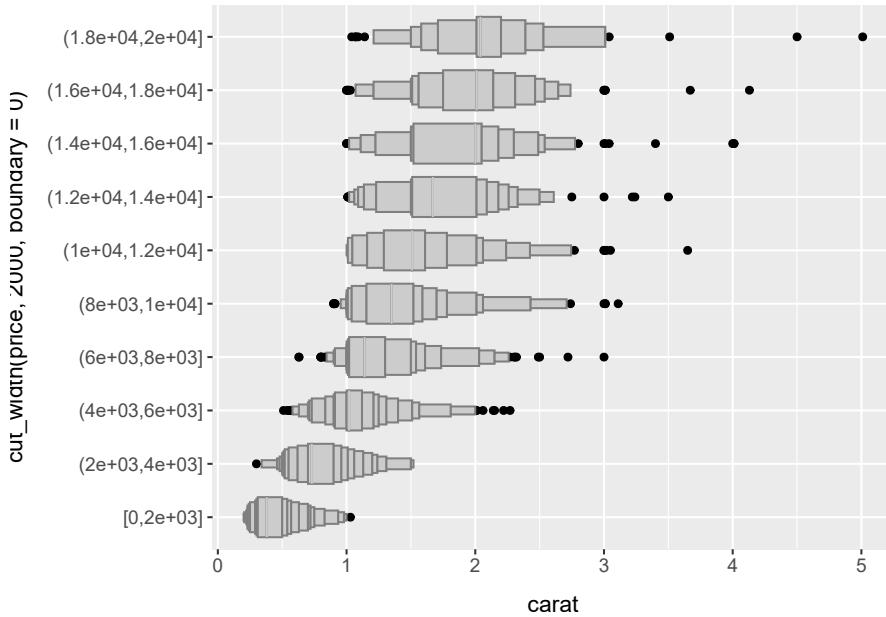
A visualização ficou um pouco melhor, porém temos uma classe que inclui valores negativos. Vamos corrigir isso utilizando o argumento `boundary`:

```
diamonds %>%
  ggplot(aes(cut_width(price, 2000, boundary = 0), carat)) +
  geom_boxplot() +
  coord_flip() +
  tema
```



Como existem muitos *outliers*, vamos tentar utilizar o *letter value*:

```
diamonds %>%
  ggplot(aes(cut_width(price, 2000, boundary = 0), carat)) +
  geom_lv() +
  coord_flip() +
  tema
```



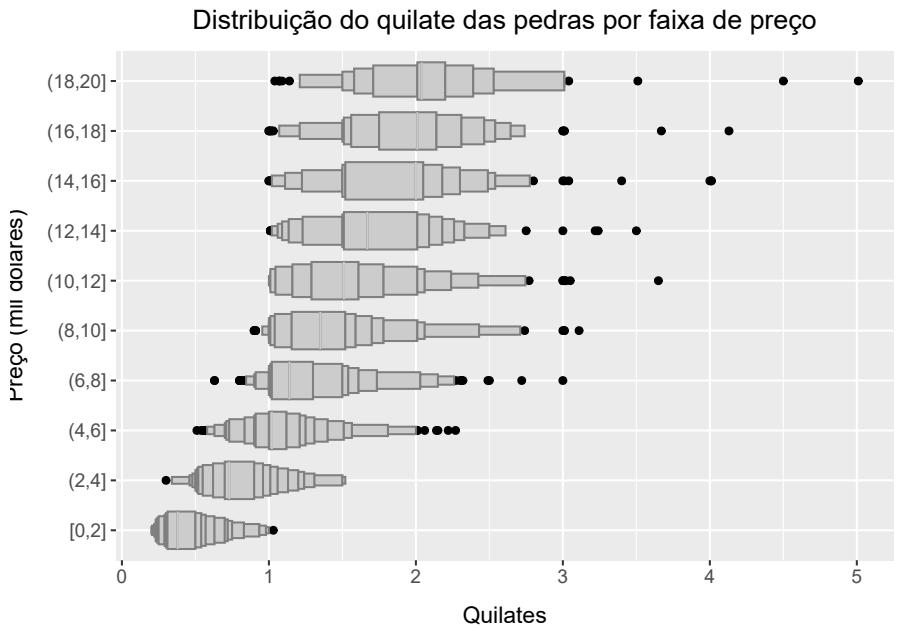
Para melhorar a visualização das classes, vamos também mudar a unidade de `price` de dólares para milhares de dólares, basta dividirmos os valores por 1000:

```
iamonds %>%
  mutate (price = price / 1000) %>%
  ggplot(aes(cut_width(price, 2, boundary = 0), carat)) +
  geom_lv() +
  coord_flip() +
  tema
```



Agora vamos melhorar os títulos do gráfico:

```
diamonds %>%
  mutate (price = price / 1000) %>%
  ggplot(aes(cut_width(price, 2, boundary = 0), carat)) +
  geom_lv() +
  coord_flip() +
  labs(
    title = "Distribuição do quilate das pedras por faixa de preço",
    y = "Quilates",
    x = "Preço (mil dólares)"
  ) +
  tema
```

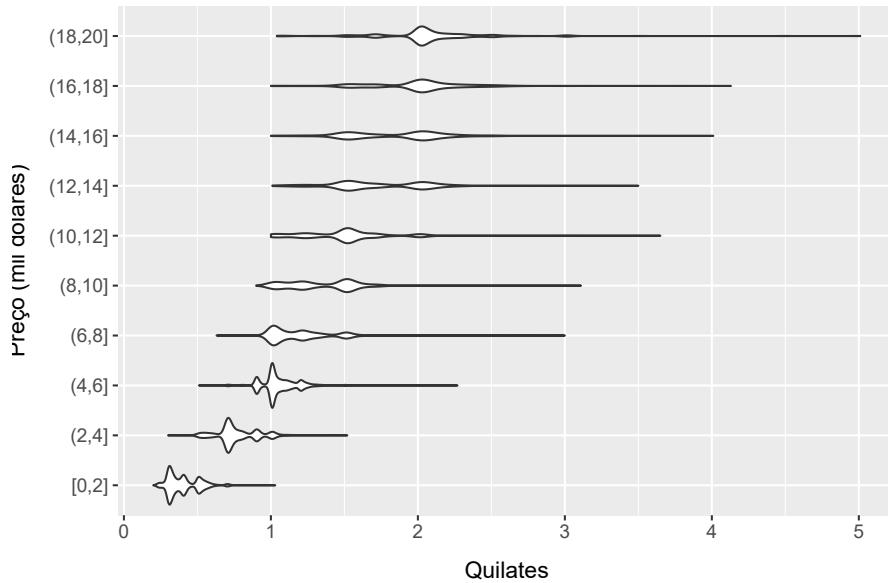


O gráfico está pronto.

Note que poderíamos ter utilizado a `geom_violin()`, porém, neste caso, não teríamos uma visão muito boa dos outliers.

```
diamonds %>%
  mutate(price = price / 1000) %>%
  ggplot(aes(cut_width(price, 2, boundary = 0), carat)) +
  geom_violin() +
  coord_flip() +
  labs(
    title = "Distribuição do quilate das pedras por faixa de preço",
    y = "Quilates",
    x = "Preço (mil dólares)"
  ) +
  tema
```

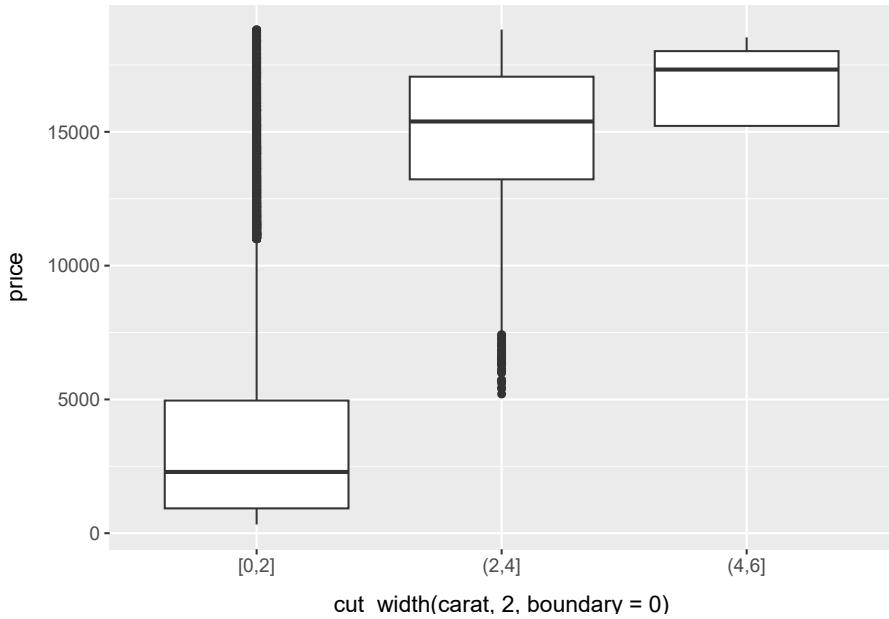
Distribuição do quilate das pedras por faixa de preço

**Exercício 5.5.12**

Como a distribuição de preços de diamantes muito grandes se compara à de diamantes pequenos? É como você esperava ou isso lhe surpreende?

Solução. ???

```
diamonds %>%
  ggplot(aes(cut_width(carat, 2, boundary = 0), price)) +
  geom_boxplot() +
  tema
```



Exercício 5.5.13

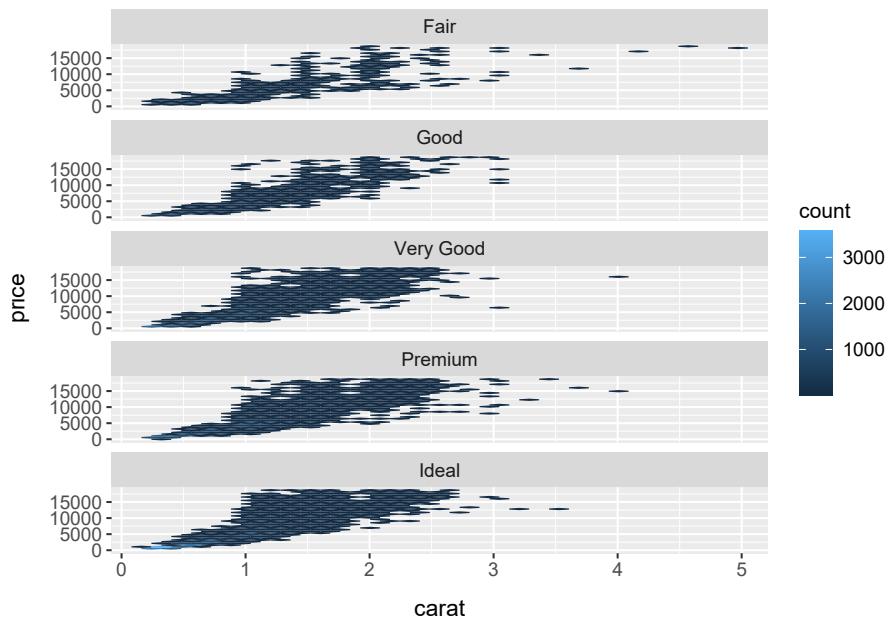
Combine duas técnicas que você aprendeu para visualizar a distribuição combinada de `cut`, `carat` e `price`.

Solução.

```
diamonds %>%
  ggplot(aes(cut_number(carat, 5), price, color = cut)) +
  geom_boxplot() +
  tema
```



```
iamonds %>%
  ggplot(aes(carat, price)) +
  geom_hex() +
  facet_wrap(~ cut, ncol = 1) +
  tema
```



```
library(ggplot2)
library(dplyr)
library(scales)

tema <- theme_minimal() +
  theme(panel.grid.major = grid::grid(), panel.grid.minor = grid::grid())
tema
```

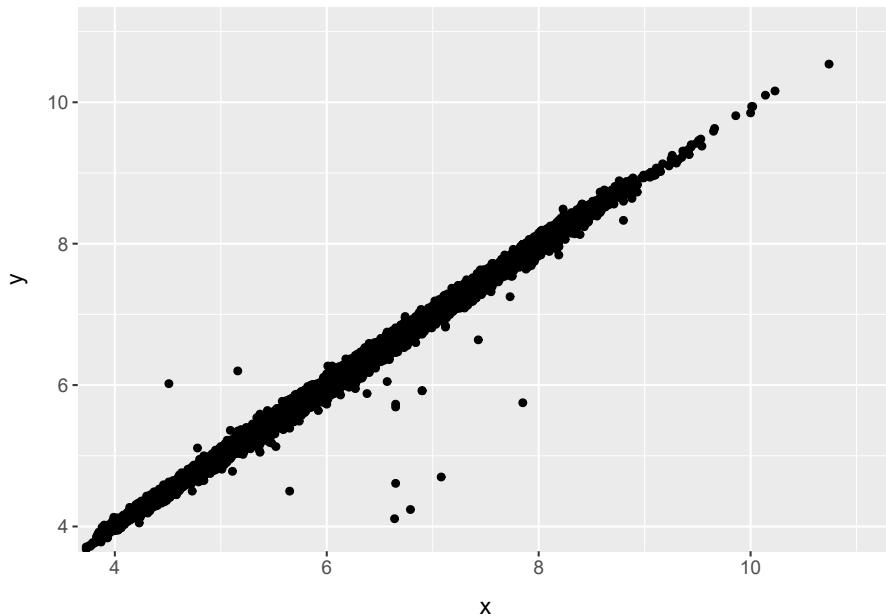
```
diamonds %>%
  ggplot(aes(cut_number(carat, 10), cut, fill = desc(price))) +
  geom_hex() +
  tema
```



Exercício 5.5.14

Gráficos bidimensionais revelam pontos fora da curva que não são visíveis em gráficos unidimensionais. Por exemplo, alguns pontos no gráfico a seguir tem uma combinação incomum de valores de x e y , que faz pontos ficarem fora da curva, mesmo embora seus valores x e y pareçam normais quando examidanos separadamente.

```
ggplot(data = diamonds) +  
  geom_point(mapping = aes(x = x, y = y)) +  
  coord_cartesian(xlim = c(4, 11), ylim = c(4, 11)) +  
  tema
```



Por que um diagrama de dispersão é uma exibição melhor do que um diagrama de caixa neste caso?

Solução. Como os valores de x e y são fortemente relacionados, os outliers não vão aparecer no extremo de uma ou outra coordenada, mas em proporções desiguais dos diamantes.

5.6 Padrões e modelos

Não temos exercícios nesta seção.

5.7 Chamadas ggplot2

Não temos exercícios nesta seção.

5.8 Aprendendo mais

Não temos exercícios nesta seção.



6

Fluxo de trabalho: projetos

6.1 O que é real?

Não temos exercícios nesta seção.

6.2 Onde sua análise vive?

Não temos exercícios nesta seção.

6.3 Caminhos e diretórios

Não temos exercícios nesta seção.

6.4 Projetos RStudio

Não temos exercícios nesta seção.

6.5 Resumo

Não temos exercícios nesta seção.



Parte II

Wrangle



7

Tibbles com tibble

7.1 Introdução

Não temos exercícios nesta seção.

7.2 Criando tibbles

Não temos exercícios nesta seção.

7.3 Tibbles *versus* `data.frame`

Não temos exercícios nesta seção.

7.4 Interagindo com códigos mais antigos

Exercício 7.4.1

Como você consegue dizer se um objeto é um tibble? (Dica: tente imprimir `mtcars`, que é um dat

Solução.

```
print(mtcars)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
## Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
## Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
## Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
## Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
## Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
## Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
## Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
## Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
## Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
## Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
## Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
## Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
## Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
## Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
## Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
## Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
## Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
## Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
## Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
## Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
## Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
## Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
## AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
## Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
## Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
## Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
## Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
## Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
## Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
## Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
## Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
## Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

```
print(as.tibble(mtcars))
```

```
## Warning: `as.tibble()` was deprecated in tibble 2.0.0.
## i Please use `as_tibble()` instead.
## i The signature and semantics have changed, see `?as_tibble`.
```

```
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.

## # A tibble: 32 x 11
##   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 21      6   160   110  3.9   2.62  16.5     0     1     4     4
## 2 21      6   160   110  3.9   2.88  17.0     0     1     4     4
## 3 22.8    4   108   93   3.85  2.32  18.6     1     1     4     1
## 4 21.4    6   258   110  3.08  3.22  19.4     1     0     3     1
## 5 18.7    8   360   175  3.15  3.44  17.0     0     0     3     2
## 6 18.1    6   225   105  2.76  3.46  20.2     1     0     3     1
## 7 14.3    8   360   245  3.21  3.57  15.8     0     0     3     4
## 8 24.4    4   147.   62   3.69  3.19  20       1     0     4     2
## 9 22.8    4   141.   95   3.92  3.15  22.9     1     0     4     2
## 10 19.2   6   168.   123  3.92  3.44  18.3     1     0     4     4
## # i 22 more rows
```

```
str(mtcars)
```

```
## 'data.frame': 32 obs. of 11 variables:
## $ mpg : num 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
## $ cyl : num 6 6 4 6 8 6 8 4 4 6 ...
## $ disp: num 160 160 108 258 360 ...
## $ hp : num 110 110 93 110 175 105 245 62 95 123 ...
## $ drat: num 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
## $ wt : num 2.62 2.88 2.32 3.21 3.44 ...
## $ qsec: num 16.5 17 18.6 19.4 17 ...
## $ vs : num 0 0 1 1 0 1 0 1 1 1 ...
## $ am : num 1 1 1 0 0 0 0 0 0 0 ...
## $ gear: num 4 4 4 3 3 3 4 4 4 ...
## $ carb: num 4 4 1 1 2 1 4 2 2 4 ...
```

```
str(as.tibble(mtcars))
```

```
## # tibble [32 x 11] (S3:tbl_df/tbl/data.frame)
## $ mpg : num [1:32] 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
## $ cyl : num [1:32] 6 6 4 6 8 6 8 4 4 6 ...
## $ disp: num [1:32] 160 160 108 258 360 ...
```

```
## $ hp : num [1:32] 110 110 93 110 175 105 245 62 95 123 ...
## $ drat: num [1:32] 3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
## $ wt : num [1:32] 2.62 2.88 2.32 3.21 3.44 ...
## $ qsec: num [1:32] 16.5 17 18.6 19.4 17 ...
## $ vs : num [1:32] 0 0 1 1 0 1 0 1 1 1 ...
## $ am : num [1:32] 1 1 1 0 0 0 0 0 0 0 ...
## $ gear: num [1:32] 4 4 4 3 3 3 3 4 4 4 ...
## $ carb: num [1:32] 4 4 1 1 2 1 4 2 2 4 ...
```

```
rownames(mtcars)
```

```
## [1] "Mazda RX4"           "Mazda RX4 Wag"      "Datsun 710"
## [4] "Hornet 4 Drive"      "Hornet Sportabout" "Valiant"
## [7] "Duster 360"          "Merc 240D"        "Merc 230"
## [10] "Merc 280"            "Merc 280C"        "Merc 450SE"
## [13] "Merc 450SL"          "Merc 450SLC"      "Cadillac Fleetwood"
## [16] "Lincoln Continental" "Chrysler Imperial" "Fiat 128"
## [19] "Honda Civic"         "Toyota Corolla"   "Toyota Corona"
## [22] "Dodge Challenger"   "AMC Javelin"     "Camaro Z28"
## [25] "Pontiac Firebird"   "Fiat X1-9"       "Porsche 914-2"
## [28] "Lotus Europa"        "Ford Pantera L"  "Ferrari Dino"
## [31] "Maserati Bora"       "Volvo 142E"
```

```
row.names(as.tibble(mtcars))
```

```
## [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14" "15"
## [16] "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28" "29" "30"
## [31] "31" "32"
```

Há várias formas que podem nos ajudar a identificar se o conjunto de dados está organizado como um data frame padrão ou como um tibble:

- Ao utilizar o comando `print()`, um data frame comum imprime todas as observações, enquanto um tibble imprime apenas as 10 primeiras;
- Ao utilizar a função `str()`, o tipo de objeto é impresso;
- Ao utilizar a função `rownames()`, um data frame exibirá os nomes das observações (se houver), enquanto um tibble exibirá sempre um sequência numérica (???).

Exercício 7.4.2

Compare e constraste as seguintes operações em `data.frame` e `tibble` equivalente. Qual é a diferença? Por que os comportamentos do data frame padrão podem lhe causar frustração?

```
df <- data.frame(abc = 1, xyz = "a")
df$x
df[, "xyz"]
df[, c("abc", "xyz")]
```

Solução. Inicialmente vamos definir um `tibble` com o mesmo conteúdo do data frame proposto.

```
tb <- tibble(abc = 1, xyz = "a")
```

A seguir, executaremos os comandos correspondentes para avaliar a saída.

```
tb$x
```

```
## Warning: Unknown or uninitialized column: `x`.
```

```
## NULL
```

```
tb[, "xyz"]
```

```
## # A tibble: 1 × 1
##   xyz
##   <chr>
## 1 a
```

```
tb[, c("abc", "xyz")]
```

```
## # A tibble: 1 × 2
##   abc   xyz
##   <dbl> <chr>
## 1     1     a
```

Concluímos que:

- Ao utilizar o operador `$`, o data frame busca exibir a primeira (?) coluna que contenha `x`, enquanto num tibble, busca-se a coluna nomeada exatamente como `x`;
- Ao utilizar o nome completo de uma única variável com o operador `[`, um data frame imprime um vetor ou um valor singular, enquanto um tibble exibe sempre outro tibble;
- ao usar o operador `[` passando um vetor de variáveis, o data frame padrão retorna outro data frame e um tibble retorna outro tibble. Neste caso o comportamento é similar.

Exercício 7.4.3

Se você tem o nome de uma variável armazenada em um objeto, por exemplo `var <- "mpg"`, como você pode extrair a variável de referência para um tibble?

Solução. Tanto para o data frame quanto para um tibble, é possível utilizar o operador `[[`.

```
var <- "mpg"
mtcars[[var]]
```

```
## [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

```
as.tibble(mtcars)[[var]]
```

```
## [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4
## [16] 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7
## [31] 15.0 21.4
```

Exercício 7.4.4

Pratique referir-se a nomes de variáveis não sintáticos, no data frame a seguir:

```
annoyng <- tibble(
  `1` = 1:10,
  `2` = `1` * 2 + rnorm(length(`1`))
)
```

- a. Extrair a variável chamada 1.
- b. Plotar um diagrama de dispersão de 1 versus 2.
- c. Criar uma nova coluna chamada 3 que é 2 dividido por 1.
- d. Renomear as colunas para one, two e three.

Solução.

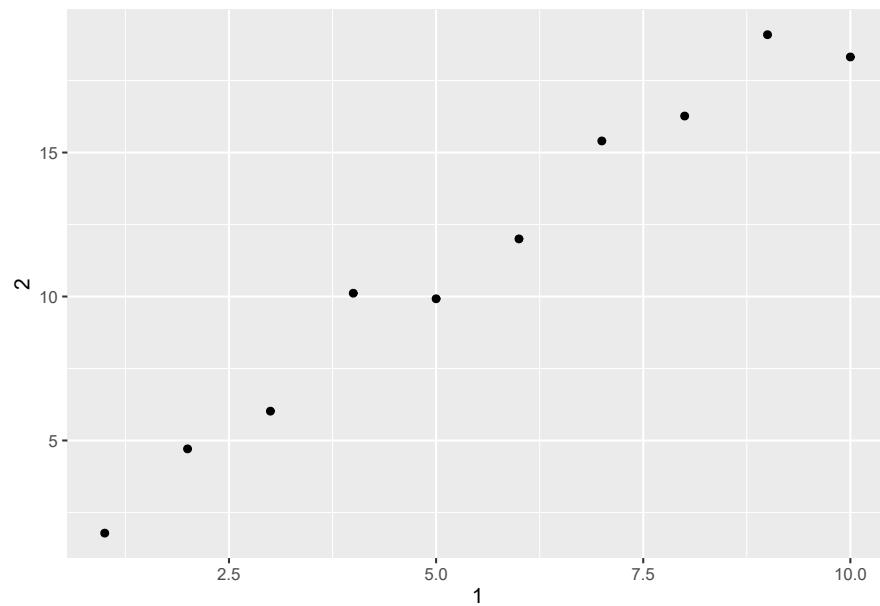
- a. Extrair a variável chamada 1.

```
annoyng$ `1`
```

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

- b. Plotar um diagrama de dispersão de 1 versus 2.

```
annoyng %>%
  ggplot(aes(`1`, `2`)) +
  geom_point()
```



- c. Criar uma nova coluna chamada 3 que é 2 dividido por 1.

```
annoyn$`3` <- annoyn$`2` / annoyn$`1`
```

- d. Renomear as colunas para one, two e three.

```
colnames(annoyn) <- c("one", "two", "three")
annoyn
```

```
## # A tibble: 10 x 3
##       one   two three
##   <int> <dbl> <dbl>
## 1     1  1.78  1.78
## 2     2  4.71  2.35
## 3     3  6.02  2.01
## 4     4 10.1   2.53
## 5     5  9.92  1.98
## 6     6 12.0   2.00
## 7     7 15.4   2.20
## 8     8 16.3   2.03
## 9     9 19.1   2.12
## 10    10 18.3   1.83
```

Exercício 7.4.5

O que `tibble::enframe()` faz? Quando você pode usá-lo?

Solução. Transforma um vetor de valores atômicos ou lista em um tibble de 2 colunas. É útil quando necessitarmos transformar um vetor em um dicionário ou uma lista de pares (nome, valor), por exemplo.

```
x <- letters
enframe(x)
```

```
## # A tibble: 26 x 2
##       name value
##   <int> <chr>
## 1     1 a
## 2     2 b
## 3     3 c
## 4     4 d
```

```
## 5      5 e
## 6      6 f
## 7      7 g
## 8      8 h
## 9      9 i
## 10     10 j
## # i 16 more rows
```

Exercício 7.4.6

Que opção controla quantos nomes de colunas adicionais são impressos no rodapé de um tibble?

Solução. Pode-se usar a opção `tibble.max_extra_cols`.

```
options(tibble.width = 30, tibble.max_extra_cols = 4)
print(as.tibble(mtcars))
```

```
## # A tibble: 32 x 11
##       mpg   cyl  disp    hp
##   <dbl> <dbl> <dbl> <dbl>
## 1     21     6   160   110
## 2     21     6   160   110
## 3    22.8     4   108    93
## 4    21.4     6   258   110
## 5    18.7     8   360   175
## 6    18.1     6   225   105
## 7    14.3     8   360   245
## 8    24.4     4   147.    62
## 9    22.8     4   141.    95
## 10   19.2     6   168.   123
## # i 22 more rows
## # i 7 more variables:
## #   drat <dbl>, wt <dbl>,
## #   qsec <dbl>, vs <dbl>, ...
```



8

Importando dados com readr

8.1 Introdução

Não temos exercícios nesta seção.

8.2 Começando

Exercício 8.2.1

Qual função você usaria para ler um arquivo em que os campos são separados por “|”?

Solução. A melhor opção é utilizar a função `read_delim()` indicando o `|` para o argumento `delim`.

```
read_delim("a | b | c\n1 | 2 | 3", delim = " | ")
```

```
## Rows: 1 Columns: 3
## -- Column specification -----
## 
## Delimiter: " | "
## dbl (3): a, b, c
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## # A tibble: 1 x 3
##       a     b     c
##   <dbl> <dbl> <dbl>
## 1     1     2     3
```

Exercício 8.2.2

Além de `file`, `skip` e `comment`, quais outros argumentos `read_csv()` e `read_tsv()` têm em comum?

Solução. Também são comuns os argumentos: `col_names`, `col_types`, `col_select`, `id`, `locale`, `na`, `quote`, `n_max`, `guess_max`, `progress`, `name_repair`, `num_threads`, `show_col_types`, `skip_empty_rows` e `lazy`.

Exercício 8.2.3

Quais são os argumentos importantes de `read_fwf()`?

Solução. ???

Eu chutaria `n`, `widths`, `start` e `end`.

Exercício 8.2.4

Às vezes, strings em um arquivo CSV contém vírgulas. Para evitar que causem problemas, elas precisam ser cercadas por um caractere de aspas, como " ou '. Por convenção, `read_csv()` supõe que as aspas serão ", e se você quiser mudá-las, precisará usar `read_delim()`. Quais argumentos você precisa especificar para ler o texto a seguir em um data frame?

"x,y\n1, 1, 'a, b'"

Solução. É possível utilizar o argumento `quote` para indicar caractere de citação.

```
read_csv("x, y\n1, 1, 'a, b'", quote = "/")
```

Exercício 8.2.5

Identifique o que há de errado com cada um dos seguintes arquivos CSV em linha. O que acontece quando você executa o código?

```
read_csv("a, b\n1, 2, 3\n4, 5, 6")
read_csv("a, b, c\n1, 2\n1, 2, 3, 4")
read_csv("a, b\n1\"1")
read_csv("a, b\n1, 2\na, b")
read_csv("a;b\n1; 3")
```

Solução. Para o primeiro caso, `read_csv("a, b\n1, 2, 3\n4, 5, 6")` o problema é que a linha de cabeçalhos tem apenas 2 colunas, enquanto as demais tem 3. O correto seria usar:

```
read_csv("a, b, c\n1, 2, 3\n4, 5, 6")  
  
## Rows: 2 Columns: 3  
## -- Column specification -----  
--  
## Delimiter: ","  
## dbl (3): a, b, c  
##  
## i Use `spec()` to retrieve the full column specification for this data.  
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.  
  
## # A tibble: 2 × 3  
##       a     b     c  
##   <dbl> <dbl> <dbl>  
## 1     1     2     3  
## 2     4     5     6
```

Para o segundo item, `read_csv("a, b, c\n1, 2\n1, 2, 3, 4")`, o problema é que há 4 colunas na segunda linha, 2 colunas na segunda e três na linha de cabeçalho. Uma correção possível seria:

```
read_csv("a, b, c, d\n1, 2, . , .\n1, 2, 3, 4", na = c("."))  
  
## Rows: 2 Columns: 4  
## -- Column specification -----  
--  
## Delimiter: ","  
## dbl (4): a, b, c, d  
##  
## i Use `spec()` to retrieve the full column specification for this data.  
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.  
  
## # A tibble: 2 × 4  
##       a     b     c     d  
##   <dbl> <dbl> <dbl> <dbl>  
## 1     1     2     NA     NA  
## 2     1     2     3     4
```

No terceiro exemplo, `read_csv("a, b\n1")`, há dois problemas: não há indicação do valor da variável `b` na segunda linha e há um caracter \ que parece estar perdido na linha. Uma solução possível é:

```
read_csv("a, b\n\"1\"", NA")  
  
## Rows: 1 Columns: 2  
## -- Column specification -----  
---  
## Delimiter: ","  
## dbl (1): a  
## lgl (1): b  
##  
## i Use `spec()` to retrieve the full column specification for this data.  
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.  
  
## # A tibble: 1 × 2  
##       a     b  
##   <dbl> <lgcl>  
## 1     1    NA
```

No caso de `read_csv("a, b\n\"1\", 2\nna, b")`, não parece haver problemas. A menos que a última linha seja uma repetição do cabeçalho.

```
read_csv("a, b\n\"1\", 2\nna, b")  
  
## Rows: 2 Columns: 2  
## -- Column specification -----  
---  
## Delimiter: ","  
## chr (2): a, b  
##  
## i Use `spec()` to retrieve the full column specification for this data.  
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.  
  
## # A tibble: 2 × 2  
##       a     b  
##   <chr> <chr>  
## 1     1     2  
## 2     a     b
```

Para o último exemplo, `read_csv("a;b\n1; 3")`, o erro é que o separador é ; e não ,.

```
read_csv2("a;b\n1; 3")  
  
## i Using ',',',' as decimal and ',.' as grouping mark. Use `read_delim()` for more control.  
  
## Rows: 1 Columns: 2  
## -- Column specification -----  
---  
## Delimiter: ";"  
## dbl (2): a, b  
##  
## i Use `spec()` to retrieve the full column specification for this data.  
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.  
  
## # A tibble: 1 × 2  
##       a     b  
##   <dbl> <dbl>  
## 1     1     3
```

8.3 Analisando um vetor

Exercício 8.3.1

Quais são os argumentos mais importantes de `locale()`?

Solução. Os argumentos mais importantes são `encoding`, `decimal_mark`, `grouping_mark`.

Exercício 8.3.2

O que acontece se você tentar configurar `decimal_mark` e `grouping_mark` com o mesmo caractere? O que acontece com o valor padrão de `grouping_mark` quando você configura `decimal_mark` como “,”? O que acontece com o valor padrão de `decimal_mark` quando você configura `grouping_mark` como “.”?

Solução.

```
locale(grouping_mark = ".")
```

```
## <locale>
## Numbers: 123.456,78
## Formats: %AD / %AT
## Timezone: UTC
## Encoding: UTF-8
## <date_names>
## Days: Sunday (Sun), Monday (Mon), Tuesday (Tue), Wednesday (Wed), Thursday
##         (Thu), Friday (Fri), Saturday (Sat)
## Months: January (Jan), February (Feb), March (Mar), April (Apr), May (May),
##          June (Jun), July (Jul), August (Aug), September (Sep), October
##          (Oct), November (Nov), December (Dec)
## AM/PM: AM/PM
```

- Não é possível configurar `decimal_mark` e `grouping_mark` com o mesmo caractere. Se isto ocorrer, teremos um erro.
- Se usarmos `decimal_mark` como “,”, o valor de `grouping_mark` é automaticamente alterado para “.”;
- Se usarmos “.” para `grouping_mark`, o valor default de `decimal_mark` é alterado automaticamente para “,”;

Exercício 8.3.3

Eu não discuti as opções `date_format` e `time_format` para `locale()`. O que elas fazem? Construa um exemplo que mostre quando elas podem ser úteis.

Solução. Servem para indicar o formato padrão de data e hora, respectivamente.

```
parse_date("20-10-01", locale = locale(date_format = "%y-%m-%d", time_format = "%h:%M:%S"))
```

```
## [1] "2020-10-01"
```

Exercício 8.3.4

Se você mora fora dos Estados Unidos, crie um novo objeto de localização que englobe as configurações para os tipos de arquivo que você mais comumente lê.

Solução.

```
locale(
  decimal_mark = ",",
  grouping_mark = ".",
```

```
date_format = "%d/%m/%Y",
time_format = "%H:%M:%S",
date_names = "pt"
)

## <locale>
## Numbers: 123.456,78
## Formats: %d/%m/%Y / %H:%M:%S
## Timezone: UTC
## Encoding: UTF-8
## <date_names>
## Days: domingo (dom), segunda-feira (seg), terça-feira (ter), quarta-feira
##         (qua), quinta-feira (qui), sexta-feira (sex), sábado (sáb)
## Months: janeiro (jan), fevereiro (fev), março (mar), abril (abr), maio (mai),
##          junho (jun), julho (jul), agosto (ago), setembro (set), outubro
##          (out), novembro (nov), dezembro (dez)
## AM/PM: AM/PM
```

Exercício 8.3.5

Qual a diferença entre `read_csv()` e `read_csv2()`?

Solução. Enquanto `read_csv()` utiliza a vírgula como delimitador, o `read_csv2()` utiliza o ponto-e-vírgula.

Exercício 8.3.6

Quais são as codificações mais usadas na Europa? Quais as codificações mais usadas na Ásia? Procure no Google para descobrir.

Solução. ???

Exercício 8.3.7

Gere uma string de formatação correta para analisar cada uma das datas e horas a seguir:

```
d1 <- "January 1, 2010"
d2 <- "2015-Mar-07"
d3 <- "06-Jun-2016"
```

```
d4 <- c("August 19 (2015)", "July 1 (2015)")  
d5 <- "12/30/14" # Dec 30, 2014  
t1 <- "1705"  
t2 <- "11:15:10.12 PM"
```

Solução.

```
parse_date(d1, "%B %d, %Y")
```

```
## [1] "2010-01-01"
```

```
parse_date(d2, "%Y-%b-%d")
```

```
## [1] "2015-03-07"
```

```
parse_date(d3, "%d-%b-%Y")
```

```
## [1] "2016-06-06"
```

```
parse_date(d4, "%B %d (%Y)")
```

```
## [1] "2015-08-19" "2015-07-01"
```

```
parse_date(d5, "%m/%d/%y")
```

```
## [1] "2014-12-30"
```

```
parse_time(t1, "%H%M")
```

```
## 17:05:00
```

```
parse_time(t2, "%I:%M:%OS %p")
```

```
## 23:15:10.12
```

8.4 Analisando um arquivo

Não temos exercícios nesta seção.

8.5 Escrevendo em um arquivo

Não temos exercícios nesta seção.

8.6 Outros tipos de dados

Não temos exercícios nesta seção.



9

Arrumando dados com tidyverse

9.1 Introdução

Não temos exercícios nesta seção.

9.2 Dados arrumados (Tidy Data)

Exercício 9.2.1

Usando a prosa, descreva como as variáveis e as observações estão organizadas em cada uma das tabelas de exemplo.

Solução. Antes de iniciarmos a discussão, vale ressaltar que as variáveis de interesse são o nome do país, o ano do registro, o total de casos de tuberculose registrados e a população estimada para o ano de registro. Dito isso, vamos analisar cada uma das tabelas.

table1

```
## # A tibble: 6 x 4
##   country     year   cases
##   <chr>       <dbl>  <dbl>
## 1 Afghanistan 1999    745
## 2 Afghanistan 2000   2666
## 3 Brazil      1999  37737
## 4 Brazil      2000  80488
## 5 China       1999 212258
## 6 China       2000 213766
## # i 1 more variable:
## #   population <dbl>
```

Em `table1`, as variáveis estão dispostas nas colunas, as informações nas linhas e os dados nas células da tabela. Este é o formato *tidy*, com os dados organizados de forma clara e consistente com o *tidyverse*.

table2

```
## # A tibble: 12 x 4
##   country   year type  count
##   <chr>     <dbl> <chr> <dbl>
## 1 Afghani~  1999 cases 7.45e2
## 2 Afghani~  1999 popu~ 2.00e7
## 3 Afghani~  2000 cases 2.67e3
## 4 Afghani~  2000 popu~ 2.06e7
## 5 Brazil    1999 cases 3.77e4
## 6 Brazil    1999 popu~ 1.72e8
## 7 Brazil    2000 cases 8.05e4
## 8 Brazil    2000 popu~ 1.75e8
## 9 China     1999 cases 2.12e5
## 10 China    1999 popu~ 1.27e9
## 11 China    2000 cases 2.14e5
## 12 China    2000 popu~ 1.28e9
```

Já em `table2`, as coisas são um pouco diferentes. As variáveis `country` e `year` estão organizadas nas colunas, porém o número de casos e o tamanho da população se encontram distribuídos em mais de uma linha. Em outras palavras, para encontrar uma única observação nesta tabela, é preciso analisar duas linhas: numa delas se encontrará o número total de casos de tuberculose naquele país e ano e, na outra, se encontrará o tamanho da população.

Vamos à `table3`:

table3

```
## # A tibble: 6 x 3
##   country      year rate
##   <chr>        <dbl> <chr>
## 1 Afghanistan 1999 745/19987~
## 2 Afghanistan 2000 2666/2059~
## 3 Brazil       1999 37737/172~
## 4 Brazil       2000 80488/174~
## 5 China        1999 212258/12~
## 6 China        2000 213766/12~
```

Nesta tabela, dois valores estão combinados na coluna `rate`. Apesar de cada linha ter a observação completa, é necessário realizar uma operação para extrair o número de casos e o tamanhpo da população.

Por fim, analisaremos a tabela 4, que é composta por duas tabelas, na verdade:

table4a

```
## # A tibble: 3 × 3
##   country    `1999` `2000`
##   <chr>      <dbl>   <dbl>
## 1 Afghanistan 745     2666
## 2 Brazil       37737   80488
## 3 China        212258  213766
```

Em `table4a` temos o número de casos, sendo que a variável `year` está representada em colunas.

table4b

```
## # A tibble: 3 × 3
##   country    `1999` `2000`
##   <chr>      <dbl>   <dbl>
## 1 Afghanistan 2.00e7  2.06e7
## 2 Brazil       1.72e8  1.75e8
## 3 China        1.27e9  1.28e9
```

Na `table4b`, temos o mesmo problema de distribuição da variável `year`, porém nela encontramos o tamanhpo da população distribuida entre as células.

Exercício 9.2.2

Calcule o `rate` para `table2` e `table4a + table 4b`. Você precisará realizar quatro operações:

- a. Extraia o número de casos de TB por país por ano.
- b. Extraia a população correspondente por país por ano.
- c. Divida os casos por população e multiplique por 10.000.
- d. Armazene no local adequado.

Com qual representação é mais fácil trabalhar? Com qual é mais difícil? Por quê?

Solução. Iniciaremos com a tabela `table2`.

```
header <- table2 %>%
  distinct(country, year)

cases <- table2 %>%
  filter(type == "cases") %>%
  select(count)

population <- table2 %>%
  filter(type == "population") %>%
  select(count)

rates <- (cases$count * 10000) / population$count

new_table2 <- tibble(
  header,
  cases = cases$count,
  population = population$count,
  rate = rates
)

new_table2
```

```
## # A tibble: 6 x 5
##   country     year   cases
##   <chr>      <dbl>  <dbl>
## 1 Afghanistan 1999    745
## 2 Afghanistan 2000   2666
## 3 Brazil       1999  37737
## 4 Brazil       2000  80488
## 5 China        1999 212258
## 6 China        2000 213766
## # i 2 more variables:
## #   population <dbl>,
## #   rate <dbl>
```

Para as tabelas `table4a` e `table4b`, temos o seguinte:

```
(rates <- tibble(
  country = table4a$country,
  `1999` = table4a$`1999` * 10000 / table4b$`1999`,
```

```
`2000` = table4a$`2000` * 10000 / table4b$`2000`  
))  
  
## # A tibble: 3 × 3  
##   country     `1999` `2000`  
##   <chr>       <dbl>  <dbl>  
## 1 Afghanistan 0.373   1.29  
## 2 Brazil      2.19    4.61  
## 3 China       1.67    1.67
```

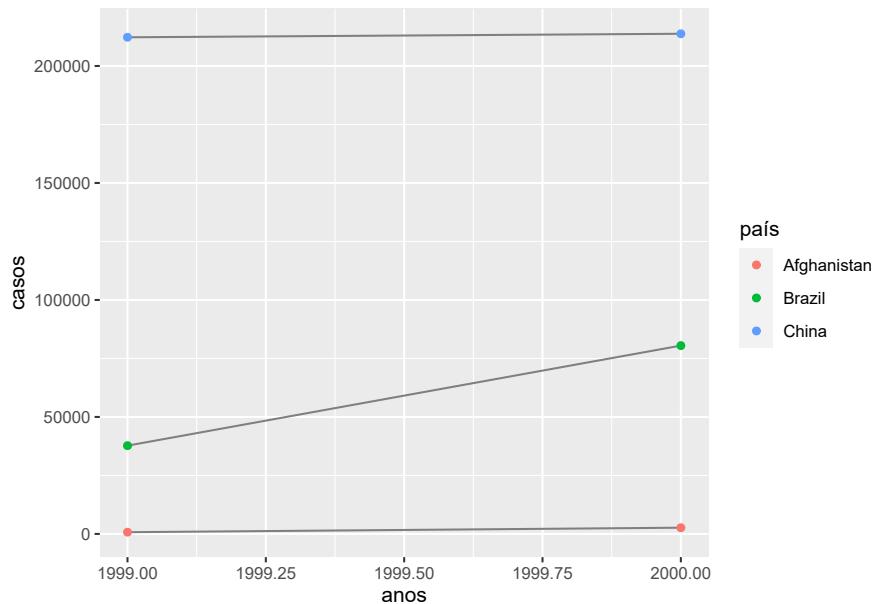
Exercício 9.2.3

Recrie o gráfico mostrando a mudança nos casos com o passar do tempo usando `table2`, em vez de `table1`. O que você precisa fazer primeiro?

Solução.

```
table2 %>%  
  group_by(country, year) %>%  
  filter(type == "cases") %>%  
  select(count) %>%  
  ggplot(aes(year, count)) +  
  geom_line(aes(group = country), color = "gray50") +  
  geom_point(aes(color = country)) +  
  labs(  
    y = "casos",  
    x = "anos",  
    color = "país"  
)
```

```
## Adding missing grouping variables: `country`, `year`
```



9.3 Espalhando e reunindo

Exercício 9.3.1

Por que `gather()` e `spread()` não são perfeitamente simétricos? Considere cuidadosamente o exemplo a seguir:

```
stocks <- tibble(
  year    = c(2015, 2015, 2016, 2016),
  half    = c( 1,      2,      1,      2),
  return  = c(1.88, 0.59, 0.92, 0.17)
)

stocks %>%
  spread(year, return) %>%
  gather("year", "return", `2015`:`2016`)
```

(Dica: observe os tipos de variáveis e pense sobre *nomes* de colunas.)

Ambos `spread()` e `gather()` têm um argumento `convert`. O que ele faz?

Solução. Ao espalhar e, depois, reunir os dados de um mesmo tibble, os tipos podem não ser conservados. No exemplo dado, os valores da variável ano acabam sendo convertidas para string. É possível efetuar uma conversão desses valores usando o parâmetro `convert`. Neste caso, o R buscará inferir o tipo de dado, ainda assim, os tipos podem não coincidir com o original.

Exercício 9.3.2

Por que este código falha?

```
table4a %>%  
  gather(1999, 2000, key = "year", value = "cases")
```

Solução. O método `gather` espera receber uma lista com os nomes das colunas que se deseja juntar. Para tal, é necessário informar ou o nome das colunas ou os seus índices. Como, no caso acima, foram informados números inteiros, a rotina os interpretará como índices das colunas, contudo o tibble não tem essa quantidade de colunas.

A solução para o problema é a seguinte:

```
table4a %>%  
  gather(`1999`, `2000`, key = "year", value = "cases")
```

```
## # A tibble: 6 x 3  
##   country     year   cases  
##   <chr>       <chr>   <dbl>  
## 1 Afghanistan 1999     745  
## 2 Brazil      1999  37737  
## 3 China        1999 212258  
## 4 Afghanistan 2000    2666  
## 5 Brazil      2000  80488  
## 6 China        2000 213766
```

Exercício 9.3.3

Por que espalhar esse tibble falha? Como você poderia adicionar uma nova coluna para corrigir o problema?

```
people <- tribble(  
  ~name,           ~key,     ~value,
```

```
#-----/-----/-----
"Phillip Woods",    "age",     45,
"Phillip Woods",    "height",  186,
"Phillip Woods",    "age",     50,
"Jessica Cordero", "age",     37,
"Jessica Cordero", "height",  156
)
```

Solução. Como existem dois valores de `age` para o registro em nome de Phillip Woods, a função `spread` não pode decidir qual delas utilizar. Para corrigir o problema, seria necessário verificar qual dos dois registros de idade de Phillip é o correto e excluir o outro.

Exercício 9.3.4

Arrume este tibble simples. Você precisará espalhá-lo ou reuní-lo? Quais são as variáveis?

```
preg <- tribble(
  ~pregnant,    ~male,    ~female,
  "yes",        NA,       10,
  "no",         20,       12
)
```

Solução.

```
preg %>%
  gather(male, female, key = "sex", value = "count")
```

```
## # A tibble: 4 x 3
##   pregnant sex   count
##   <chr>     <chr> <dbl>
## 1 yes      male    NA
## 2 no       male    20
## 3 yes      female  10
## 4 no       female  12
```

9.4 Separando e unindo

Exercício 9.4.1

O que os argumentos `extra` e `fill` fazem em `separate()`? Experimente as várias opções para os dois conjuntos de dados de brinquedos a seguir:

```
tibble(x = c("a,b,c", "d,e,f,g", "h,i,j")) %>%
  separate(x, c("one", "two", "three"), extra = "merge")
```

```
## # A tibble: 3 x 3
##   one   two   three
##   <chr> <chr> <chr>
## 1 a     b     c
## 2 d     e     f,g
## 3 h     i     j
```

```
tibble(x = c("a,b,c", "d,e", "f,g,i")) %>%
  separate(x, c("one", "two", "three"), remove = FALSE)
```

```
## Warning: Expected 3 pieces. Missing pieces filled with `NA` in 1 rows [2].
```

```
## # A tibble: 3 x 4
##   x     one   two   three
##   <chr> <chr> <chr> <chr>
## 1 a,b,c a     b     c
## 2 d,e   d     e     <NA>
## 3 f,g,i f     g     i
```

Solução. Durante a separação de um data frame, o parâmetro `extra` indica o que fazer quando houver mais valores excedendo a quantidade de colunas parametrizada. Da mesma forma o argumento `fill` indica o que fazer quando há menos argumentos do que a quantidade de colunas solicitada.

Exercício 9.4.2

`unite()` e `separate()` têm um argumento `remove`. O que ele faz? Por que você o configuraria como `FALSE`?

Solução. O argumento `remove` é utilizado para determinar se os valores originais serão mantidos numa coluna do data set resultante. Podemos usar `remove = FALSE` quando desejarmos manter a configuração original para efeito de comparação e controle.

Exercício 9.4.3

Compare e contraste `separate()` e `extract()`. Por que há três variações de separação (por posição, por separador e com grupos), mas apenas uma para união?

Solução. Enquanto `separate()` usa um separador ou posição de caractere numa string para realizar a separação, `extract()` usa uma expressão regular, o que torna o processo mais dinâmico. Com `extract()` pode-se, por exemplo, tratar variáveis cujos valores tem diferentes tipos de separador.

O objetivo de `separate()` e `extract()` é separar uma coluna em várias. Existem diversas formas pelas quais a coluna a ser separada estará disposta, por isso existem várias formas de realizar a separação. Já no caso de `unite()`, o objetivo é juntar colunas em uma única e, assim, não há muitas opções a serem expressas, não sendo necessárias outras sobreposições.

9.5 Valores faltantes

Exercício 9.5.1

Compare e contraste os argumentos `fill` de `spread()` e `complete()`.

Solução. Em `spread()`, o argumento `fill` é um caractere a ser utilizado quando um valor faltante for encontrado. Já em `complete()`, é possível informar uma lista nomeada indicando qual valor será atribuídos aos valores faltantes em cada uma das colunas.

Exercício 9.5.2

O que o argumento de direção de `fill()` faz?

Solução. O argumento é utilizado para controlar a forma como serão preenchidos os valores faltantes.

A opção padrão é `down`, que preenche os valores faltantes de uma coluna com o valor da linha imediatamente acima. `up` preenche com o valor imediatamente abaixo. `updown` e `downup` parecem ser redundantes. Verificar.

```
treatment <- tribble(  
  ~person, ~treatment, ~response,  
  "Derrick", 1, 7,  
  NA, 2, 10,  
  NA, 3, 9,  
  "Kath", 1, 4,  
  NA, 2, 6,  
  "TESTE", 1, 4,  
)  
  
treatment %>% fill(person, .direction = "up")
```

```
## # A tibble: 6 x 3  
##   person treatment response  
##   <chr>     <dbl>     <dbl>  
## 1 Derrick      1        7  
## 2 Kath         2       10  
## 3 Kath         3        9  
## 4 Kath         1        4  
## 5 TESTE        2        6  
## 6 TESTE        1        4
```

```
treatment %>% fill(person, .direction = "down")
```

```
## # A tibble: 6 x 3  
##   person treatment response  
##   <chr>     <dbl>     <dbl>  
## 1 Derrick      1        7  
## 2 Derrick      2       10  
## 3 Derrick      3        9  
## 4 Kath         1        4  
## 5 Kath         2        6  
## 6 TESTE        1        4
```

```
treatment %>% fill(person, .direction = "downup")
```

```
## # A tibble: 6 x 3  
##   person treatment response
```

```
##   <chr>     <dbl>     <dbl>
## 1 Derrick      1       7
## 2 Derrick      2      10
## 3 Derrick      3       9
## 4 Kath         1       4
## 5 Kath         2       6
## 6 TESTE        1       4
```

```
treatment %>% fill(person, .direction = "updown")
```

```
## # A tibble: 6 x 3
##   person treatment response
##   <chr>     <dbl>     <dbl>
## 1 Derrick      1       7
## 2 Kath         2      10
## 3 Kath         3       9
## 4 Kath         1       4
## 5 TESTE        2       6
## 6 TESTE        1       4
```

9.6 Estudo de caso

Exercício 9.6.1

Neste estudo de caso eu configuro `na.rm = TRUE` só para facilitar a verificação de que tínhamos os valores corretos. Isso é razoável? Pense sobre como os valores faltantes são representados nesse conjunto de dados. Há valores faltantes implícitos? qual é a diferença entre um NAE zero?

Solução.

```
who %>%
  gather(
    new_sp_m014:newrel_f65,
    key = "code",
    value = "cases"
  ) %>%
  mutate(
```

```

code = stringr::str_replace(code, "newrel", "new_rel")
) %>%
separate(code, c("new", "type", "sexage")) %>%
select(-new, -iso2, -iso3) %>%
separate(sexage, c("sex", "age"), sep = 1)

## # A tibble: 405,440 x 6
##   country     year type sex
##   <chr>      <dbl> <chr> <chr>
## 1 Afghanis~  1980 sp    m
## 2 Afghanis~  1981 sp    m
## 3 Afghanis~  1982 sp    m
## 4 Afghanis~  1983 sp    m
## 5 Afghanis~  1984 sp    m
## 6 Afghanis~  1985 sp    m
## 7 Afghanis~  1986 sp    m
## 8 Afghanis~  1987 sp    m
## 9 Afghanis~  1988 sp    m
## 10 Afghanis~ 1989 sp    m
## # i 405,430 more rows
## # i 2 more variables:
## #   age <chr>, cases <dbl>

```

Acima, repetimos o estudo de caso mantendo os valores faltantes explícitos. Para este exemplo, não faria muita diferença prática removê-los ou não, porém em outros cenários pode ser importante manter explícitos os valores faltantes, sobretudo no momento da comunicação dos resultados.

No exemplo utilizado, caso uma observação possua o valor `NA` para a variável `casos`, significa que não estão disponíveis dados sobre TB neste país e ano específico, ou seja, não se sabe se houveram ou não casos de TB. Já o valor zero representa que os dados estavam disponíveis para coleta e é certo que não houve nenhum novo caso de TB naquele ano.

Exercício 9.6.2

O que acontece se você negligenciar o passo `mutate()?` (`mutate(key = stringr::str_replace(key, "newrel", "new_rel"))?`)?

Solução. A operação é importante para o sucesso da análise, pois sem ela ocorre erro durante o passo de separação: é pedido para separar os valores em três colunas (`new`, `type` e `sexage`), porém só é encontrada uma divisão em alguns dos registros, gerando apenas as duas primeiras colunas (a terceira será preenchida com `NA`).

```

who %>%
  gather(
    new_sp_m014:newrel_f65,
    key = "code",
    value = "cases",
    na.rm = TRUE
  ) %>%
  separate(code, c("new", "type", "sexage")) %>%
  select(-new, -iso2, -iso3) %>%
  separate(sexage, c("sex", "age"), sep = 1) %>%
  count(type)

## Warning: Expected 3 pieces. Missing pieces filled with `NA` in 2580 rows [73467, 73468,
## 73469, 73470, 73471, 73472, 73473, 73474, 73475, 73476, 73477, 73478, 73479,
## 73480, 73481, 73482, 73483, 73484, 73485, 73486, ...].

## # A tibble: 17 x 2
##   type     n
##   <chr> <int>
## 1 ep      14304
## 2 f014    190
## 3 f1524   184
## 4 f2534   182
## 5 f3544   183
## 6 f4554   183
## 7 f5564   183
## 8 f65     185
## 9 m014    190
## 10 m1524  182
## 11 m2534  183
## 12 m3544  184
## 13 m4554  184
## 14 m5564  185
## 15 m65    182
## 16 sn     14342
## 17 sp     44820

```

Exercício 9.6.3

Afirmei que `iso2` e `iso3` eram redundantes com `country`. Confirme essa afirmação.

Solução. Para encontrar a resposta, podemos contar os elementos distintos da tripla formada por `country`, `iso2` e `iso3`:

```
who %>%
  select(country, iso2, iso3) %>%
  distinct() %>%
  group_by(country) %>%
  filter(n() > 1)

## # A tibble: 0 x 3
## # Groups:   country [0]
## # i 3 variables:
## #   country <chr>,
## #   iso2 <chr>, iso3 <chr>
```

Exercício 9.6.4

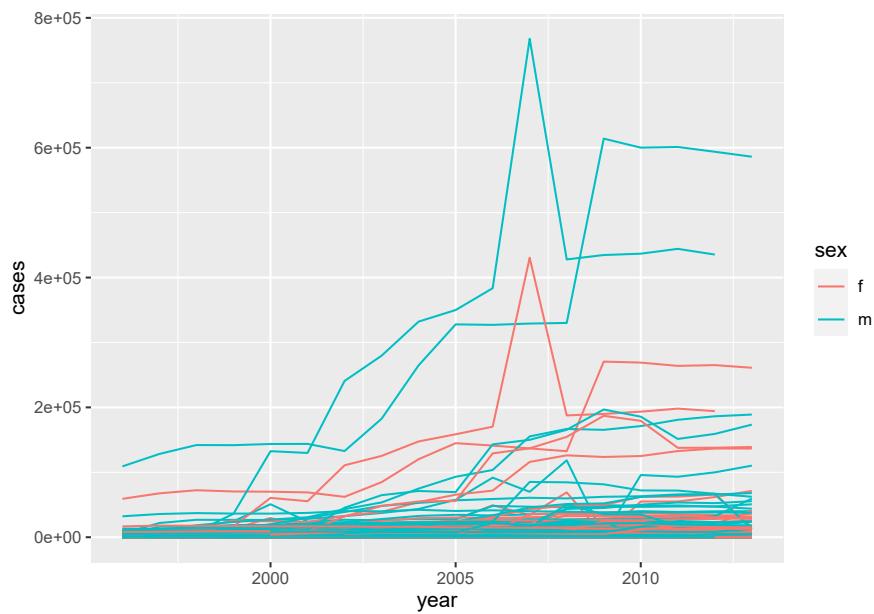
Para cada país, ano e gênero, calcule o número total de casos de TB. Faça uma visualização informativa dos dados.

Solução.

```
who %>%
  filter( year > 1995) %>%
  gather(
    new_sp_m014:newrel_f65,
    key = "code",
    value = "cases",
    na.rm = T
  ) %>%
  mutate(
    code = stringr::str_replace(code, "newrel", "new_rel")
  ) %>%
  separate(code, c("new", "type", "sexage")) %>%
  select(-new, -iso2, -iso3) %>%
  separate(sexage, c("sex", "age"), sep = 1) %>%
  group_by(country, year, sex) %>%
  summarise(
    cases = sum(cases)
  ) %>%
  unite(country_sex, country, sex, remove = F) %>%
  ggplot(aes(
    x = year,
    y = cases,
    group = country_sex,
```

```
    color = sex)
) +
  geom_line()
```

```
## `summarise()` has grouped output by 'country', 'year'. You can override using
## the `groups` argument.
```



9.7 Dados desarrumados (não tidy)

Não temos exercícios nesta seção.

10

Dados relacionais com dplyr

10.1 Introdução

Não temos exercícios nesta seção.

10.2 nycflights13

Exercício 10.2.1

Imagine que você quisesse desenhar (aproximadamente) a rota que cada avião fez da sua origem ao seu destino. De quais variáveis você precisaria? Quais tabelas você precisaria combinar?

Solução. x

Exercício 10.2.2

Eu esqueci de desenhar o relacionamento entre `weather` e `airports`. Qual é o relacionamento e como ele deveria aparecer no diagrama?

Solução. Utilizariamos as tabelas `weather` e `airports` por meio da variável `origin` de `weather`.

Exercício 10.2.3

`weather` só contém informações dos aeroportos de origem (NYC). Se contivesse registro de clima de todos os aeroportos dos Estados Unidos, qual relação adicional definiria com `flights`?

Solução. Haveria uma nova relação entre esses dois conjuntos de dados, utilizando além da data e hora de chegada, a variável `dest`.

Exercício 10.2.4

Nós sabemos que alguns dias do ano são “especiais”, e menos pessoas que o normal viajam nesse período. Como você poderia representar esses dados como um data frame? Quais seriam as chaves primárias dessa tabela? Como ela se conectararia às tabelas existentes?

Solução. Haveria um novo conjunto de dados chamado, digamos, `special_dates`, e a chave seria formada pelas variáveis `year`, `month` e `day`

10.3 Chaves (keys)

Exercício 10.3.1

Adicione uma surrogate key para `flights`.

Solução.

```
(flights1 <- flights %>%
  mutate(id = row_number()))
```

```
## # A tibble: 336,776 x 20
##   year month   day dep_time
##   <int> <int> <int>    <int>
## 1 2013     1     1      517
## 2 2013     1     1      533
## 3 2013     1     1      542
## 4 2013     1     1      544
## 5 2013     1     1      554
## 6 2013     1     1      554
## 7 2013     1     1      555
## 8 2013     1     1      557
## 9 2013     1     1      557
## 10 2013    1     1      558
## # i 336,766 more rows
## # i 16 more variables:
## #   sched_dep_time <int>,
## #   dep_delay <dbl>,
## #   arr_time <int>,
## #   sched_arr_time <int>, ...
```

Exercício 10.3.2

Identifique as keys nos seguintes conjuntos de dados:

- a. Lahman::Batting
- b. babynames::babynames
- c. nasaweather::atmos
- d. fueleconomy::vehicles
- e. ggplot2::diamonds

(Você precisará instalar alguns pacotes e ler algumas documentações.)

Solução. x

Exercício 10.3.3

Desenhe um diagrama ilustrando as conexões entre as tabelas Batting, Master e Salaries no pacote **Lahman**. Desenhe outro diagrama que mostre o relacionamento entre Master, Managers e AwardsManagers.

Como você caracteriza o relacionamento entre as tabelas Batting, Pitching e Fielding?

Solução. x

10.4 Mutating joins**Exercício 10.4.1**

Calcule o atraso médio por destino, depois faça join no data frame `airports` para que possa exibir a distribuição espacial dos atrasos. Eis uma maneira fácil de desenhar um mapa dos Estados Unidos:

```
airports %>%
  semi_join(flights, c("faa" = "dest")) %>%
  ggplot(aes(lon, lat)) +
  borders("state") +
  geom_point() +
  coord_quickmap()
```



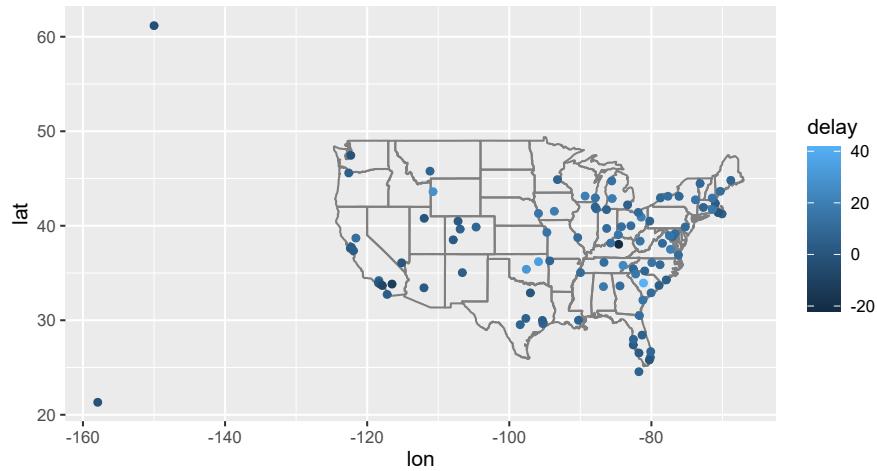
(Não se preocupe se você não entender o que `semi_join()` faz - você aprenderá isso em seguida.)

Você pode querer usar `size` ou `color` dos pontos para exibir o atraso médio de casa aeroporto.

Solução.

```
flights %>%
  group_by(dest) %>%
  summarise(
    delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  left_join(airports, by = c("dest" = "faa")) %>%
  select (name, delay, lat, lon) %>%
  ggplot(aes(
    x = lon,
    y = lat,
    color = delay
  )) +
  borders("state") +
  geom_point() +
  coord_quickmap()
```

```
## Warning: Removed 4 rows containing missing values (`geom_point()`).
```



Exercício 10.4.2

Adicione a localização da origem e destino (isto é, `lat` e `lon`) para `flights`.

Solução.

```
flights %>%
  group_by(dest, origin) %>%
  summarise(
    delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  ungroup() %>%
  left_join(airports, by = c("dest" = "faa")) %>%
  left_join(airports, by = c("origin" = "faa"))
```

```
## `summarise()` has grouped output by 'dest'. You can override using the
## ` `.groups` argument.
```

```
## # A tibble: 224 x 17
##   dest   origin  delay name.x
##   <chr>  <chr>   <dbl> <chr>
## 1 ABQ     JFK     4.38  Albuqu~
```

```

##  2 ACK   JFK    4.85 Nantuc~
##  3 ALB   EWR   14.4 Albany~
##  4 ANC   EWR   -2.5 Ted St~
##  5 ATL   EWR   13.2 Hartsf~
##  6 ATL   JFK    6.27 Hartsf~
##  7 ATL   LGA   11.3 Hartsf~
##  8 AUS   EWR   -0.474 Austin~
##  9 AUS   JFK    10.3 Austin~
## 10 AVL   EWR   8.80 Ashevi~
## # i 214 more rows
## # i 13 more variables:
## #   lat.x <dbl>, lon.x <dbl>,
## #   alt.x <dbl>, tz.x <dbl>,
## #   ...

```

Exercício 10.4.3

Há um relacionamento entre a idade de um avião e seus atrasos?

Solução. Para responder essa questão, precisaremos:

1. calcular o atraso médio de um avião;
2. juntar os dados do avião; e
3. plotar o gráfico de dispersão correspondente.

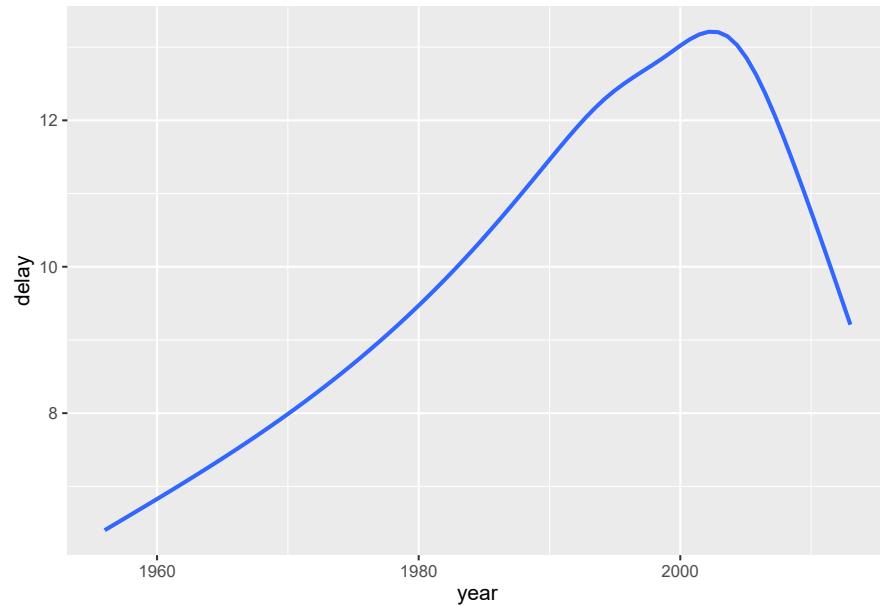
```

flights %>%
  group_by(tailnum) %>%
  summarise(
    delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  filter(delay > 0) %>%
  left_join(
    planes %>%
      select(tailnum, year),
    by = "tailnum"
  ) %>%
  ggplot(aes(year, delay)) +
  geom_smooth(se = FALSE)

## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'

## Warning: Removed 519 rows containing non-finite values (`stat_smooth()`).

```



Não há relação (melhorar resposta!)

Exercício 10.4.4

Quais condições climáticas tornam mais provável haver um atraso?

Solução. Vamos iniciar a nossa análise juntando os conjuntos de dados:

```
(flights_weather <- flights %>%
  inner_join(
    weather,
    by = c(
      "origin" = "origin",
      "year" = "year",
      "month" = "month",
      "day" = "day",
      "hour" = "hour"
    )
  ))
```

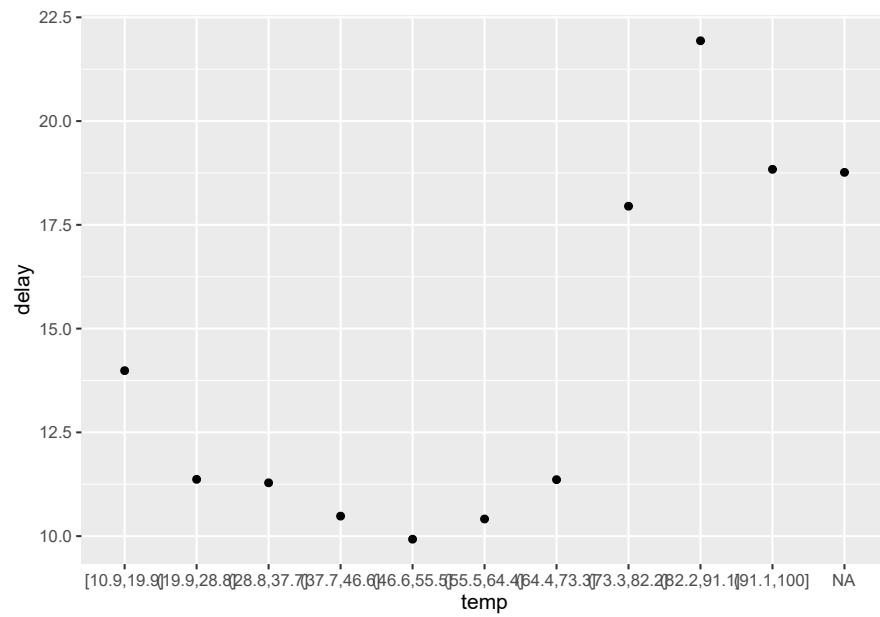
```
## # A tibble: 335,220 x 29
##       year month   day dep_time
##       <int> <int> <int>     <int>
```

```
##   1 2013     1     1    517
##   2 2013     1     1    533
##   3 2013     1     1    542
##   4 2013     1     1    544
##   5 2013     1     1    554
##   6 2013     1     1    554
##   7 2013     1     1    555
##   8 2013     1     1    557
##   9 2013     1     1    557
##  10 2013     1     1    558
## # i 335,210 more rows
## # i 25 more variables:
## #   sched_dep_time <int>,
## #   dep_delay <dbl>,
## #   arr_time <int>,
## #   sched_arr_time <int>, ...
```

Agora vamos avaliar cada uma das condições climáticas antes em relação ao atraso médio na decolagem.

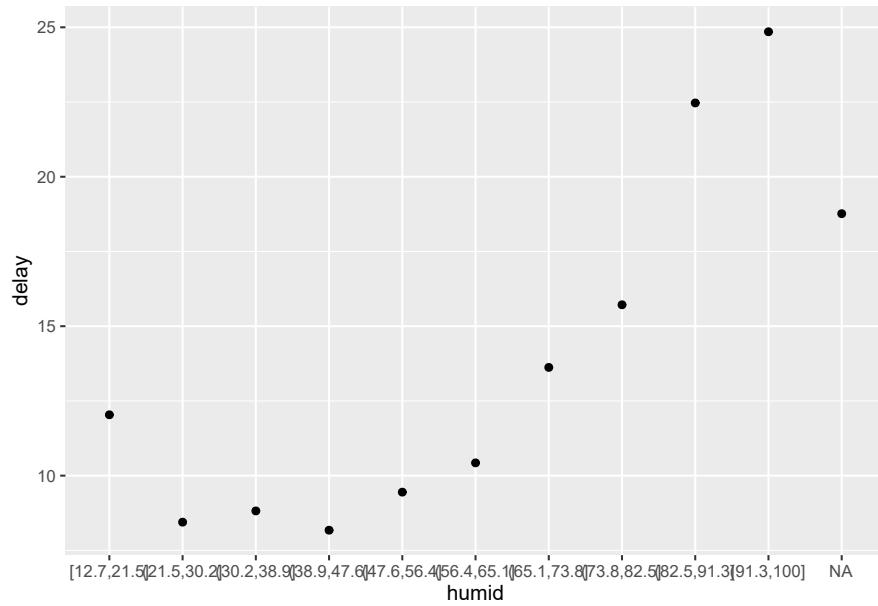
Para a temperatura do ar, não encontramos uma relação, conforme exemplo abaixo.

```
flights_weather %>%
  mutate(temp = cut_interval(temp, n = 10)) %>%
  group_by(temp) %>%
  summarise(delay = mean(dep_delay, na.rm = TRUE)) %>%
  ggplot(aes(temp, delay)) +
  geom_point()
```



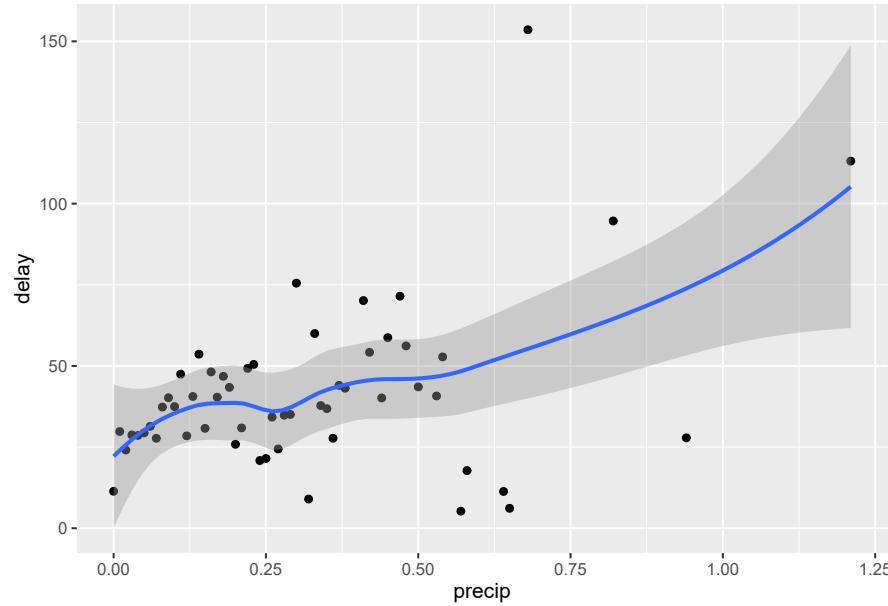
Para a umidade do ar, parece haver uma pequena relação, porém esta pode estar associada a outras variáveis (precip e visib, mais especificamente).

```
flights_weather %>%
  mutate(humid = cut_interval(humid, n = 10)) %>%
  group_by(humid) %>%
  summarise(delay = mean(dep_delay, na.rm = TRUE)) %>%
  ggplot(aes(humid, delay)) +
  geom_point()
```



Para a quantidade de chuva, parece haver uma associação muito fraca.

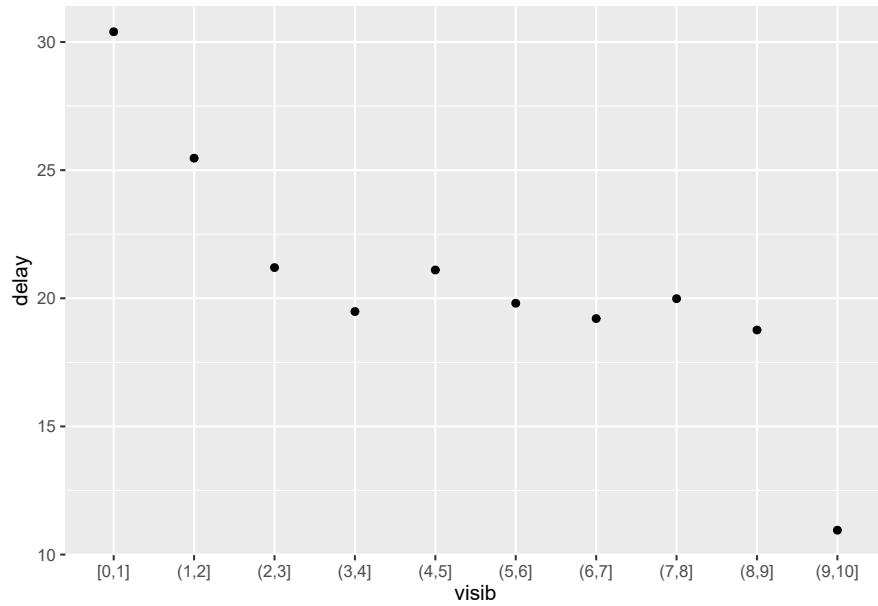
```
flights_weather %>%
  group_by(precip) %>%
  summarise(delay = mean(dep_delay, na.rm = TRUE)) %>%
  ggplot(aes(precip, delay)) +
  geom_point() +
  geom_smooth()  
  
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Para a visibilidade, já podemos perceber uma relação negativa forte.

```
flights_weather %>%
  mutate(visib = cut_interval(visib, n = 10)) %>%
  group_by(visib) %>%
  summarise(delay = mean(dep_delay, na.rm = TRUE)) %>%
  ggplot(aes(visib, delay)) +
  geom_point() +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Exercício 10.4.5

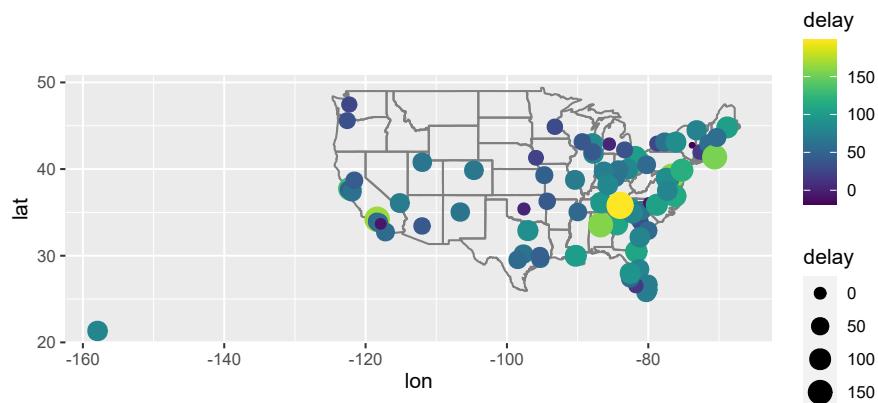
O que aconteceu no dia 13 de junho de 2013? Exiba o padrão espacial de atrasos e, então, uso o Google para fazer uma referência cruzada com o clima.

Solução. Houve grande número de atrasos nos voos para a região sudeste dos Estados Unidos. O fato está diretamente associado a fortes tempestades que ocorreram na região entre os dias 12 e 13 de junho de 2013.

```
flights %>%
  filter(year == 2013, month == 6, day == 13) %>%
  group_by(dest) %>%
  summarise(
    delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  inner_join(airports, by = c("dest" = "faa")) %>%
  select (name, delay, lat, lon) %>%
  ggplot(aes(
    x = lon,
    y = lat,
    color = delay,
    size = delay
  )) +
```

```
borders("state") +  
geom_point() +  
coord_quickmap() +  
scale_colour_viridis()
```

```
## Warning: Removed 3 rows containing missing values (`geom_point()`).
```



10.5 Filtering joins

Exercício 10.5.1

O que significa para um voo ter um `tailnum` faltante? O que os números de cauda que não têm um registro correspondente em `planes` têm em comum? (Dica: uma variável explica aproximadamente 90% dos problemas.)

Solução. Primeiro vamos avaliar o caso de `tailnum` faltante no conjunto `flights`:

```
(flights %>%
  filter(is.na(tailnum)))

## # A tibble: 2,512 x 19
##   year month   day dep_time
##   <int> <int> <int>     <int>
## 1 2013     1     2       NA
## 2 2013     1     2       NA
## 3 2013     1     3       NA
## 4 2013     1     3       NA
## 5 2013     1     4       NA
## 6 2013     1     4       NA
## 7 2013     1     5       NA
## 8 2013     1     7       NA
## 9 2013     1     8       NA
## 10 2013    1     9       NA
## # i 2,502 more rows
## # i 15 more variables:
## #   sched_dep_time <int>,
## #   dep_delay <dbl>,
## #   arr_time <int>,
## #   sched_arr_time <int>, ...
```

```
flights %>%
  filter(is.na(tailnum), !is.na(air_time)) %>%
  nrow()
```

```
## [1] 0
```

Para este conjunto, observamos que `air_time` é sempre faltante, o que indica que o vôo foi cancelado (possivelmente o cancelamento aconteceu antes mesmo de ser designada uma aeronave para executar o voo).

Agora vamos avaliar o voos que não possuem um número de cauda correspondente no conjunto de dados `planes`. Inicialmente, vamos verificar quantos são os voos nessa condição:

```
(flights_without_planes <-
  flights %>%
  anti_join(planes, by = "tailnum")) %>%
  nrow()
```

```
## [1] 52606
```

Há 52.606 vôos sem correspondencia no conjunto de dados `planes`, isso equivale a aproximadamente 15,62% dos vôos. Para avaliar o que eles tem em comum, vamos usar um pouco da nossa intuição: entre as variáveis, as que mais podem ter relação com um código de avião não cadastrado são a companhia que a operou (`carrier`) e o número do voo/percurso (`flight`). Vamos investigá-los!

```
flights_without_planes %>%
  count(carrier, sort = TRUE) %>%
  mutate(p = n / sum(n))
```

```
## # A tibble: 10 x 3
##   carrier     n      p
##   <chr>    <int>  <dbl>
## 1 MQ        25397 0.483
## 2 AA        22558 0.429
## 3 UA        1693  0.0322
## 4 9E        1044  0.0198
## 5 B6         830   0.0158
## 6 US         699   0.0133
## 7 FL         187   0.00355
## 8 DL         110   0.00209
## 9 F9          50   0.000950
## 10 WN         38   0.000722
```

Vimos que aproximadamente 91,16% dos vôos feitos em aeronaves não registradas foram operadas pela American Airlines Inc. ou pela Envoy Air e, avaliando a documentação de `planes`, percebemos que essas companhias não reportam os números de cauda, por isso os valores estão faltantes.

A ausência de registro para os 8,84% restantes não podem ser explicadas a menos que assumamos que se tratam de erro de registro. Por se tratar de um valor muito pequeno em relação ao total de voos (aproximadamente 1,38% do total), entendemos que os mesmos podem ser ignorados.

Exercício 10.5.2

Filtre os voos para exibir apenas aqueles que fizeram pelo menos 100 rotas.

Solução. Em primeiro lugar, precisamos identificar a quantidade de rotas realizadas por cada voo:

```
(flights_over_100 <- flights %>%  
  group_by(tailnum) %>%  
  count(sort = TRUE) %>%  
  filter(!is.na(tailnum), n >= 100))
```

```
## # A tibble: 1,217 x 2  
## # Groups:   tailnum [1,217]  
##   tailnum     n  
##   <chr>    <int>  
## 1 N725MQ     575  
## 2 N722MQ     513  
## 3 N723MQ     507  
## 4 N711MQ     486  
## 5 N713MQ     483  
## 6 N258JB     427  
## 7 N298JB     407  
## 8 N353JB     404  
## 9 N351JB     402  
## 10 N735MQ    396  
## # i 1,207 more rows
```

Agora vamos filtrar o total de voos:

```
flights %>%  
  semi_join(flights_over_100, by = "tailnum")
```

```
## # A tibble: 228,390 x 19  
##   year month   day dep_time  
##   <int> <int> <int>    <int>  
## 1 2013     1     1      517  
## 2 2013     1     1      533  
## 3 2013     1     1      544  
## 4 2013     1     1      554  
## 5 2013     1     1      555  
## 6 2013     1     1      557  
## 7 2013     1     1      557  
## 8 2013     1     1      558  
## 9 2013     1     1      558  
## 10 2013    1     1      558  
## # i 228,380 more rows  
## # i 15 more variables:  
## #   sched_dep_time <int>,
```

```
## #   dep_delay <dbl>,
## #   arr_time <int>,
## #   sched_arr_time <int>, ...
```

Exercício 10.5.3

Combine `fueleconomy::vehicles` e `fueleconomy::common` para encontrar apenas os registros para os modelos mais comuns.

Solução.

```
vehicles %>%
  semi_join(common, by = c("make", "model"))
```

```
## # A tibble: 14,531 x 12
##       id make  model    year
##     <dbl> <chr> <chr>    <dbl>
## 1  1833 Acura Integra  1986
## 2  1834 Acura Integra  1986
## 3  3037 Acura Integra  1987
## 4  3038 Acura Integra  1987
## 5  4183 Acura Integra  1988
## 6  4184 Acura Integra  1988
## 7  5303 Acura Integra  1989
## 8  5304 Acura Integra  1989
## 9  6442 Acura Integra  1990
## 10 6443 Acura Integra  1990
## # i 14,521 more rows
## # i 8 more variables:
## #   class <chr>, trans <chr>,
## #   drive <chr>, cyl <dbl>,
## #   ...
```

Exercício 10.5.4

Encontre as 48 horas (no curso de um ano inteiro) que tiveram os piores atrasos. Faça as referências cruzadas com os dados de `weather`. Você consegue ver algum padrão?

Solução. Em primeiro lugar, vamos buscar as 48 horas com piores atrasos (consideraremos o atraso médio de decolagem dos vôos para cada intervalo de 1h, destes selecionaremos os 48 piores). Para isso:

- iniciaremos separando a hora em que estava planejada a decolagem;

- agrupamos os voos por hora (incluímos a origem no agrupamento porque estamos considerando o atraso na decolagem!);
- calculamos o atraso médio para cada um dos grupos;
- ordenamos conforme o atraso;
- seleciono as 48 piores horas (não contínuas);

```
(most_delayed <- flights %>%
  mutate(hour = sched_dep_time %% 100) %>%
  group_by(origin, year, month, day, hour) %>%
  summarise(dep_delay = mean(dep_delay, na.rm = TRUE)) %>%
  ungroup() %>%
  arrange(desc(dep_delay)) %>%
  slice(1:48))
```

```
## `summarise()` has grouped output by 'origin', 'year', 'month', 'day'. You can
## override using the `.groups` argument.
```

```
## # A tibble: 48 x 6
##   origin year month day
##   <chr>   <int> <int> <int>
## 1 LGA     2013    7     28
## 2 EWR     2013    2      9
## 3 EWR     2013    2      9
## 4 LGA     2013    9      2
## 5 LGA     2013    7     22
## 6 LGA     2013    7     28
## 7 JFK     2013    4     10
## 8 LGA     2013    9     12
## 9 EWR     2013    3      8
## 10 LGA    2013   12      5
## # i 38 more rows
## # i 2 more variables:
## #   hour <dbl>,
## #   dep_delay <dbl>
```

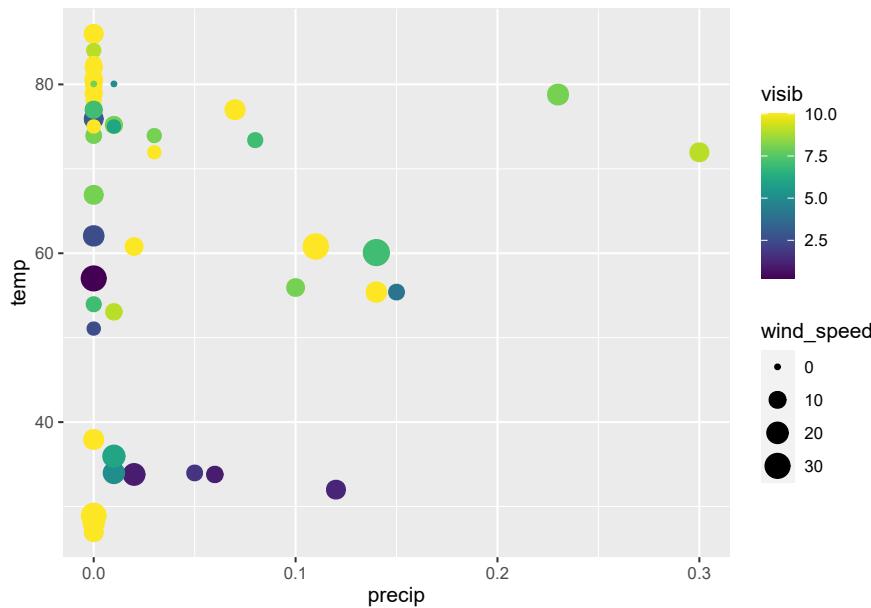
Agora vamos selecionar as condições de tempo nestas horas que selecionamos.

```
(weather_most_delayed <- weather %>%
  semi_join(most_delayed, by = c("origin", "year", "month", "day", "hour"))) %>%
  select (temp, humid, precip, wind_speed, visib)
```

```
## # A tibble: 48 x 5
##       temp humid precip
##       <dbl> <dbl>  <dbl>
## 1    27.0   65.8    0
## 2    28.0   60.1    0
## 3    28.9   57.9    0
## 4    33.8   95.8   0.06
## 5    34.0   96.5   0.05
## 6    80.1   79.2    0
## 7     86    57.1    0
## 8    73.4   94.1   0.08
## 9    84.0   69.6    0
## 10   78.8   93.5   0.23
## # i 38 more rows
## # i 2 more variables:
## #   wind_speed <dbl>,
## #   visib <dbl>
```

Vamos tentar visualizar esses dados de forma mais clara:

```
weather_most_delayed %>%
  ggplot(aes(
    x = precip,
    y = temp,
    color = visib,
    size = wind_speed
  )) +
  geom_point() +
  scale_colour_viridis()
```



Exercício 10.5.5

O que `anti_join(flights, airports, by = c("dest" = "faa"))` lhe diz? O que `anti_join(airports, flights, by = c("faa" = "dest"))` lhe diz?

Solução. O comando `anti_join(flights, airports, by = c("dest" = "faa"))` busca todos os voos cujo destino não está na lista de aeroportos, enquanto `anti_join(airports, flights, by = c("faa" = "dest"))` busca todos os aeroportos que não são destino de nenhum dos voos.

Exercício 10.5.6

Você pode esperar que haja um relacionamento implícito entre avião e linha aérea, visto que cada avião é conduzido por uma única linha aérea. Confirme ou rejeite essa hipótese usando as ferramentas que você aprendeu na seção anterior.

Solução. Não é verdade que uma aeronave é operada apenas por uma única companhia aérea. Temos 17 aeronaves sendo operadas por mais de uma companhia.

```
flights %>%
  filter(!is.na(tailnum)) %>%
  select(tailnum, carrier) %>%
  distinct()
```

```
group_by(tailnum) %>%
  filter(n() > 1) %>%
  left_join(airlines, by = "carrier") %>%
  arrange(tailnum, carrier)
```

```
## # A tibble: 34 x 3
## # Groups:   tailnum [17]
##   tailnum carrier name
##   <chr>    <chr>   <chr>
## 1 N146PQ  9E      Endeavor A~
## 2 N146PQ  EV      ExpressJet~
## 3 N153PQ  9E      Endeavor A~
## 4 N153PQ  EV      ExpressJet~
## 5 N176PQ  9E      Endeavor A~
## 6 N176PQ  EV      ExpressJet~
## 7 N181PQ  9E      Endeavor A~
## 8 N181PQ  EV      ExpressJet~
## 9 N197PQ  9E      Endeavor A~
## 10 N197PQ  EV     ExpressJet~
## # i 24 more rows
```

10.6 Problemas de joins

Não temos exercícios nesta seção.

10.7 Operações de conjuntos

Não temos exercícios nesta seção.



11

Strings com stringr

11.1 Introdução

Não temos exercícios nesta seção.

11.2 O básico de string

Exercício 11.2.1

Em códigos que não usam **stringr**, você verá com frequência, `paste()` e `paste0()`. Qual é a diferença entre as duas funções? Elas são equivalentes a quais funções **stringr**? Como as funções diferem ao lidar com `NA`?

Solução. A diferença entre `paste()` e `paste0()` é que na segunda o separador é sempre a string vazia “”, enquanto na primeira podemos especificar um separador. Ambas as funções são equivalentes à `str_c()`.

As funções `paste()` e `paste0()` transformam `NA` em uma string e o concatena normalmente com as demais strings. Já no caso de `str_c()`, a operação de concatenação com `NA` resulta sempre em `NA`.

Exercício 11.2.2

Em suas próprias palavras, descreva a diferença entre os argumentos `sep` e `collapse` para `str_c()`.

Solução. Quando `str_c()` recebe dois ou mais vetores, ela retorna um novo vetor em que cada posição corresponde à concatenação das posições correspondentes dos vetores de entrada:

```
a <- c("a", "b", "c")
b <- c("1", "2", "3")
str_c(a, b)
```

```
## [1] "a1" "b2" "c3"
```

Caso seja informado um separador, cada posição no vetor resultante, será a combinação das posições equivalentes dos vetores de entrada, separados conforme o parâmetro `sep`:

```
str_c(a, b, sep = " | ")
```

```
## [1] "a | 1" "b | 2" "c | 3"
```

Agora, caso seja informado um valor para `collapse`, a função irá juntar todos os elementos do vetor resultante em uma única string, utilizando o valor de `collapse` para separar os elementos:

```
str_c(a, b, collapse = "|")
```

```
## [1] "a1|b2|c3"
```

Exercício 11.2.3

Use `str_length()` e `str_sub()` para extrair o caractere do meio de uma string. O que você fará se a string tiver um número par de caracteres?

Solução. Para o caso de strings de tamanho ímpar, existe um caractere central e ele será retornado. Para o caso de uma string com comprimento par, traremos os dois caracteres centrais.

```
a <- "Essa é uma string de tamanho ímpar!"
b <- "Já essa é uma string de tamanho par!"

str_central <- function(x) {
  l <- str_length(x)
  q <- l %/% 2
```

```
r <- l %% 2

if (r == 0)
  return(str_sub(x, q, q + 1))
else
  return(str_sub(x, q + 1, q + 1))
}

str_central(a)
```

```
## [1] "
```

```
str_central(b)
```

```
## [1] "in"
```

Exercício 11.2.4

O que `str_wrap()` faz? Quando você pode querer usá-lo?

Solução. Esta função é utilizada para quebrar um texto em linhas (ou parágrafos). Ele pode ser útil quando precisamos imprimir um texto em uma largura específica, como nas impressoras térmicas muito utilizadas ainda hoje.

Exercício 11.2.5

O que `str_trim()` faz? Qual é o oposto de `str_trim()`?

Solução. A função `str_trim()` remove espaços em branco do início e do fim de um texto. O oposto é a função `str_pad()`, que adiciona caracteres a uma string até que ela atinja um determinado tamanho.

Exercício 11.2.6

Escreva uma função que transforme (por exemplo) um vetor `c("a", "b", "c")` na string `a, b and c`. Pense cuidadosamente sobre o que ela deve fazer se lhe for dado um vetor de comprimento 0, 1 ou 2.

Solução. `x`

11.3 Combinando padrões com expressões regulares

Exercício 11.3.1

Explique por que cada uma dessas strings não combina com uma `\: "","\\,"\\\"`.

Solução. x

Exercício 11.3.2

Como você combina a sequência `"'\\"`?

Solução. x

Exercício 11.3.3

Com quais padrões a expressão regular `\...\\..\\..` combinará? Como você a representaria como uma string?

Solução. x

Exercício 11.3.4

Como você combinaria a string literal `"$^$"`?

Solução. x

Exercício 11.3.5

Dado o *corpus* de palavras comuns em `stringr::words`, crie expressões regulares que encontrem todas as palavras que:

- Comecem com “y”.
- Terminem com “x”.
- Tenham exatamente 3 letras de comprimento. (Não trapaceie usando `str_length()!`)
- Tenham sete ou mais letras.

Já que a lista é longa, você pode querer usar o argumento `match` para `str_view()` para exibir apenas as palavras que combinem ou não combinem.

Solução. x

Exercício 11.3.6

Crie expressões regulares para encontrar todas as palavras que:

- a. comecem com uma vogal.
- b. contenham apenas consoantes. (Dica: pense sobre combinar “não” vogais.)
- c. Terminem com ed, mas não com eed.
- d. Terminem com ing ou ize.

Solução. x

Exercício 11.3.7

Verifique empiricamente “i antes de e exceto depois de c”.

Solução. x

Exercício 11.3.8

O “q” é sempre seguido por um “u”?

Solução. x

Exercício 11.3.9

Escreva ma expressão regular que combine uma palavra se ela provavelmente for escrita em inglês britânico, não em inglês norte-americano.

Solução. x

Exercício 11.3.10

Crie uma expressão regular que combinará números de telefone, comumente escritos em seu país.

Solução. x

Exercício 11.3.11

Descreva os equivalentes de ?, +, e * na forma {m, n}.

Solução. x

Exercício 11.3.12

Descreva em palavras o que essas expressões regulares combinam (leia cuidadosamente para verificar se estou usando uma expressão regular ou uma string que define uma expressão regular):

- a. `^.*$`
- b. `"\\{.+\}\\}"`
- c. `\d{4}-\d{2}-\d{2}`
- d. `"\\\\{4}"`

Solução. x

Exercício 11.3.13

Crie expressões regulares para encontrar todas as palavras que:

- a. Comecem com três consoantes.
- b. Tenham três ou mais vogais.
- c. Tenham dois ou mais pares seguidos de vogal-consoante.

Solução. x

Exercício 11.3.14

Resolva a cruzadinha de regexp para iniciantes em <https://regexecrossword.com/challenges/beginner>.

Solução. x

Exercício 11.3.15

Descreva em palavras com o que essas expressões regulares combinarão.

- a. `(.)\1\1`
- b. `"(.)(.)\\2\\1"`
- c. `(..)\1`
- d. `"(..)\\.\\1.\\1"`
- e. `"(..)(.).*\\3\\2\\1"`

Solução. x

Exercício 11.3.16

Construa expressões regulares para combinar palavras que:

- a. comecem e terminem com o mesmo caractere.
- b. Contenham um par de letras repetida (por exemplo, “Church” contém “ch” duas vezes).
- c. Contenham uma letra repetida em pelo menos três lugares (por exemplo, “eleven” contém três “e”).

Solução. x

11.4 Ferramentas**Exercício 11.4.1**

Para cada um dos desafios a seguir, tente resolver a questão usando uma expressão regular e uma combinação de múltiplas chamadas `str_detect()`:

- a. Encontre todas as palavras que comecem ou terminem com x.
- b. Encontre todas as palavras que comecem com uma vogal e terminem com uma consoante.
- c. Há alguma palavra que contenha pelo menos uma de cada uma das cinco vogais diferentes?
- d. Qual palavra tem o maior número de vogais? Qual palavra tem a maior proporção de vogais? (Dica: qual é o denominador?)

Solução. x

Exercício 11.4.2

No exemplo anterior, talvez tenha notado que a expressão regular combinou “flicke-red”, que não é uma cor. Modifique a regex para corrigir o problema.

Solução. x

Exercício 11.4.3

Dos dados das frases de Harvard, extraia:

- a. A primeira palavra de cada frase.

- b. Todas as palavras terminadas em `ing`.
- c. Todos os plurais.

Solução. x

Exercício 11.4.4

Encontre todas as palavras que vêm depois de um “número”, como “one”, “two”, “three” etc. Retire tanto o número, quanto a palavra.

Solução. x

Exercício 11.4.5

Encontre todas as contrações. Separe as partes antes e depois do apóstrofo.

Solução. x

Exercício 11.4.6

Substitua todas as barras em uma string por barras invertidas.

Solução. x

Exercício 11.4.7

Implemente uma versão de `str_to_lower()` usando `replace_all()`.

Solução. x

Exercício 11.4.8

Troque as primeiras e as últimas letras em `words`. Quais dessas strings ainda são palavras?

Solução. x

Exercício 11.4.9

Separe uma string como “apples, pears, and bananas” em componentes individuais.

Solução. x

Exercício 11.4.10

Por que é melhor separar por `boundary("word")` do que " "?

Solução. x

Exercício 11.4.11

Separar por uma string vazia (" ") faz o quê? Experimente, e depois leia a documentação.

Solução. x

11.5 Outros tipos de padrões**Exercício 11.5.1**

Como você encontraria todas as strings contendo `com regex()` versus `com fixed()`?

Solução. x

Exercício 11.5.2

Quais são as cinco palavras mais comuns em `sentences`?

Solução. x

11.6 Outros usos para expressões regulares

Não temos exercícios nesta seção.

11.7 `stringi`

Exercício 11.7.1

Encontre as funções de `stringi` que:

- a. Contem o número de palavras.
- b. Encontrem strings duplicadas.
- c. Gerem texto aleatório.

Solução. x

Exercício 11.7.2

Como você controla a linguagem que `stri_sort()` usa para fazer a classificação?

Solução. x

12

Fatores com forcats

12.1 Introdução

Não temos exercícios nesta seção.

12.2 Criando fatores

Não temos exercícios para esta seção.

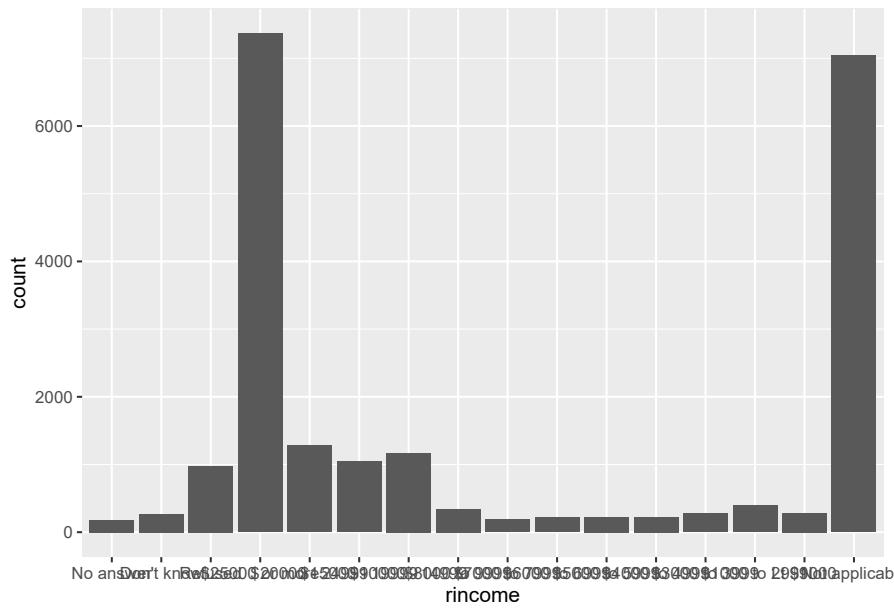
12.3 General Social Survey

Exercício 12.3.1

Explore a distribuição de `rincome` (reported income - renda relatada). O que torna o gráfico de barra padrão tão difícil de entender? Como você melhoraria o gráfico?

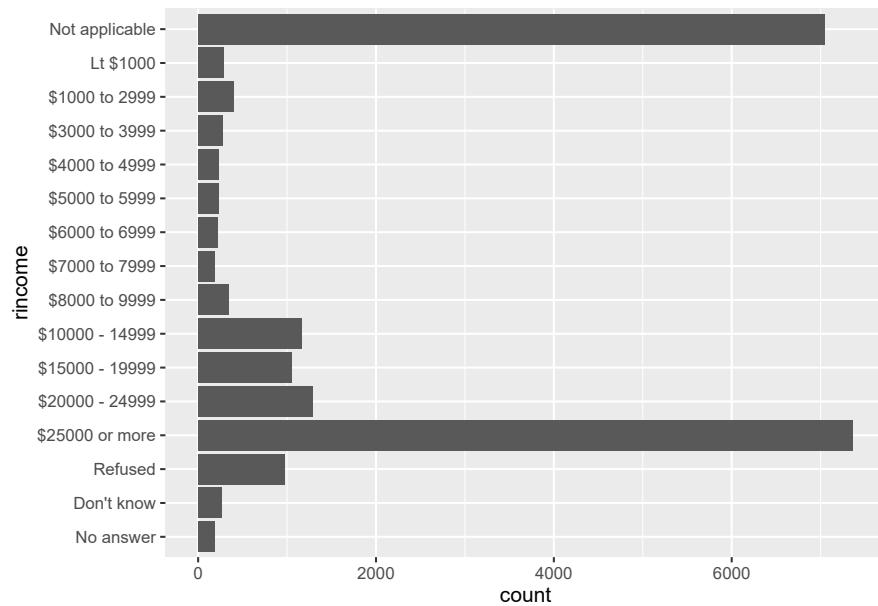
Solução.

```
gss_cat %>%
  ggplot(aes(rincome)) +
  geom_bar()
```



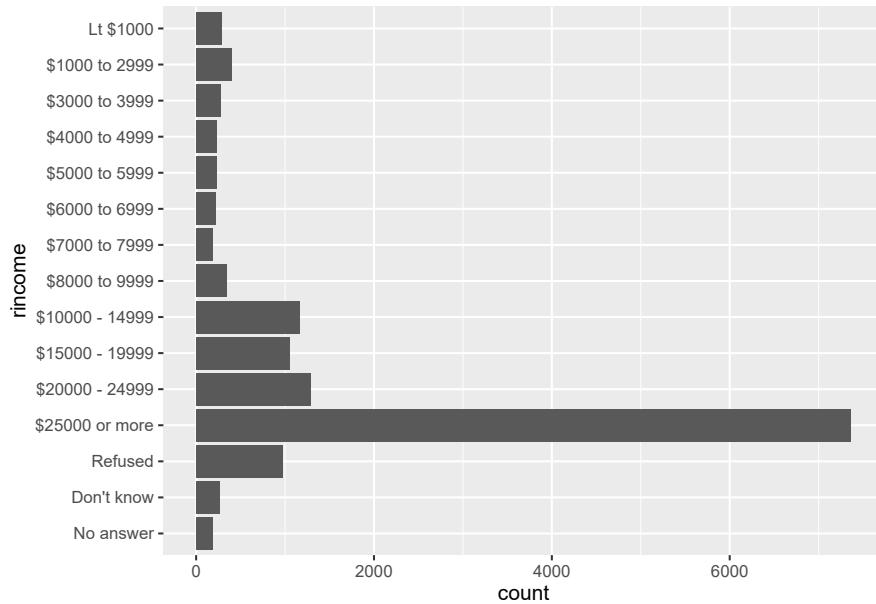
Como temos muitas classes possíveis para `rincome`, a visualização fica comprometida, por isso vamos inverter os eixos do gráfico.

```
gss_cat %>%
  ggplot(aes(rincome)) +
  geom_bar() +
  coord_flip()
```



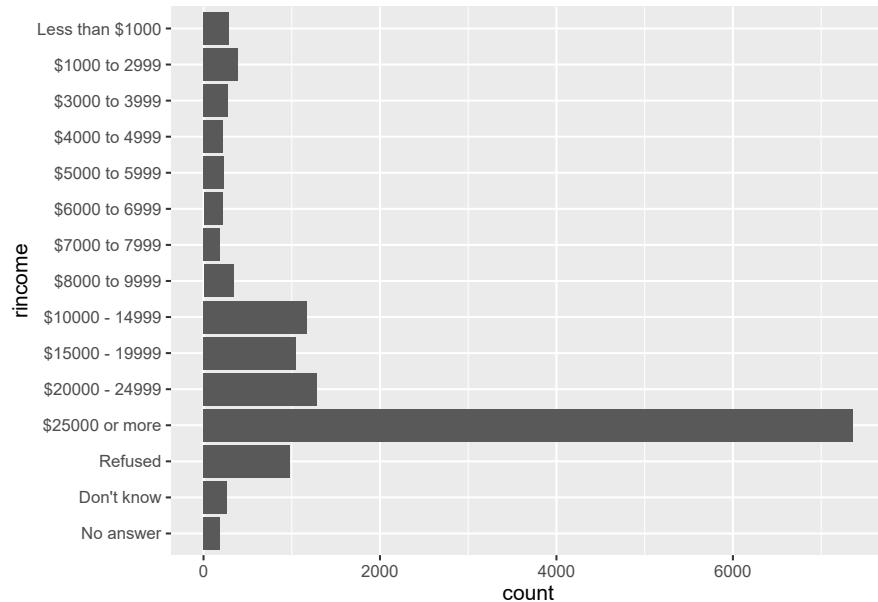
Vamos também remover a categoria `Not applicable`, que não agrupa nenhuma informação o nosso gráfico, uma vez que queremos avaliar apenas as pessoas que possuem uma renda informada.

```
gss_cat %>%
  filter(rincome != "Not applicable") %>%
  ggplot(aes(rincome)) +
  geom_bar() +
  coord_flip()
```



Vamos também substituir o termo LT \$1000 por Less than \$1000.

```
gss_cat %>%
  filter(rincome != "Not applicable") %>%
  mutate(rincome = fct_recode(rincome, "Less than $1000" = "Lt $1000")) %>%
  ggplot(aes(rincome)) +
  geom_bar() +
  coord_flip()
```



Ainda poderíamos pensar em se é possível agrupar algumas das classes para tornar a visualização melhor.

Exercício 12.3.2

Qual é a `relig` mais comum nessa pesquisa? Qual é a `partyid` mais comum?

Solução.

```
gss_cat %>%
  count(relig) %>%
  arrange(desc(n))
```

```
## # A tibble: 15 x 2
##   relig             n
##   <fct>           <int>
## 1 Protestant       10846
## 2 Catholic         5124
## 3 None              3523
## 4 Christian        689
## 5 Jewish             388
## 6 Other              224
## 7 Buddhism            147
## 8 Hindu              100
## 9 Mormon             100
## 10 Buddhist            97
## 11 Buddhist Christian  97
## 12 Buddhist Hindu      97
## 13 Buddhist Mormon     97
## 14 Buddhist None        97
## 15 Buddhist Other        97
```

```
##  8 Inter-nondenominatio~ 109
##  9 Moslem/islam        104
## 10 Orthodox-christian   95
## 11 No answer            93
## 12 Hinduism              71
## 13 Other eastern         32
## 14 Native american       23
## 15 Don't know             15
```

A religião com maior número de adeptos é a Protestante.

```
gss_cat %>%
  count(partyid) %>%
  arrange(desc(n))
```

```
## # A tibble: 10 x 2
##   partyid          n
##   <fct>      <int>
## 1 Independent     4119
## 2 Not str democrat 3690
## 3 Strong democrat 3490
## 4 Not str republican 3032
## 5 Ind,near dem    2499
## 6 Strong republican 2314
## 7 Ind,near rep     1791
## 8 Other party      393
## 9 No answer         154
## 10 Don't know       1
```

E o partido com maior número de afiliados é o Independent.

Exercício 12.3.3

A qual `relig` é aplicada `denom` (denominação)? Como você pode descobrir isso com uma tabela? e com uma visualização?

Solução. Primeiro vamos verificar as categorias possíveis para `relig` e para `denom`:

```
writeLines("relig:")
```

```
## relig:
```

```
levels(gss_cat$relig)
```

```
## [1] "No answer"           "Don't know"
## [3] "Inter-nondenominational" "Native american"
## [5] "Christian"            "Orthodox-christian"
## [7] "Moslem/islam"         "Other eastern"
## [9] "Hinduism"              "Buddhism"
## [11] "Other"                 "None"
## [13] "Jewish"                "Catholic"
## [15] "Protestant"            "Not applicable"
```

```
writeLines("\n\n\ndenom:")
```

```
##  
## denom:
```

```
levels(gss_cat$denom)
```

```
## [1] "No answer"           "Don't know"           "No denomination"
## [4] "Other"                 "Episcopal"             "Presbyterian-dk wh"
## [7] "Presbyterian, merged" "Other presbyterian"   "United pres ch in us"
## [10] "Presbyterian c in us" "Lutheran-dk which"  "Evangelical luth"
## [13] "Other lutheran"       "Wi evan luth synod" "Lutheran-mo synod"
## [16] "Luth ch in america"  "Am lutheran"          "Methodist-dk which"
## [19] "Other methodist"      "United methodist"     "Afr meth ep zion"
## [22] "Afr meth episcopal"  "Baptist-dk which"    "Other baptists"
## [25] "Southern baptist"    "Nat bapt conv usa"  "Nat bapt conv of am"
## [28] "Am bapt ch in usa"   "Am baptist asso"    "Not applicable"
```

Nossa experiência nos diz que as denominações se referem à religião protestante, mas, para validar nossa hipótese, iremos remover os registros com denominação nas categorias Not applicable, No answer, Don't know e No denomination.

```
gss_cat %>%
  filter(!denom %in% c("Not applicable", "No answer", "Don't know", "No denomination")) %>%
  count(relig)
```

```
## # A tibble: 1 x 2
##   relig      n
##   <fct>     <int>
## 1 Protestant 9559
```

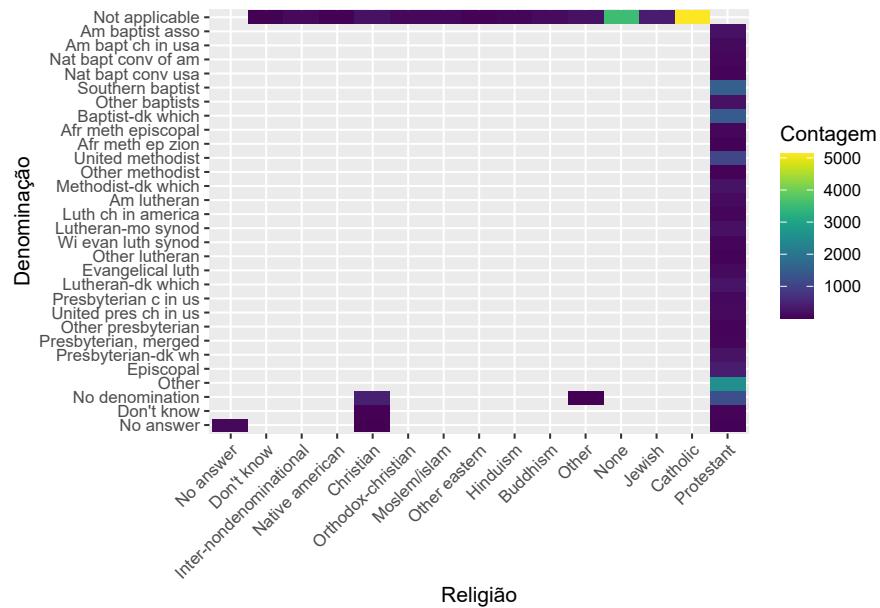
Uma outra possibilidade é:

```
gss_cat %>%
  count(relig, denom)
```

```
## # A tibble: 47 x 3
##   relig      denom     n
##   <fct>     <fct> <int>
## 1 No answer  No a~    93
## 2 Don't know Not ~   15
## 3 Inter-nondenom~ Not ~ 109
## 4 Native american Not ~  23
## 5 Christian    No a~    2
## 6 Christian    Don'~   11
## 7 Christian    No d~   452
## 8 Christian    Not ~  224
## 9 Orthodox-chris~ Not ~  95
## 10 Moslem/islam Not ~ 104
## # i 37 more rows
```

Para a forma gráfica, podemos usar o seguinte:

```
gss_cat %>%
  count(relig, denom) %>%
  ggplot(aes(relig, denom, fill = n)) +
  geom_raster() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(
    x = "Religião",
    y = "Denominação",
    fill = "Contagem"
  ) +
  scale_fill_viridis()
```



12.4 Modificando a ordem dos fatores

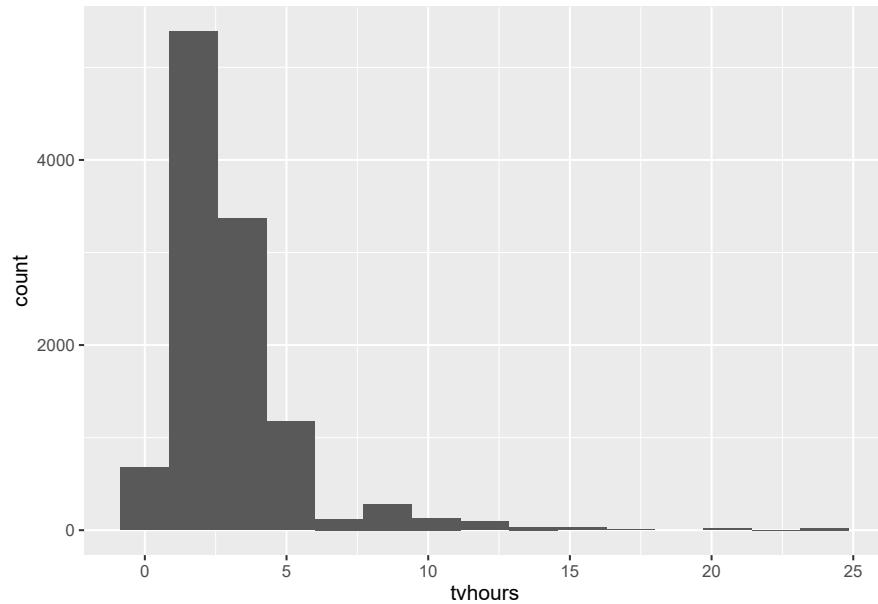
Exercício 12.4.1

Há alguns números suspeitosamente altos em `tvhours`. A média é um bom resumo?

Solução. Inicialmente vamos avaliar a distribuição de `tvhours`.

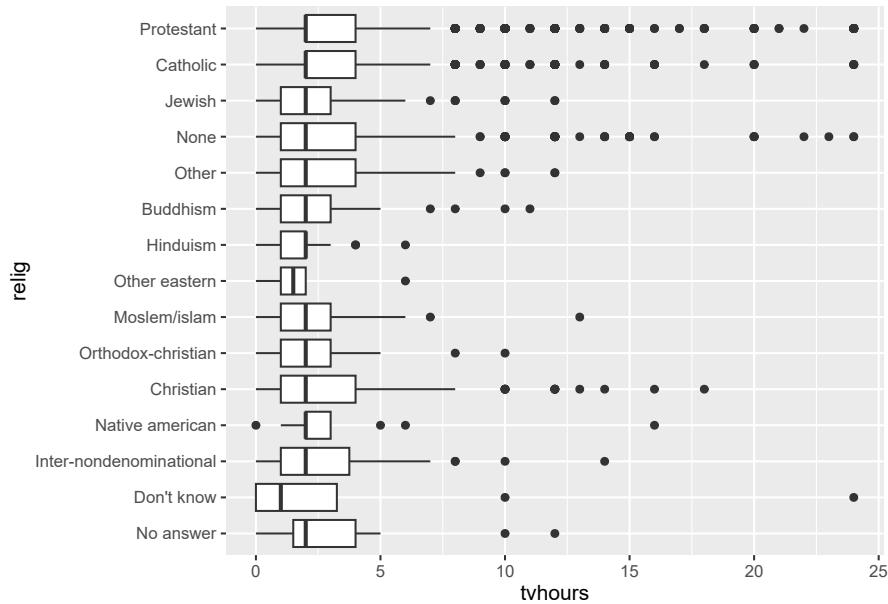
```
gss_cat %>%
  ggplot(aes(tvhours)) +
  geom_histogram(bins = 15)

## Warning: Removed 10146 rows containing non-finite values (`stat_bin()`).
```



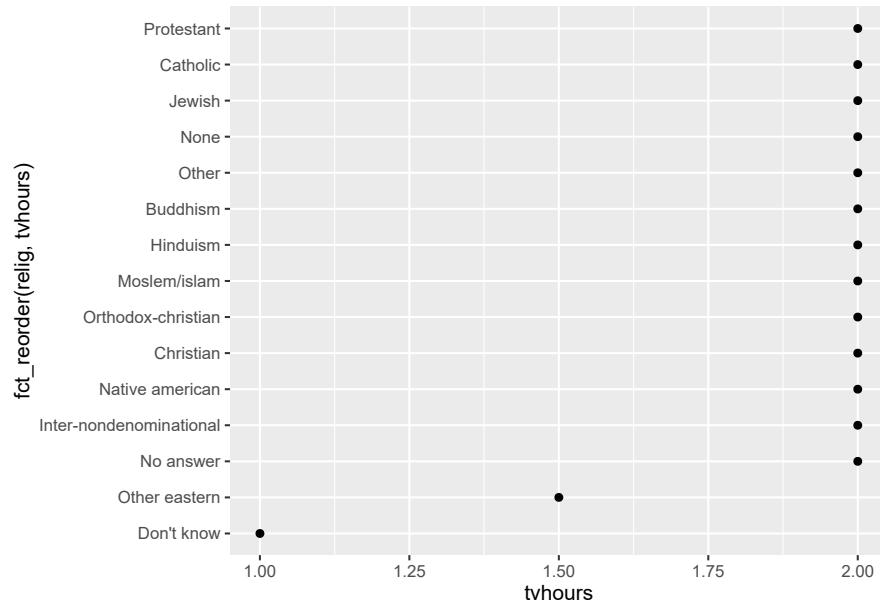
```
gss_cat %>%
  ggplot(aes(tvhours, relig)) +
  geom_boxplot()

## Warning: Removed 10146 rows containing non-finite values (`stat_boxplot()`).
```



De fato, há muitos valores discrepantes, nesse caso, seria melhor utilizarmos a mediana.

```
gss_cat %>%
  group_by(relig) %>%
  summarise(tvhours = median(tvhours, na.rm = TRUE)) %>%
  ggplot(aes(tvhours, fct_reorder(relig, tvhours))) +
  geom_point()
```



Exercício 12.4.2

Para cada fator em gss_cat, identifique se a ordem dos níveis é arbitrária ou com princípios.

Solução. Primeiro vamos visualizar o cabeçalho do conjunto de dados para identificar as colunas que são fatores.

```
gss_cat %>%
  head()

## # A tibble: 6 x 9
##   year marital    age race
##   <int> <fct>     <int> <fct>
## 1 2000 Never mar~    26 White
## 2 2000 Divorced     48 White
## 3 2000 Widowed      67 White
## 4 2000 Never mar~    39 White
## 5 2000 Divorced     25 White
## 6 2000 Married       25 White

## # i 5 more variables:
## #   rincome <fct>,
## #   partyid <fct>,
```

```
## #   relig <fct>, denom <fct>,
## #   ...
```

Agora, vamos listar as categorias em cada fator para avalia se são variáveis ordinais ou não.

```
levels(gss_cat$marital)
```

```
## [1] "No answer"      "Never married"  "Separated"     "Divorced"
## [5] "Widowed"        "Married"
```

```
levels(gss_cat$race)
```

```
## [1] "Other"          "Black"          "White"         "Not applicable"
```

```
levels(gss_cat$rincome)
```

```
## [1] "No answer"      "Don't know"    "Refused"       "$25000 or more"
## [5] "$20000 - 24999"  "$15000 - 19999"  "$10000 - 14999" "$8000 to 9999"
## [9] "$7000 to 7999"   "$6000 to 6999"   "$5000 to 5999"  "$4000 to 4999"
## [13] "$3000 to 3999"   "$1000 to 2999"   "Lt $1000"      "Not applicable"
```

```
levels(gss_cat$partyid)
```

```
## [1] "No answer"      "Don't know"    "Other party"
## [4] "Strong republican" "Not str republican" "Ind,near rep"
## [7] "Independent"    "Ind,near dem"   "Not str democrat"
## [10] "Strong democrat"
```

```
levels(gss_cat$relig)
```

```
## [1] "No answer"           "Don't know"
## [3] "Inter-nondenominational" "Native american"
## [5] "Christian"           "Orthodox-christian"
## [7] "Moslem/islam"        "Other eastern"
## [9] "Hinduism"             "Buddhism"
## [11] "Other"                "None"
## [13] "Jewish"               "Catholic"
## [15] "Protestant"           "Not applicable"
```

```
levels(gss_cat$denom)
```

```
## [1] "No answer"           "Don't know"           "No denomination"
## [4] "Other"                "Episcopal"            "Presbyterian-dk wh"
## [7] "Presbyterian, merged" "Other presbyterian"   "United pres ch in us"
## [10] "Presbyterian c in us" "Lutheran-dk which"  "Evangelical luth"
## [13] "Other lutheran"       "Wi evan luth synod" "Lutheran-mo synod"
## [16] "Luth ch in america"  "Am lutheran"         "Methodist-dk which"
## [19] "Other methodist"      "United methodist"    "Afr meth ep zion"
## [22] "Afr meth episcopal"  "Baptist-dk which"  "Other baptists"
## [25] "Southern baptist"     "Nat bapt conv usa"  "Nat bapt conv of am"
## [28] "Am bapt ch in usa"   "Am baptist asso"   "Not applicable"
```

Apenas `rincome` pode ser ordenada.

Exercício 12.4.3

Por que mover “Not applicable” para a frente dos níveis o move para a parte de baixo do gráfico?

Solução. Porque o gráfico é montado de baixo para cima.

12.5 Modificando níveis de fatores

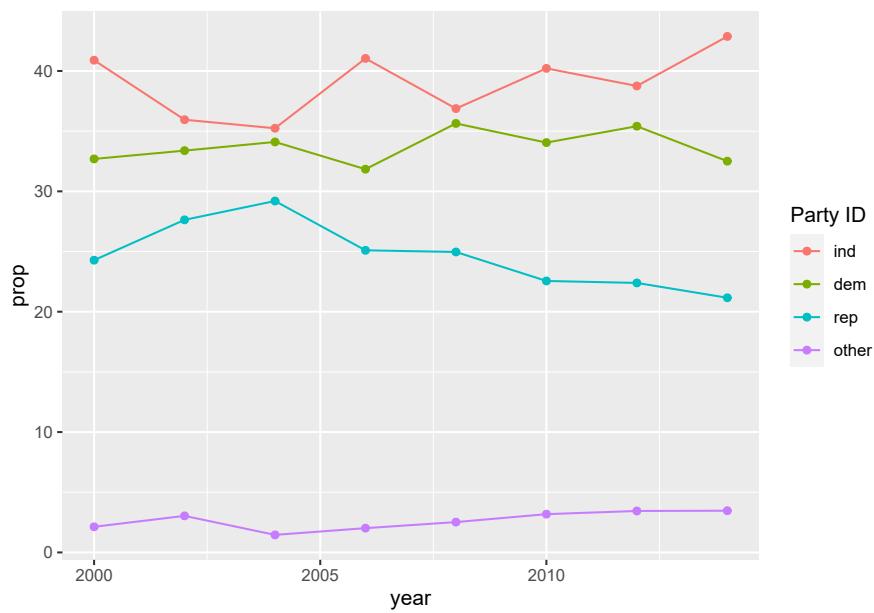
Exercício 12.5.1

Como a proporção de pessoas identificadas como Democratas, Republicanos e Independentes mudou ao longo do tempo?

Solução. Para solucionar este exercício, vamos realizar os seguintes passos:

1. Recodificar os valores do fator `partyid`;
2. Contar os respondentes filiados a cada partido por ano;
3. Calcular as proporções por ano; e
4. Plotar o resultado no gráfico.

```
gss_cat %>%
  mutate(partyid = fct_collapse(partyid,
    other = c("No answer", "Don't know", "Other party"),
    rep = c("Strong republican", "Not str republican"),
    ind = c("Ind,near rep", "Independent", "Ind,near dem"),
    dem = c("Strong democrat", "Not str democrat")))
  ) %>%
  count(year, partyid) %>%
  group_by(year) %>%
  mutate(prop = n * 100 / sum(n)) %>%
  ggplot(aes(
    x = year,
    y = prop,
    color = fct_reorder2(partyid, year, prop)
  )) +
  geom_line() +
  geom_point() +
  labs(color = "Party ID")
```



Exercício 12.5.2

Como você poderia colapsar `rincome` em um conjunto de pequeno de categorias?

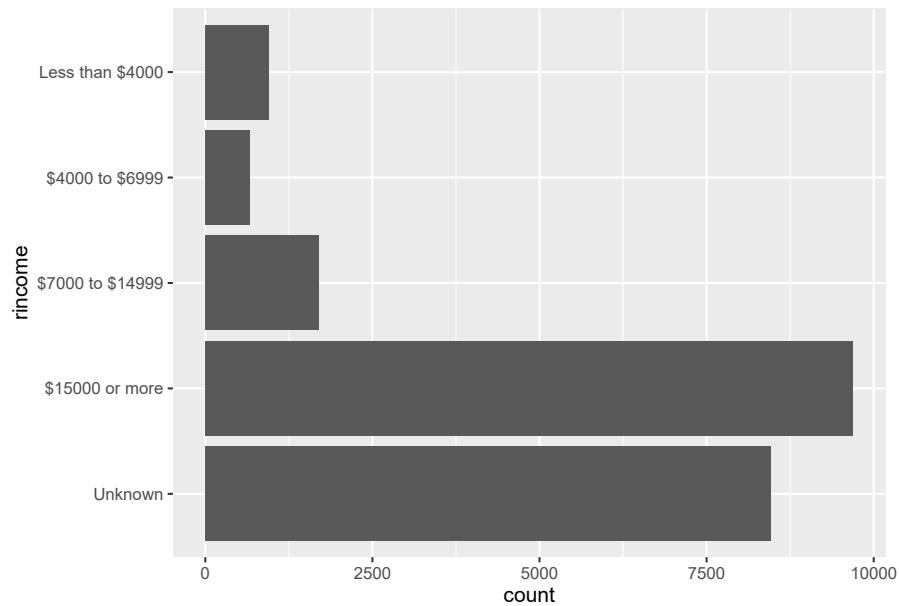
Solução.

```
levels(gss_cat$rincome)
```

```
## [1] "No answer"      "Don't know"     "Refused"       "$25000 or more"
## [5] "$20000 - 24999"  "$15000 - 19999"  "$10000 - 14999" "$8000 to 9999"
## [9] "$7000 to 7999"   "$6000 to 6999"   "$5000 to 5999"  "$4000 to 4999"
## [13] "$3000 to 3999"   "$1000 to 2999"   "Lt $1000"      "Not applicable"
```

```
gss_cat %>%
  mutate(
    rincome = fct_collapse(rincome,
      "Unknown" = c("No answer", "Don't know", "Refused", "Not applicable"),
      "Less than $4000" = c("Lt $1000", str_c("$", c("1000", "3000"), " to ", c("2999", "3999")),
      "$4000 to $6999" = c(str_c("$", c("4000", "5000", "6000"), " to ", c("4999", "5999",
      "$7000 to $14999" = c(str_c("$", c("7000", "8000", "10000"), " to ", c("7999", "9999",
      "$15000 or more" = c("$15000 - 19999", "$20000 - 24999", "$25000 or more")
    )
  ) %>%
  ggplot(aes(rincome)) +
  geom_bar() +
  coord_flip()

## Warning: There was 1 warning in `mutate()` .
## i In argument: `rincome = fct_collapse(...)` .
## Caused by warning:
## ! Unknown levels in `f`: $10000 to 14999
```





13

Datas e horas com lubridate

No decorrer deste capítulo, utilizaremos o seguinte código como base:

```
make_datetime_100 <- function(year, month, day, time) {  
  make_datetime(year, month, day, time %/% 100, time %% 100)  
}  
  
flights_dt <- flights %>%  
  filter(!is.na(dep_time), !is.na(arr_time)) %>%  
  mutate(  
    dep_time = make_datetime_100(year, month, day, dep_time),  
    arr_time = make_datetime_100(year, month, day, arr_time),  
    sched_dep_time = make_datetime_100(year, month, day, sched_dep_time),  
    sched_arr_time = make_datetime_100(year, month, day, sched_arr_time)  
  ) %>%  
  select(origin, dest, ends_with("delay"), ends_with("time"))
```

13.1 Introdução

Não temos exercícios nesta seção.

13.2 Criando data/horas

Exercício 13.2.1

O que acontece se você analisar uma string que contenha datas inválidas?

```
ymd(c("2010-10-10", "bananas"))
```

Solução. Ao tentar avaliar uma string que não corresponda a um formato de data, **lubridate** emite um alerta indicando falha na conversão e substitui a posição correspondente por NA.

```
ymd(c("2010-10-10", "bananas"))
```

```
## Warning: 1 failed to parse.  
## [1] "2010-10-10" NA
```

Exercício 13.2.2

O que o argumento `tzone` para `today()` faz? Por que ele é importante?

Solução. O argumento serve para determinar a timezone. Como em diferentes partes do mundo podemos ter datas diferentes, o valor de `today()` pode ser diferente do esperado.

Exercício 13.2.3

Use a função adequada de **lubridate** para analisar cada uma das datas a seguir:

```
d1 <- "January 1, 2010"  
d2 <- "2015-Mar-07"  
d3 <- "06-Jun-2017"  
d4 <- c("August 19 (2015)", "July 1 (2015)")  
d5 <- "12/30/14" # Dec 30, 2014
```

Solução.

```
mdy(d1)
```

```
## [1] "2010-01-01"
```

```
ymd(d2)
```

```
## [1] "2015-03-07"
```

```
dmy(d3)
```

```
## [1] "2017-06-06"
```

```
mdy(d4)
```

```
## [1] "2015-08-19" "2015-07-01"
```

```
mdy(d5)
```

```
## [1] "2014-12-30"
```

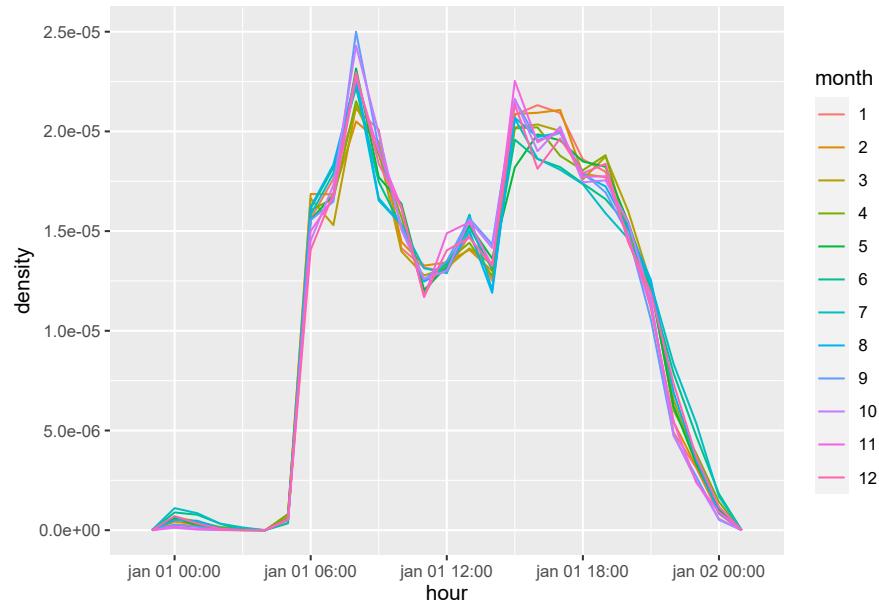
13.3 Componentes de data-hora

Exercício 13.3.1

Como a distribuição dos tempos de voo dentro de um dia mudam ao longo do curso do ano?

Solução. Para avaliar o cenário, vamos recorrer ao data set `flights_dt` e criar duas novas variáveis. A primeira contém apenas a hora do dia (utilizamos a função `mutate()` para levar todas as datas ao primeiro dia do ano) e a segunda conterá o mês. Em seguida, plotaremos o gráfico mostrando a densidade por hora, agrupado pelo mês.

```
flights_dt %>%
  filter(!is.na(dep_time)) %>%
  mutate(hour = update(dep_time, yday = 1)) %>%
  mutate(month = factor(month(dep_time))) %>%
  ggplot(aes(
    x = hour,
    y = ..density..,
    color = month
  )) +
  geom_freqpoly(binwidth = 60 * 60)
```



Notamos que não há grandes variações durante o ano. A quantidade de voos a cada hora é aproximadamente constante durante todo o ano.

Exercício 13.3.2

Compare `dep_time`, `sched_dep_time` e `dep_delay`. São consistentes? Explique suas descobertas.

Solução. Consideraremos os dados consistentes se eles satisfizerem `dep_time = sched_dep_time + dep_delay`.

```
flights_dt %>%
  filter(!is.na(dep_time)) %>%
  mutate(calc_dep_time = sched_dep_time + dep_delay * 60) %>%
  filter(dep_time != calc_dep_time) %>%
  select(dep_time, sched_dep_time, dep_delay, calc_dep_time) %>%
  arrange(desc(dep_delay))
```

```
## # A tibble: 1,205 x 4
##   dep_time
##   <dttm>
## 1 2013-01-09 06:41:00
## 2 2013-06-15 14:32:00
```

```
## 3 2013-01-10 11:21:00
## 4 2013-09-20 11:39:00
## 5 2013-07-22 08:45:00
## 6 2013-04-10 11:00:00
## 7 2013-06-27 09:59:00
## 8 2013-12-05 07:56:00
## 9 2013-05-03 11:33:00
## 10 2013-01-01 08:48:00
## # i 1,195 more rows
## # i 3 more variables:
## #   sched_dep_time <dttm>,
## #   dep_delay <dbl>,
## #   calc_dep_time <dttm>
```

Parece que a variável `dep_time` contém a hora correta, mas não a data. Vamos confirmar essa hipótese movendo todos as horas para o mesmo dia e recalcular a diferença.

```
flights_dt %>%
  filter(!is.na(dep_time)) %>%
  mutate(
    calc_dep_time = sched_dep_time + dep_delay * 60,
    dep_time_2 = update(dep_time, year = 2013, month = 1, day = 1),
    calc_dep_time_2 = update(calc_dep_time, year = 2013, month = 1, day = 1)
  ) %>%
  filter(dep_time_2 != calc_dep_time_2) %>%
  select(dep_time, dep_time_2, calc_dep_time, calc_dep_time_2, dep_delay) %>%
  arrange(desc(dep_delay))
```

```
## # A tibble: 0 x 5
## # i 5 variables:
## #   dep_time <dttm>,
## #   dep_time_2 <dttm>,
## #   calc_dep_time <dttm>,
## #   calc_dep_time_2 <dttm>,
## #   ...
```

De fato, não há nenhum voo cuja hora real de decolagem seja diferente do nosso cálculo. Sendo assim, a única inconsistência é que a data real da decolagem não foi ajustada nos casos em que o atraso fez o voo sair apenas no dia seguinte.

Exercício 13.3.3

Compare `air_time` com a duração entre a partida e a chegada. Explique seus resultados. (Dica: considere a localização do aeroporto.)

Solução. Inicialmente vamos calcular a diferença entre a partida e a chegada.

```
flights_dt %>%
  filter(!is.na(dep_time)) %>%
  mutate(
    calc_dep_time = sched_dep_time + dep_delay * 60,
    flight_duration = arr_time - dep_time,
    diff = flight_duration - air_time
  ) %>%
  select(origin, dest, dep_time, calc_dep_time, arr_time, air_time, flight_duration, diff) %>%
  arrange(diff)

## # A tibble: 328,063 x 8
##   origin dest
##   <chr>  <chr>
## 1 EWR    HNL
## 2 EWR    SFO
## 3 EWR    SFO
## 4 EWR    PHX
## 5 EWR    SEA
## 6 JFK    LAX
## 7 EWR    SAN
## 8 JFK    LAS
## 9 EWR    LAS
## 10 JFK   PDX
## # i 328,053 more rows
## # i 6 more variables:
## #   dep_time <dttm>,
## #   calc_dep_time <dttm>,
## #   arr_time <dttm>,
## #   air_time <dbl>, ...
```

Notamos que existem alguns voos para os quais `flights_duration` é negativa. Esse problema parece estar relacionado à inconsistência apontada no exercício anterior, que ocorre tanto na variável `dep_time`, quanto na variável `arr_time`. Vamos tentar resolver o problema ajustando a data de chegada.

```
flights_dt %>%
  filter(!is.na(dep_time)) %>%
  mutate(
    calc_dep_time = sched_dep_time + dep_delay * 60,
    calc_arr_time = if_else(arr_time <= calc_dep_time, arr_time + (24*60*60), arr_time),
    flight_duration = calc_arr_time - calc_dep_time,
    diff = flight_duration - air_time
  ) %>%
  select(origin, dest, , calc_dep_time, calc_arr_time, air_time, flight_duration, diff) %>%
  arrange(diff)

## # A tibble: 328,063 x 7
##   origin dest
##   <chr>  <chr>
## 1 JFK    HNL
## 2 JFK    HNL
## 3 JFK    HNL
## 4 EWR    HNL
## 5 JFK    HNL
## 6 EWR    HNL
## 7 EWR    HNL
## 8 EWR    HNL
## 9 JFK    HNL
## 10 EWR   HNL
## # i 328,053 more rows
## # i 5 more variables:
## #   calc_dep_time <dttm>,
## #   calc_arr_time <dttm>,
## #   air_time <dbl>,
## #   flight_duration <drtn>,
## #   ...
```

Exercício 13.3.4

Como o tempo médio de atraso muda ao longo do dia? Você deveria usar `dep_time` ou `sched_dep_time`? Por quê?

Solução. Para avaliar este cenário, iremos verificar o atraso nos voos agrupando-os pela hora de decolagem agendada.

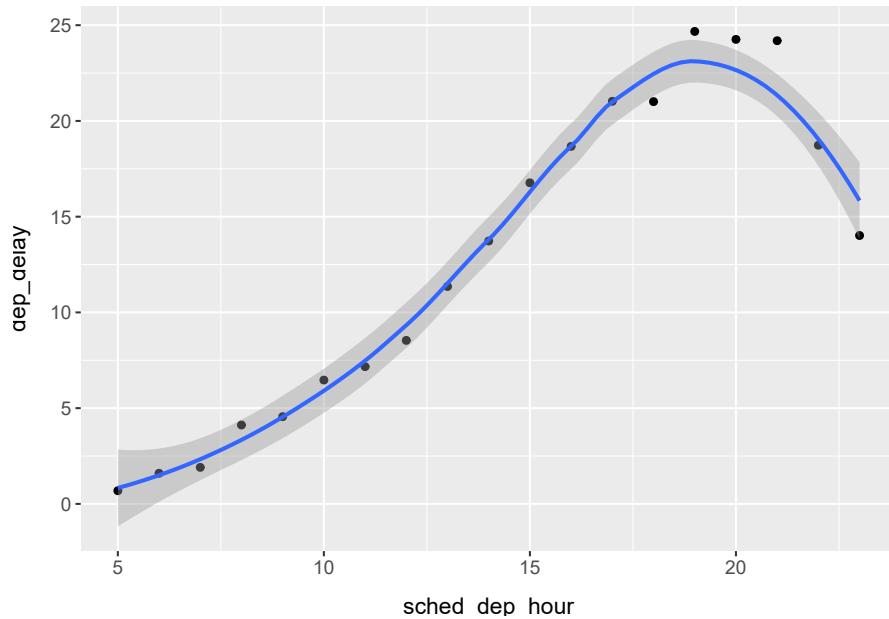
```
flights_dt %>%
  mutate(
```

```

    sched_dep_hour = hour(sched_dep_time)
) %>%
group_by(sched_dep_hour) %>%
summarise(dep_delay = mean(dep_delay)) %>%
ggplot(aes(x = sched_dep_hour, y = dep_delay)) +
  geom_point() +
  geom_smooth() +
  tema

```

`geom_smooth()` using method = 'loess' and formula = 'y ~ x'



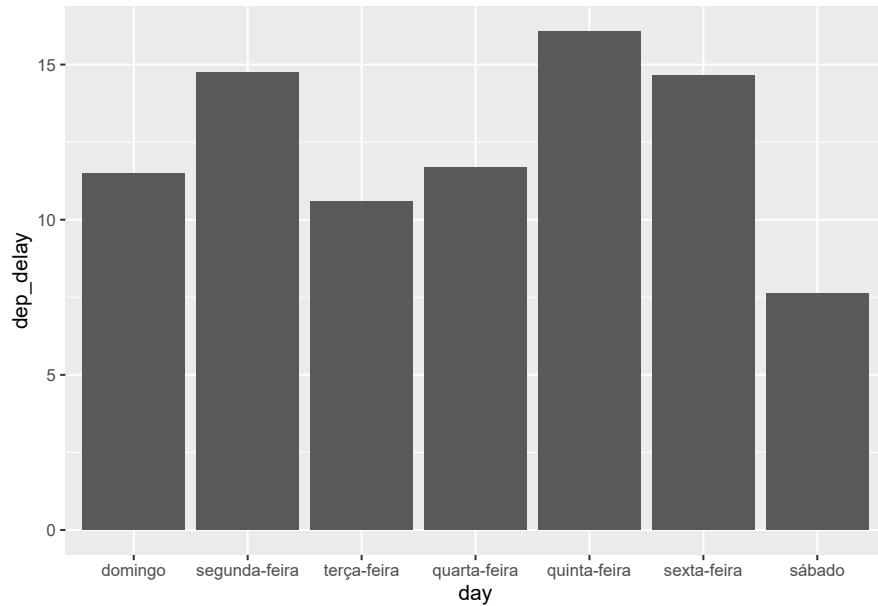
Na média, os maiores atrasos ocorrem entre o final da tarde e início da noite.

Exercício 13.3.5

Em que dia da semana você deve partir se quiser minimizar a chance de uma atraso?

Solução. Para solucionar este caso, vamos identificar o dia da semana de cada voo, realizar o agrupamento e calcular a média em cada grupo.

```
flights_dt %>%
  mutate(day = wday(sched_dep_time, label = TRUE, abbr = FALSE)) %>%
  group_by(day) %>%
  summarise(dep_delay = mean(dep_delay)) %>%
  ggplot(aes(day, dep_delay)) +
  geom_col()
```



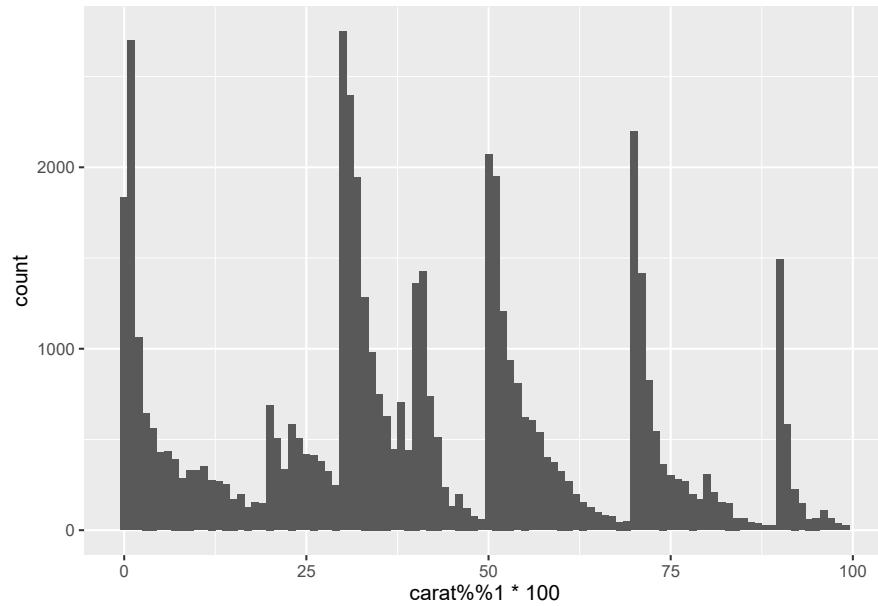
O melhor dia para viajar é no sábado!

Exercício 13.3.6

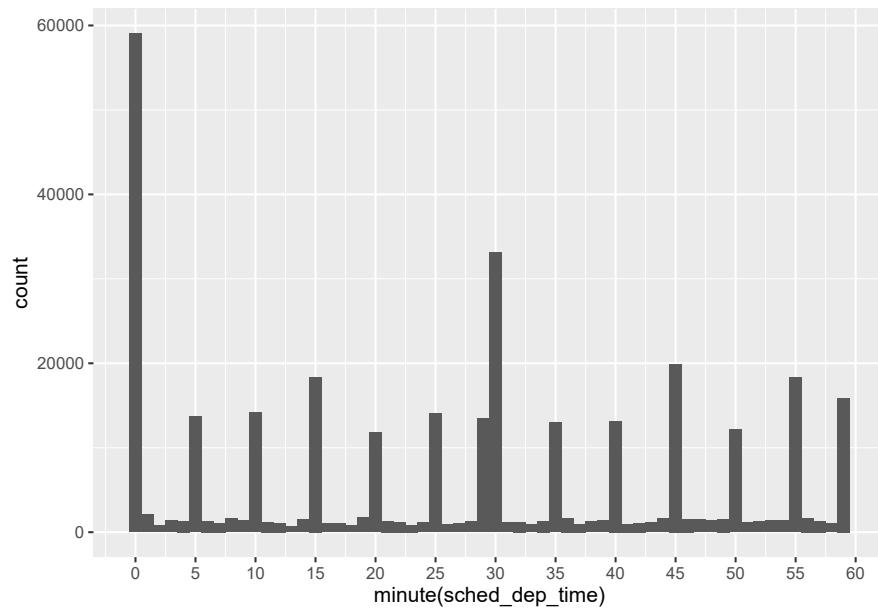
O que torna as distribuições de diamonds\$carat e flights\$sched_dep_time similares?

Solução. Em primeiro lugar iremos plotar ambas as distribuições para avaliá-las melhor.

```
diamonds %>%
  ggplot(aes(x = carat %% 1 * 100)) +
  geom_histogram(binwidth = 1)
```



```
flights_dt %>%
  ggplot(aes(x = minute(sched_dep_time))) +
  geom_histogram(binwidth = 1) +
  scale_x_continuous(breaks = seq(0, 60, 5))
```



Em ambos os casos, percebemos uma concentração de observações em um determinado valor (os picos nos gráficos). Isso se deve ao fato de humanos terem uma predileção por valores arredondados.

Exercício 13.3.7

Confirme minha hipótese de que partidas antecipadas nos minutos 20-30 e 50-60 são causadas por vôos agendados que saem cedo. Dica: crie uma variável binária que lhe diga se houve atraso ou não.

Solução. x

13.4 Intervalos de tempo

Exercício 13.4.1

Por que há `months()` e não `dmonths()`?

Solução. Não faz muito sentido haver uma função `dmonths()` porque há muitas variações. Alguns meses tem 28, 29, 30 ou 31 dias.

Exercício 13.4.2

Explique `days(overnight * 1)` para alguém que acabou de começar a aprender R. Como isso funciona?

Solução. A função `days()` retorna uma quantidade de dias. O argumento recebido é um produto entre `overnight` (que indica se o vôo decola antes da meia noite e chega ao destino depois da meia noite, assumindo os valores 0 ou 1, respectivamente) e 1. Dessa forma, a depender do valor de `overnight`, adicionaremos ou não um dia à data de decolagem/pouso.

Exercício 13.4.3

Crie um vetor de datas dando o primeiro dia de cada mês em 2015. Crie um vetor de datas dando o primeiro dia de cada mês no ano atual.

Solução. Como as funções em **lubridate** podem receber vetores como argumento, vamos utilizar a função `months()` em conjunto com o vetor `0:11`.

```
dmy(01012015) + months(0:11)
```

```
## [1] "2015-01-01" "2015-02-01" "2015-03-01" "2015-04-01" "2015-05-01"  
## [6] "2015-06-01" "2015-07-01" "2015-08-01" "2015-09-01" "2015-10-01"  
## [11] "2015-11-01" "2015-12-01"
```

Já para pegar o primeiro dia de cada mês do ano atual, utilizaremos a mesma técnica, porém usando também as funções `today()` e `floor_date()`:

```
floor_date(today(), unit = "year") + months(0:11)
```

```
## [1] "2024-01-01" "2024-02-01" "2024-03-01" "2024-04-01" "2024-05-01"  
## [6] "2024-06-01" "2024-07-01" "2024-08-01" "2024-09-01" "2024-10-01"  
## [11] "2024-11-01" "2024-12-01"
```

Exercício 13.4.4

Escreva uma função que, dado o seu aniversário (como uma data) retorne quantos anos você tem.

Solução.

```
how_old <- function(birthday) {  
  (dmy(19081990) %--% today()) %/% years(1)  
}  
  
how_old(19081990)
```

```
## [1] 33
```

Exercício 13.4.5

Por que `(today() %--% (today() + years(1)) / months(1))` não funciona?

Solução. Faltava um parentesis. o correto deveria ser:

```
(today() %--% (today() + years(1))) / months(1)
```

```
## [1] 12
```

13.5 Fusos horários

Não temos exercícios para esta seção.



Parte III

Programar



14

Pipes com magrittr

14.1 Introdução

Não temos exercícios para esta seção.

14.2 Alternativas ao piping

Não temos exercícios para esta seção.

14.3 Quando não usar o pipe

Não temos exercícios para esta seção.

14.4 Outras ferramentas do `magrittr`

Não temos exercícios para esta seção.



15

Funções

15.1 Introdução

Não temos exercícios nesta seção.

15.2 Quando você deveria escrever uma função?

Exercício 15.2.1

Por que `TRUE` não é um parâmetro para `rescale01()`? O que aconteceria se `x` contivesse um único valor faltante e `na.rm` fosse `FALSE`?

Solução. Inicialmente, reescrevemos a função `rescale01` mantendo os valores padrão para `na.rm` e `finite`.

```
rescale01 <- function(x) {  
  rng <- range(x)  
  (x - rng[1])/(rng[2] - rng[1])  
}  
  
x <- c(1, 2, 3, NA, 5)  
  
rescale01(x)
```

```
## [1] NA NA NA NA NA
```

Como a maior parte das operações envolvendo um valor faltante resulta também em um valor faltante, a presença de um único `NA` se propagaria por todo o vetor e tornaria a função pouco útil.

Exercício 15.2.2

Na segunda variante de `rescale01()`, valores infinitos não são alterados. Reescreva `rescale01()` para que `-Inf` seja mapeado para 0 e `Inf` seja mapeado para 1.

Solução.

```
rescale01 <- function(x) {
  rng <- range(x, na.rm = TRUE, finite = TRUE)
  y <- (x - rng[1])/(rng[2] - rng[1])

  y[y == -Inf] <- 0
  y[y == Inf] <- 1

  y
}

x <- c(-Inf, 1, 2, 3, 4, 5, Inf, NA)

rescale01(x)
```

```
## [1] 0.00 0.00 0.25 0.50 0.75 1.00 1.00 NA
```

Exercício 15.2.3

Pratique transformar os seguintes fragmentos de código em funções. Pense sobre o que cada função faz. Como você a chamaria, De quantos argumentos precisa? Você consegue reescrevê-la para que seja mais expressiva ou menos duplicativa?

```
mean(is.na(x))

x / sum(x, na.rm = TRUE)

sd(x, na.rm = TRUE) / mean(x, na.rm = TRUE)
```

Solução. No primeiro bloco de código, o objetivo é calcular a proporção de valores faltantes dentro de um vetor.

```
proportion_na <- function(x) {
  mean(is.na(x))
}

proportion_na(c(NA, 1, NA, 3, NA))
```

```
## [1] 0.6
```

No segundo caso, o código padroniza o vetor de modo que a soma seja 1.

```
sum_to_one <- function(x) {
  x / sum(x, na.rm = TRUE)
}

sum_to_one(0:4)
```

```
## [1] 0.0 0.1 0.2 0.3 0.4
```

No terceiro caso, calcula-se o coeficiente de variação.

```
coef_variation <- function(x) {
  sd(x, na.rm = TRUE) / mean(x, na.rm = TRUE)
}

coef_variation(0:5)
```

```
## [1] 0.7483315
```

Exercício 15.2.4

Siga <http://nicercode.github.io/intro/writing-functions.html> para escrever suas próprias funções a fim de calcular a variação e inclinação de um vetor numérico.

Solução. Inicialmente vamos carregar o dataset utilizado no site acima e, em seguida, definiremos a função para cálculo da inclinação.

```
data <- data.frame(
  Height = c(31, 41, 42, 64, 47, 52, 57, 27, 40, 33, 51, 41, 38, 61, 46, 34, 50, 51, 33, 41, 67, 30, 48, 37, 61,
  Weight = c(4.16, 5.82, 3.51, 7.16, 6.17, 5.32, 23.44, 1.76, 4.01, 2.58, 5.98, 2.75, 2.15, 21.59, 6.86, 3.36, 1
)

skewness <- function(x) {
  n <- length(x)
  v <- var(x)
  m <- mean(x)
```

```
    sum((x - m) ^ 3) / ((n - 2) * (v ^ (3/2)))
}
```

```
skewness(data$Height)
```

```
## [1] 0.3011218
```

```
skewness(data$Weight)
```

```
## [1] 1.953767
```

Exercício 15.2.5

Escreva `both_na()` uma função que recebe dois vetores de mesmo comprimento e retorna o número de posições que tem um `NA` em ambos os vetores.

Solução.

```
x <- c(1, 2, 3, NA, 5, NA, 7, 8, NA, 10)
y <- c(1, NA, 3, NA, 5, 6, 7, NA, NA, 10)
```

```
both_na <- function(x, y) {
  sum(is.na(x) & is.na(y))
}
```

```
both_na(x, y)
```

```
## [1] 2
```

Exercício 15.2.6

O que as funções a seguir fazem? Por que são úteis, mesmo embora sejam tão curtas?

```
is_directory <- function(x) file.info(x)$isdir

is_readable <- function(x) file.access(x, 4) == 0
```

Solução. A primeira função verifica se uma string passada como parâmetro corresponde a um diretório, enquanto a segunda verifica se uma string (ou vetor de strings) corresponde a arquivos que podem ser lidos pelo programa.

Elas são úteis quando estamos trabalhando com arquivos dentro do R, por exemplo, para verificarmos a existência e as permissões de acesso a um arquivo selecionado pelo usuário. Embora sejam funções curtas, elas simplificam as verificações e tornam o código mais legível. É muito mais entendível termos uma chamada a `is_readable()` do que sabermos o que significa `file.access(x, 4) == 0`.

Exercício 15.2.7

Leia a letra completa (<https://bit.ly/littlebunnyfoofoo>) de “Little Bunny Foo Foo”. Há bastante duplicação nessa música. Estenda o exemplo de piping inicial para recriar a música completa, e use funções para reduzir a duplicação.

Solução. x

15.3 Funções são para humanos e computadores

Exercício 15.3.1

Leia o código-fonte para cada uma das três funções a seguir, descubra o que elas fazem e então dê ideias de nomes melhores.

```
f1 <- function(string, prefix) {  
  substr(string, 1, nchar(prefix)) == prefix  
}  
  
f2 <- function(x) {  
  if (length(x) <= 1 ) return(NULL)  
  x[ -length(x)]  
}  
  
f3 <- function(x, y) {  
  rep(y, length.out = length(x))  
}
```

Solução. A primeira função tem como objetivo verificar se uma determinada string começa com o prefixo informado. Sugerimos que a função seja nomeada como `str_starts_with`.

```
str_starts_with <- function(string, prefix) {
  substr(string, 1, nchar(prefix)) == prefix
}
```

A função `f2` remove de um vetor o seu último elemento. Sugerimos renomeá-la `vct_remove_last`.

```
vct_remove_last <- function(x) {
  if (length(x) <= 1) return(NULL)
  x[-length(x)]
}
```

A última função tem como objetivo expandir um vetor até que ele atinja o tamanho de um segundo. Por esta razão, faz sentido chamar a função de `vct_expand`.

```
vct_expand <- function(x, y) {
  rep(y, length.out = length(x))
}
```

Exercício 15.3.2

Pegue uma função que você tenha escrito recentemente e passe cinco minutos pensando em nomes melhores para ela e seus argumentos.

Solução. `x`

Exercício 15.3.3

Compare e contraste `rnorm()` e `MASS::mvrnorm()`. Como você poderia torná-las mais consistentes?

Solução. `x`

Exercício 15.3.4

Construa um argumento sobre porque `norm_r()`, `norm_d()`, etc. seriam melhores do que `rnorm()`, `dnorm()`. Construa um argumento para o caso oposto.

Solução. `x`

15.4 Execução condicional

Exercício 15.4.1

Qual é a diferença entre `if` e `ifelse()`? Leia cuidadosamente a ajuda e construa três exemplos que ilustrem as principais diferenças.

Solução. Enquanto `if` é um bloco de controle de fluxo da linguagem, `ifelse` é uma função constuída em R. A instrução `if` avalia uma condição e desvia o fluxo de execução para o bloco de código correspondente. Já a função `ifelse` avalia um vetor de condições e retorna um valor dos vetores `yes` e `no`, conforme a posição da condições.

Por exemplo, suponha que queiramos classificar um conjunto de valores conforme a paridade.

```
x <- 0:10
```

Caso tentemos avaliar todos os valores do vetor de uma única vez com a instrução `if`, receberemos uma mensagem de erro.

```
if(x %% 2 == 0) {  
  y <- "PAR"  
} else {  
  y <- "IMPAR"  
}
```

```
## Error in if (x%%2 == 0) {: the condition has length > 1
```

Precisaríamos utilizar outro bloco de controle para avaliar cada posição do vetor individualmente:

```
for (i in 1:length(x)) {  
  if (x[i] %% 2 == 0) {  
    y[i] <- "PAR"  
  } else {  
    y[i] <- "IMPAR"  
  }  
}  
  
y
```

```
## [1] "PAR"    "IMPAR"  "PAR"    "IMPAR"  "PAR"    "IMPAR"  "PAR"    "IMPAR"  "PAR"
## [10] "IMPAR" "PAR"
```

Utilizando a função `ifelse`, temos um código muito mais enxuto:

```
(y <- ifelse(x %% 2 == 0, "PAR", "IMPAR"))
```

```
## [1] "PAR"    "IMPAR"  "PAR"    "IMPAR"  "PAR"    "IMPAR"  "PAR"    "IMPAR"  "PAR"
## [10] "IMPAR" "PAR"
```

Exercício 15.4.2

Escreva uma função que diga “good morning”, “good afternoon”, ou “good evening”, dependendo da hora do dia. (Dica: use um argumento de tempo padrão de `lubridate::now()`. Isso facilitará testar sua função.)

Solução.

```
greet <- function(time = lubridate::now()) {
  h <- lubridate::hour(time)

  if(h >= 6 && h < 12) {
    print("Bom dia!")
  } else if(h >= 12 && h < 18) {
    print("Boa tarde!")
  } else {
    print("Boa noite!")
  }
}
```

```
greet()
```

```
## [1] "Boa tarde!"
```

```
greet(ymd_hm("2023-12-31 00:00"))
```

```
## [1] "Boa noite!"
```

```
greet(ymd_hm("2023-12-31 06:00"))
```

```
## [1] "Bom dia!"
```

```
greet(ymd_hm("2023-12-31 11:59"))
```

```
## [1] "Bom dia!"
```

```
greet(ymd_hm("2023-12-31 12:00"))
```

```
## [1] "Boa tarde!"
```

```
greet(ymd_hm("2023-12-31 17:59"))
```

```
## [1] "Boa tarde!"
```

```
greet(ymd_hm("2023-12-31 18:00"))
```

```
## [1] "Boa noite!"
```

```
greet(ymd_hm("2023-12-31 23:59"))
```

```
## [1] "Boa noite!"
```

Exercício 15.4.3

Implemente um função `fizzbuzz`. Ela recebe um único número como entrada. Se o número for divisível por três, retorna um “fizz”. Se for divisível por cinco, retorna um “buzz”. Se for divisível por três e por cinco, retorna um “fizzbuzz”. Caso contrário, retorna o número. Certifique-se de escrever o código antes de criar a função.

Solução.

```
fizzbuzz <- function(x) {  
  result <- ""  
  
  if(x %% 3 == 0)  
    result <- "fizz"  
  
  if(x %% 5 == 0)  
    result <- str_c(result, "buzz")  
  
  if (str_length(result) == 0)  
    result <- x  
  
  result  
}  
  
fizzbuzz(3)
```

```
## [1] "fizz"
```

```
fizzbuzz(5)
```

```
## [1] "buzz"
```

```
fizzbuzz(15)
```

```
## [1] "fizzbuzz"
```

```
fizzbuzz(13)
```

```
## [1] 13
```

Exercício 15.4.4

Como você poderia usar `cut()` para simplificar esse conjunto de declarações if-else agrupadas?

```
if(temp <= 0) {  
  "freezing"  
} else if(temp <= 10) {  
  "cold"  
} else if(temp <= 20) {  
  "cool"  
} else if(temp <= 30) {  
  "warm"  
} else {  
  "hot"  
}
```

Como você mudaria a chamada de `cut()` se eu usasse `<`, em vez de `<=`? Qual é a outra vantagem principal de `cut()` para esse problema? (Dica: o que acontece se você tem muitos valores em `temp`?)

Solução. Inicialmente vamos verificar na ajuda que a função `cut()` visa classificar um vetor `x` conforme os seus valores caem em intervalos definidos por um vetor `y`, atribuindo *labels* aos valores de `x`.

Desta forma, podemos resumir o código acima no seguinte:

```
temp <- c(-10, 5, 4, 15, 20, 30, 50)  
  
cut(  
  x = temp,  
  breaks = c(-Inf, 0, 10, 20, 30, Inf),  
  labels = c("freezing", "cold", "cool", "warm", "hot"),  
  right = TRUE  
)  
  
## [1] freezing cold     cool     cool     warm     hot  
## Levels: freezing cold cool warm hot
```

Caso tivesse sido utilizado `<` em vez de `<=`, bastaria definirmos como `FALSE` o argumento `right` da função `cut()`.

```
cut(  
  x = temp,  
  breaks = c(-Inf, 0, 10, 20, 30, Inf),  
  labels = c("freezing", "cold", "cool", "warm", "hot"),  
  right = FALSE  
)
```

```
## [1] freezing cold     cool      warm     hot      hot
## Levels: freezing cold cool warm hot
```

Por fim, a vantagem de usarmos `cut()` é que ele é otimizado para trabalhar com vetores, enquanto `if` pode avaliar apenas valores únicos.

Exercício 15.4.5

O que ocorre se você usar `switch()` com valores numéricos? O que esta chamada de `switch()` faz? O que acontece de `x` for “e”?

```
switch(x,
  a = ,
  b = "ab",
  c = ,
  d = "cd"
)
```

Solução. Se passarmos um valor numérico para `switch()`, ele será interpretado como uma posição e será retornada o valor correspondente à posição (ou `NULL`, caso a posição não tenha sido listada em `switch()`).

```
x <- 2
```

```
switch(x,
  a = ,
  b = "ab",
  c = ,
  d = "cd"
)
```

```
## [1] "ab"
```

Caso seja informado “e”, será retornado `NULL`, uma vez que o valor não foi listado.

Note também que, para `x = "a"`, é retornado o valor “ab”. Isso ocorre porque `switch()` retornará o primeiro elemento não ausente da lista. O código acima pode ser entendido como: “se `x = "a"` ou `x = "b"`, retorne “ab”; se `x = c` ou `x = d`, retorne “cd”; retorne `NULL` nos demais casos.”

15.5 Argumentos de funções

Exercício 15.5.1

O que `commass(letters, collapse = "-")` faz? Por quê?

Solução. x

Exercício 15.5.2

Seria bom se você pudesse fornecer vários caracteres ao argumento `pad`, por exemplo, `rule("Title", pad = "-+")`. Por que isso não funciona atualmente? Como você corrigiria isso?

Solução. x

Exercício 15.5.3

O que o argumento `trim` para `mean()` faz? Quando você pode usá-lo?

Solução. x

Exercício 15.5.4

O valor padrão para o argumento `method` para `cor()` é `c("pearson", "kendall", "spearmann")`. O que isso significa? Qual valor é usado por padrão?

Solução. x

15.6 Retorno de valores

Não temos exercícios nesta seção.

15.7 Ambiente

Não temos exercícios nesta seção.



16

Vetores

16.1 Introdução

Não temos exercícios nesta seção.

16.2 O Básico de vetores

Não temos exercícios nesta seção.

16.3 Tipos importantes de vetores atômicos

Exercício 16.3.1

Descreva a diferença entre `is.finite(x)` e `!is.infinity(x)`.

Solução. Para obter essa resposta, vamos comparar o comportamento dessas duas funções aplicadas sobre um vetor contendo `0`, `NA`, `NaN`, `Inf` e `-Inf`:

```
x <- c(0, NA, NaN, Inf, -Inf)
```

```
x
```

```
## [1] 0 NA NaN Inf -Inf
```

```
is.finite(x)  
  
## [1] TRUE FALSE FALSE FALSE FALSE  
  
is.infinite(x)  
  
## [1] FALSE FALSE FALSE TRUE TRUE
```

Podemos notar que a função `is.finite()` considera que os valores `NA`, `NaN`, `Inf` e `-Inf` não são finitos, enquanto todo o resto é finito. Já a função `is.infinite()` funciona de uma forma levemente diferente, não sendo a simples negação da função anterior. `is.infinite()` considera apenas `Inf` e `-Inf` como infinitos e todo o restante (valores não faltantes, `NA` e `NaN`) como não infinitos.

Em outras palavras, podemos dizer que `NA` e `NaN` não são nem finitos, nem infinitos.

Exercício 16.3.2

Leia o código fonte de `dplyr::near()`. Como ele funciona?

Solução. Esta função verifica se o módulo da diferença entre os números é menor do que um valor de tolerância. A tolerância padrão é um valor característico da máquina na qual está rodando o R, um valor muito pequeno, em torno da raiz quadrada de `2.220446e-16`.

Exercício 16.3.3

Um vetor lógico pode receber três valores possíveis. Quantos valores possíveis um vetor `integer` pode receber? Quantos valores possíveis um `double` pode receber? Use o Google para pesquisar.

Solução. x

Exercício 16.3.4

Pense em pelo menos quatro funções que permitem que você converta um `double` em um `integer`. Como elas diferem? Seja preciso.

Solução. As funções `as.integer()` e `trunc()` retornam apenas a parte inteira de um determinado número passado como parâmetro. A função `floor()` retorna o maior inteiro menor do que ou igual ao número recebido como parâmetro. A função `ceiling()` retornam o menor inteiro que é maior do que ou igual ao número recebido como atributo. Já a função `round()` retorna o número arredondado para um inteiro.

Exercício 16.3.5

Quais funções do pacote **readr** possibilitam que você transforme uma string em um vetor lógico, integer ou double?

Solução. As funções são, respectivamente, `parse_logical()`, `parse_integer()` e `parse_number()`.

16.4 Usando vetores atômicos**Exercício 16.4.1**

O que `mean(is.na(x))` lhe diz sobre um vetor `x`? e `sum(!is.finite(x))`?

Solução. O comando `mean(is.na(x))` calcula a proporção de valores ausentes ou `NaN` no vetor `x`. Já o comando `sum(!is.finite(x))` calcula o número de valores não finitos em `x`.

Exercício 16.4.2

Leia cuidadosamente a documentação de `is.vector()`. O que ele realmente testa? Por que `is.atomic()` não concorda com as definições de vetores atômicos acima?

Solução. A função `is.vector()` retorna `TRUE` se `x` não contiver nenhum atributo além do nome. Neste caso, uma lista é também considerada um vetor. A função `is.atomic()` verifica se um objeto é de um dos tipos atômicos (lógico, inteiro, numérico, complexo, caractere e raw). Ela não leva em conta se o objeto tem outros atributos além do nome e esse é o ponto de diferença para a definição apresentada no livro.

Exercício 16.4.3

Compare e contraste `setNames()` com `purrr::set_names()`.

Solução. `x`

Exercício 16.4.4

Crei funções que recebam um vetor como entrada e retornem:

- a. o último valor. Você deveria usar `[` ou `[[`?

- b. Os elementos das posições pares.
- c. Cada elemento, exceto o último valor.
- d. Apenas números pares (e nenhum valor faltante).

Solução. Inicialmente, vamos definir um vetor que será utilizado em todos os exercícios:

```
x <- c(1, 2, NA, 4, NaN, 6, Inf, 8, 9)
```

- a. Para retornar o último elemento do vamos usar o seguinte:

```
last <- function(x) {
  x[[length(x)]]
}

last(x)
```

```
## [1] 9
```

- b. Para retornar os elementos nas posições pares:

```
even_positions <- function(x) {
  x[-seq(1,9,2)]
}

even_positions(x)
```

```
## [1] 2 4 6 8
```

- c. Para retornar todos os elementos, exceto o último:

```
remove_last <- function(x) {
  x[-length(x)]
}

remove_last(x)
```

```
## [1] 1 2 NA 4 NaN 6 Inf 8
```

- d. Para retornar apenas os números pares, mas nenhum valor faltante:

```
even_values <- function(x) {  
  (x %% 2 == 0 & is.finite(x))  
}  
  
even_values(x)
```

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

Exercício 16.4.5

Por que `x[-which(x > 0)]` não é o mesmo que `x[x <= 0]`?

Solução. `x`

Exercício 16.4.6

O que acontece quando você faz um subconjunto com integer positivo que é maior do que o comprimento do vetor? O que acontece quando você faz um subconjunto com um nome que não existe?

Solução. Em ambos os casos, será retornado `NA`.

16.5 Vetores recursivos (listas)

Exercício 16.5.1

Desenhe as seguintes listas como conjuntos agrupados:

- a. `list(a, b, list(c, d), list(e, f))`
- b. `list(list(list(list(list(a))))))`

Solução. `x`

Exercício 16.5.2

O que acontece se você fizer um subconjunto de um tibble como se estivesse fazendo um subconjunto de uma lista?

Solução. Como o tibble é um tipo especial de lista, os subconjuntos funcionarão da mesma maneira. A principal diferença que se pode notar entre tibbles e listas comuns, é que todas as “colunas” de um tibble vão ter o mesmo tamanho.

16.6 Atributos

Não temos exercícios para esta seção.

16.7 Vetores aumentados

Exercício 16.7.1

O que `hms::hms(3600)` retorna? Como é impresso? Sobre qual tipo primitivo o vetor aumentado é construído? Quais atributos ele usa?

Solução. `hms::hms(3600)` retorna a hora “01:00:00”, que é impresso no formato padrão de hora que usamos.

O tipo básico sobre o qual é construído é `double`:

```
x <- hms::hms(3600)
```

```
typeof(x)
```

```
## [1] "double"
```

Os atributos do objeto são `units` e `class`:

```
attributes(x)
```

```
## $units
## [1] "secs"
##
## $class
## [1] "hms"      "difftime"
```

Exercício 16.7.2

Teste fazer um tibble que tenha colunas com comprimentos diferentes. O que acontece?

Solução. Caso uma das colunas seja um escalar, o tibble será construído e o valor escalar reciclado até o tamanho da maior coluna.

```
tibble(x = 1, y = 1:5)
```

```
## # A tibble: 5 × 2
##       x     y
##   <dbl> <int>
## 1     1     1
## 2     1     2
## 3     1     3
## 4     1     4
## 5     1     5
```

Contudo, se ambas as colunas forem vetores de tamanhos diferentes e maiores do que 1, um erro será retornado.

```
tibble(x = 1:2, y = 1:4)
```

```
## Error in `tibble()`:
## ! Tibble columns must have compatible sizes.
## * Size 2: Existing data.
## * Size 4: Column `y`.
## i Only values of size one are recycled.
```

Exercício 16.7.3

Com base na definição anterior, há um problema em ter uma lista como uma coluna de um tibble?

Solução. Desde que a lista informada tenha a quantidade correta de elementos, não há problema em criar o tibble.

```
tibble(  
  x = 1:3,  
  y = list("a", 1:3, list(1:3))  
)
```

```
## # A tibble: 3 × 2  
##       x     y  
##   <int> <list>  
## 1     1 <chr [1]>  
## 2     2 <int [3]>  
## 3     3 <list [1]>
```

17

Iteração com purrr

Em um dos exercícios desta seção, iremos comparar a performance de dois métodos utilizando o pacote **rbenchmark**.

```
library("rbenchmark")
```

17.1 Introdução

Não temos exercícios para esta seção.

17.2 Loops for

Exercício 17.2.1

Escreva loops for para:

- a. Calcular a média de cada coluna em `mtcars`.
- b. Determinar o tipo de cada colunas em `nycflights13::flights`.
- c. Calcular o número de valores únicos em cada coluna de `iris`.
- d. Gerar 10 valores aleatórios normalmente distribuídos para cada `mu = -10, 0, 10, 100.`

Pense sobre saída, seuência e corpo *antes* de começar a escrever o loop.

Solução.

- a. Para calcular a média de cada uma das colunas de `mtcars`, podemos utilizar o seguinte *loop*:

```

output <- vector("double", ncol(mtcars))
for(i in seq_along(mtcars)) {
  output[[i]] <- mean(mtcars[[i]])
}

output

## [1] 20.090625  6.187500 230.721875 146.687500  3.596563  3.217250
## [7] 17.848750  0.437500  0.406250  3.687500  2.812500

```

b. Para determinar o tipo de cada uma das colunas de `nycflights13::flights`, podemos usar o seguinte:

```

output <- vector("character", ncol(flights))
for (i in seq_along(flights)) {
  output[[i]] <- typeof(flights[[i]])
}

output

## [1] "integer"   "integer"   "integer"   "integer"   "integer"   "double"
## [7] "integer"   "integer"   "double"    "character" "integer"   "character"
## [13] "character" "character" "double"    "double"    "double"    "double"
## [19] "double"

```

c. Para calcular o número de valores únicos em cada coluna do dataset `iris`, podemos usar o seguinte:

```

output <- vector("integer", ncol(iris))
for (i in seq_along(iris)) {
  output[[i]] <- iris[[1]] %>%
    unique() %>%
    length()
}

output

```

```
## [1] 35 35 35 35 35
```

d. Para gerar 10 distribuições normais aleatórias para cada um dos valores citados de `mu`, podemos usar o seguinte:

```
n <- 10
mu <- c(-10, 0, 10, 100)
normals <- vector("list", length(mu))
for (i in seq_along(normals)) {
  normals[[i]] <- rnorm(n, mean = mu[[i]])
}
str(normals)

## List of 4
## $ : num [1:10] -11.02 -9.83 -9.14 -12.5 -10.09 ...
## $ : num [1:10] 0.418 -0.236 -0.213 1.713 1.322 ...
## $ : num [1:10] 10.88 9.85 11.6 11.23 11.14 ...
## $ : num [1:10] 98.8 99.7 99.5 103.1 100.5 ...
```

Exercício 17.2.2

Elimine o loop for em cada um dos exemplos a seguir aproveitando uma função existente que dê certo com vetores.

```
out <- ""
for(x in letters) {
  out <- stringr::str_c(out, x)
}

x <- sample(100)
sd <- 0
for (i in seq_along(x)) {
  sd <- sd + (x[i] - mean(x)) ^ 2
}
sd <- sqrt(sd / (length(x) - 1))

x <- runif(100)
out <- vector("numeric", length(x))
out[1] <- x[1]
for (i in 2:length(x)) {
  out[i] <- out[i - 1] + x[i]
}
```

Solução. O primeiro loop apenas concatena os valores do vetor, e poderia ser substituído por:

```
stringr::str_c(letters, collapse = "")
```

```
## [1] "abcdefghijklmnopqrstuvwxyz"
```

O segundo loop realiza o cálculo do desvio padrão e poderia ser substituído por:

```
x <- sample(100)
sd <- sd(x)
sd
```

```
## [1] 29.01149
```

O último loop calcula a soma cumulativa de um vetor e poderia ser substituído por:

```
x <- runif(100)
out <- cumsum(x)
out
```

```
## [1]  0.3538083  0.3945823  0.4413880  0.9897417  1.0329999  1.6483127
## [7]  2.3156320  2.7272238  3.3362075  3.5521414  4.5418137  4.6921853
## [13]  5.0117227  5.2465283  6.1707765  6.9333120  7.0665232  7.5412228
## [19]  8.2346457  9.0208561  9.9201496 10.0388252 10.8936189 11.4841401
## [25] 11.5573675 12.4204475 12.6321860 13.2910739 14.2209617 15.1347410
## [31] 15.6365591 16.1661659 16.9592488 17.6838586 17.9201595 18.8353828
## [37] 19.0609482 19.9513397 20.1841585 20.8475252 21.0276840 21.5650052
## [43] 22.1351938 22.5162037 22.7999979 23.0728642 23.3926577 24.3613883
## [49] 24.3614335 25.0024871 25.5479950 26.4360941 27.1160498 27.5545142
## [55] 28.1310179 28.1989308 29.0028257 29.0760067 29.3954062 30.1728611
## [61] 31.1167954 31.6799687 32.5587460 33.0302280 33.6687764 34.3573347
## [67] 34.3869736 34.9669715 35.1002186 35.3692568 36.0703883 36.3249867
## [73] 36.8086244 36.8479010 37.2757523 38.2720669 38.9864245 39.3124931
## [79] 39.8056399 40.3899609 40.5266476 41.1383906 41.4979187 42.3978495
## [85] 42.9589725 43.2538442 43.7279552 43.9528413 44.8358273 45.6492699
## [91] 45.9494113 46.2500975 46.2925701 46.4735129 47.2107887 47.7805391
## [97] 48.2776787 48.4318656 48.9712018 49.2358491
```

Exercício 17.2.3

Combine suas habilidades de escrita de funções e loops for:

- a. Escreva um loop for que imprima (`print()`) a letra da música infantil “Alice the Camel”.
- b. Converta a cantiga infantil “Ten in the Bed” em uma função. Generalize-a para qualquer número de pessoas em qualquer estrutura de dormir.
- c. Converta a música “99 Bottles of Beer on the Wall” em uma função. Generalize-a para qualquer número de qualquer tipo de recipiente contendo qualquer líquido em qualquer superfície.

Solução.

a.

```
humps <- 5
for (i in seq(humps, 0, -1)) {
  cat(
    str_c("Alice the camel has ", rep(i, 3), " humps", collapse = "\n"),
    collapse = "\n"
  )

  if(i == 0) {
    cat("Now Alice is a horse.\n")
  } else {
    cat("So go, Alice, go. Boom, boom, boom!\n\n")
  }
}
```

```
## Alice the camel has 5 humps
## Alice the camel has 5 humps
## Alice the camel has 5 humps
## So go, Alice, go. Boom, boom, boom!
##
## Alice the camel has 4 humps
## Alice the camel has 4 humps
## Alice the camel has 4 humps
## So go, Alice, go. Boom, boom, boom!
##
## Alice the camel has 3 humps
## Alice the camel has 3 humps
## Alice the camel has 3 humps
## So go, Alice, go. Boom, boom, boom!
##
## Alice the camel has 2 humps
## Alice the camel has 2 humps
```

```
## Alice the camel has 2 humps
## So go, Alice, go. Boom, boom, boom!
##
## Alice the camel has 1 humps
## Alice the camel has 1 humps
## Alice the camel has 1 humps
## So go, Alice, go. Boom, boom, boom!
##
## Alice the camel has 0 humps
## Alice the camel has 0 humps
## Alice the camel has 0 humps
## Now Alice is a horse.
```

b.

```
n_in_the_bed <- function(n = 10, structure = "bed") {
  cat("LYRICS FOR ", n, " IN THE ", str_to_upper(structure), "\n\n")

  for (i in n:2) {
    cat("There were ", i, " in the ", structure, "\n")
    cat("And the little one said \"Roll over, roll over\"\n")
    cat("So they all rolled over and one fell out \n\n")
  }

  cat("There was ", i - 1, " in the bad\n")
  cat("And the little one said, \"Goodnight\" \n\n")
}

n_in_the_bed()
```

```
## LYRICS FOR 10 IN THE BED
##
## There were 10 in the bed
## And the little one said "Roll over, roll over"
## So they all rolled over and one fell out
##
## There were 9 in the bed
## And the little one said "Roll over, roll over"
## So they all rolled over and one fell out
##
## There were 8 in the bed
## And the little one said "Roll over, roll over"
## So they all rolled over and one fell out
```

```
##  
## There were 7 in the bed  
## And the little one said "Roll over, roll over"  
## So they all rolled over and one fell out  
##  
## There were 6 in the bed  
## And the little one said "Roll over, roll over"  
## So they all rolled over and one fell out  
##  
## There were 5 in the bed  
## And the little one said "Roll over, roll over"  
## So they all rolled over and one fell out  
##  
## There were 4 in the bed  
## And the little one said "Roll over, roll over"  
## So they all rolled over and one fell out  
##  
## There were 3 in the bed  
## And the little one said "Roll over, roll over"  
## So they all rolled over and one fell out  
##  
## There were 2 in the bed  
## And the little one said "Roll over, roll over"  
## So they all rolled over and one fell out  
##  
## There was 1 in the bad  
## And the little one said, "Goodnight"
```

```
n_in_the_bed(5, "rede")
```

```
## LYRICS FOR 5 IN THE REDE  
##  
## There were 5 in the rede  
## And the little one said "Roll over, roll over"  
## So they all rolled over and one fell out  
##  
## There were 4 in the rede  
## And the little one said "Roll over, roll over"  
## So they all rolled over and one fell out  
##  
## There were 3 in the rede  
## And the little one said "Roll over, roll over"  
## So they all rolled over and one fell out
```

```
##
## There were 2 in the rede
## And the little one said "Roll over, roll over"
## So they all rolled over and one fell out
##
## There was 1 in the bad
## And the little one said, "Goodnight"
```

C.

```
beverage_in_the_surface <- function(n = 99, recipient = "bottle", beverage = "beer", surface = "wall") {

  for (i in seq(n, 2, -1)) {
    cat(i, " ", recipient, " of ", beverage, " in the ", surface, "\n", collapse = "", sep = "")
    cat("Take one down and pass it around\n", collapse = "", sep = "")
    cat(i - 1, " ", recipient, " of ", beverage, " in the ", surface, "\n\n", collapse = "", sep = "")
  }

  if(i == 2) {
    cat(i - 1, " ", recipient, " of ", beverage, " in the ", surface, "\n", collapse = "", sep = "")
    cat("Take one down and pass it around\n", collapse = "", sep = "")
    cat("No more ", recipient, " of ", beverage, " in the ", surface, "\n\n", collapse = "", sep = "")

    cat("No more ", recipient, " of ", beverage, " in the ", surface, "\n", collapse = "", sep = "")
    cat("Go to the store and buy some more\n", collapse = "", sep = "")
    cat(n, " ", recipient, " of ", beverage, " in the ", surface, "\n\n", collapse = "", sep = "")
  }
}

beverage_in_the_surface(5, surface = "sofa")

## 5 bottle of beer in the sofa
## Take one down and pass it around
## 4 bottle of beer in the sofa
##
## 4 bottle of beer in the sofa
## Take one down and pass it around
## 3 bottle of beer in the sofa
##
## 3 bottle of beer in the sofa
## Take one down and pass it around
## 2 bottle of beer in the sofa
##
```

```
## 2 bottle of beer in the sofa
## Take one down and pass it around
## 1 bottle of beer in the sofa
##
## 1 bottle of beer in the sofa
## Take one down and pass it around
## No more bottle of beer in the sofa
##
## No more bottle of beer in the sofa
## Go to the store and buy some more
## 5 bottle of beer in the sofa
```

Exercício 17.2.4

É comum ver loops for que não pré-alocam a saída e, em vez disso, aumentam o comprimento de um vetor a cada passo:

```
output <- vector("integer", 0)
for (i in seq_along(x)) {
  output <- c(output, lengths(x[[i]])))
}
output
```

Como isso afeta o desempenho? Projete e execute um experimento.

Solução. Para avaliar o benefício de usar a alocação prévia para o vetor de saída, vamos inicialmente definir duas funções. A primeira delas, utiliza a alocação prévia da saída, enquanto a segunda, atualiza o vetor de saída a cada passo.

```
no_alloc <- function(x) {
  out <- vector("integer", 0)
  for (i in seq_along(x)) {
    out <- c(out, lengths(x[[i]])))
  }
  out
}

with_alloc <- function(x) {
  out <- vector("integer", length(x))
  for (i in seq_along(x)) {
    out[[i]] <- lengths(x[[i]]))
  }
  out
}
```

Com as funções definidas, vamos avaliar o tempo de execução utilizando a biblioteca **rbenchmark**. Vamos utilizar um objeto simples como argumento: um vetor com mil números inteiros.

```
x <- 1:1000
benchmark(
  "No allocation" = no_alloc(x),
  "With allocation" = with_alloc(x)
)

##          test replications elapsed relative user.self sys.self user.child
## 1   No allocation           100    0.28       4    0.25    0.03      NA
## 2 With allocation           100    0.07       1    0.08    0.00      NA
##   sys.child
## 1      NA
## 2      NA
```

Já podemos verificar uma diferença considerável. O método sem alocação prévia demorou aproximadamente 7.5 vezes mais para ser executado!

17.3 Variações do loop `for`

Exercício 17.3.1

Imagine que você tenha um diretório cheio de arquivos CSV que deseja ler. Você tem os caminhos deles em um vetor, `files <- dir("data/", pattern = "\\.csv$", full.names = TRUE)`, e agora quer ler cada um com `read_csv()`. Escreva o loop que os carregará em um único data frame.

Solução.

```
files <- dir("data\\", pattern = "\\.csv$", full.names = TRUE)

df <- vector("list", length(files))
for (i in seq_along(files)) {
  df[[i]] <- read_csv(files[[i]])
}
```

```
## Rows: 169 Columns: 8
## -- Column specification -----
## 
## Delimiter: ","
## chr (1): Plot
## dbl (5): No.stems, Height, Weight, Seed.heads, Seeds.in.25.heads
## lgl (2): Seed.herbivore, Root.herbivore
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
print(df)
```

```
## [[1]]
## # A tibble: 169 x 8
##   Plot   Seed.herbivore
##   <chr>  <lgcl>
## 1 plot-2 TRUE
## 2 plot-2 TRUE
## 3 plot-2 TRUE
## 4 plot-2 TRUE
## 5 plot-2 TRUE
## 6 plot-2 TRUE
## 7 plot-4 FALSE
## 8 plot-6 TRUE
## 9 plot-6 TRUE
## 10 plot-6 TRUE
## # i 159 more rows
## # i 6 more variables:
## #   Root.herbivore <lgcl>,
## #   No.stems <dbl>,
## #   Height <dbl>,
## #   Weight <dbl>, ...
```

```
df <- bind_rows(df)
```

```
print(df)
```

```
## # A tibble: 169 x 8
##   Plot   Seed.herbivore
```

```

## <chr> <lgl>
## 1 plot-2 TRUE
## 2 plot-2 TRUE
## 3 plot-2 TRUE
## 4 plot-2 TRUE
## 5 plot-2 TRUE
## 6 plot-2 TRUE
## 7 plot-4 FALSE
## 8 plot-6 TRUE
## 9 plot-6 TRUE
## 10 plot-6 TRUE
## # i 159 more rows
## # i 6 more variables:
## #   Root.herbivore <lgl>,
## #   No.stems <dbl>,
## #   Height <dbl>,
## #   Weight <dbl>, ...

```

Exercício 17.3.2

O que acontece se você usar `for (nm in names(x))` e `x` não tiver nomes? E se somente alguns dos elementos forem nomeados? E se os nomes não forem únicos?

Solução. Caso o objeto `x` não tenha nomes, nenhum passo do loop será executado.

```

x <- c(1,2,3)
for (nm in names(x)) {
  print(nm)
}

```

Caso apenas alguns dos elementos tenha nome, os elementos não nomeados estarão com a propriedade `name` igual a `NA`, o que resultará em erros explícitos ao tentar buscá-los.

```

names(x) <- c("a", "b")
for (nm in names(x)) {
  print(x[[nm]])
}

```

```

## [1] 1
## [1] 2

```

```
## Error in x[[nm]]: índice fora de limites
```

Caso tenhamos nomes repetidos, apenas o primeiro elemento será recuperado.

```
names(x) <- c("a", "b", "a")
for (nm in names(x)) {
  print(x[[nm]])
}
```

```
## [1] 1
## [1] 2
## [1] 1
```

Exercício 17.3.3

Escreva uma função que imprima a média de cada coluna numérica em um data frame, junto com seu nome. Por exemplo, `show_mean(iris)` imprimiria:

```
show_mean(iris)
#> Sepal.Length: 5.84
#> Sepal.Width:  3.06
#> Petal.Length: 3.76
#> Petal.Width:  1.20
```

Solução.

```
show_mean <- function(x, digits = 2) {
  names <- names(x)
  for (i in seq_along(x)) {
    if (is.numeric(x[[i]])) {
      name <- names[[i]]
      mean <- mean(x[[i]], rm.na = TRUE)
      max_length <- max(str_length(names))

      cat(
        str_pad(str_c(name, ":"), max_length + 1, side = "right"),
        format(mean, digits = digits, nsmall = digits),
        sep = " ",
        collapse = "\n"
      )
    }
  }
}
```

```
    }
}

show_mean(iris)
```

```
## Sepal.Length: 5.84
## Sepal.Width:  3.06
## Petal.Length: 3.76
## Petal.Width:  1.20
```

Exercício I7.3.4

O que este código faz? Como ele funciona?

```
trans <- list(
  disp = function(x) x * 0.0163871,
  am = function(x) {
    factor(x, labels = c("auto", "manual"))
  }
)

for (var in names(trans)) {
  mtcars[[var]] <- trans[[var]](mtcars[[var]])
}
```

Solução. O código altera os valores das colunas `disp` e `am` do data frame `mtcars`. Inicialmente o código define uma lista contendo duas funções (`disp` e `am`) e, após, faz um loop percorrendo os nomes dos elementos nesta lista e, em cada passo, altera o valor da coluna correspondente, chamando a função da lista e passando a coluna como argumento.

17.4 Loops `for` versus funcionais

Exercício I7.4.1

Leia a documentação de `apply()`. No segundo caso, quais dois loops `for` ela generaliza?

Solução. x

Exercício 17.4.2

Adapte `col_summary()` para que se aplique apenas a colunas numéricas. Você pode querer começar com uma função `is_numeric()` que retorne um vetor lógico que tenha `TRUE` correspondente a cada coluna numérica.

Solução. x

17.5 As funções `map`**Exercício 17.5.1**

x

Solução. x

Exercício 17.5.2

x

Solução. x

Exercício 17.5.3

x

Solução. x

Exercício 17.5.4

x

Solução. x

Exercício 17.5.5

x

Solução. x

17.6 Lidando com falhas

Não temos exercícios para esta seção.

17.7 Fazendo `map` com vários argumentos

Não temos exercícios para esta seção.

17.8 Walk

Não temos exercícios para esta seção.

17.9 Outros padrões para loops `for`

Exercício 17.9.1

x

Solução. x

Exercício 17.9.2

x

Solução. x

Exercício 17.9.3

x

Solução. x

Parte IV

Modelar



18

O básico de modelos com modelr

Neste capítulo, utilizaremos o pacote **modelr**:

```
library(modelr)
```

18.1 Introdução

Não temos exercícios para esta seção.

18.2 Um modelo simples

Exercício 18.2.1

A desvantagem do modelo linear é que ele é sensível a valores incomuns, porque a distância incorpora um termo quadrado. Encaixe um modelo linear nos seguintes dados simulados e visualize os resultados. Reexecute algumas vezes para gerar diferentes conjuntos de dados simulados. O que você nota sobre o modelo?

```
simla <- tibble(
  x = rep(1:10, each = 3),
  y = x * 1.5 + 6 + rt(length(x), df = 2)
)
```

Solução.

```

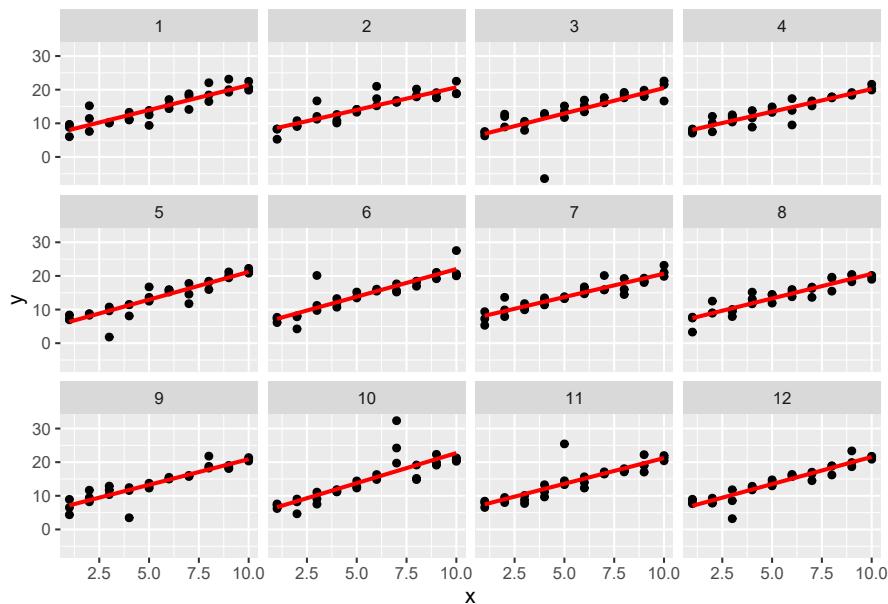
data <- function(i) {
  tibble(
    x = rep(1:10, each = 3),
    y = x * 1.5 + 6 + rt(length(x), df = 2),
    .id = i
  )
}

models <- map_df(1:12, data)

ggplot(models, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm", color = "red", se = FALSE) +
  facet_wrap(~.id, ncol = 4)

```

`geom_smooth()` using formula = 'y ~ x'



Exercício 18.2.2

Uma maneira de tornar os modelos lineares mais robustos é usar uma medida de distância diferente. Por exemplo, em vez de distância da raiz quadrática média, você poderia usar a distância média absoluta:

```
measure_distance <- function(mod, data) {  
  diff <- data$y - make_prediction(mod, data)  
  mean(abs(diff))  
}
```

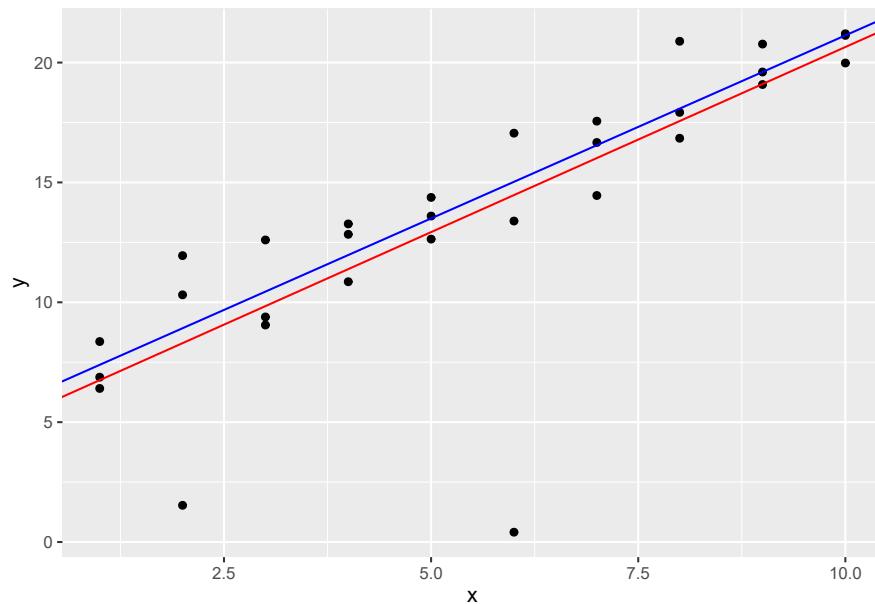
Use `optim()` para ajustar esse modelo nos dados previamente simulados e compare-o ao modelo linear.

Solução. Inicialmente, vamos reescrever as duas distâncias e construir os modelos lineares.

```
# Definindo a distância baseada no erro quadrático médio  
erro_quadratico_medio <- function(mod, data) {  
  diff <- data$y - (mod[1] + data$x * mod[2])  
  sqrt(mean(diff ^ 2))  
}  
  
# Definindo a distância baseada na média da distância absoluta  
distancia_media_absoluta <- function(mod, data) {  
  diff <- data$y - make_prediction(mod, data)  
  mean(abs(diff))  
}  
  
# Definindo a função auxiliar  
make_prediction <- function(mod, data) {  
  mod[1] + mod[2] * data$x  
}  
  
# Gerando o modelo otimizado  
modelo_erro_quadratico_medio <- optim(c(0,0), erro_quadratico_medio, data = sim1a)  
modelo_distancia_media_absoluta <- optim(c(0,0), distancia_media_absoluta, data = sim1a)
```

Agora iremos plotar cada um dos modelos junto aos dados, para que possamos compará-los visualmente.

```
ggplot(sim1a, aes(x,y)) +  
  geom_point() +  
  geom_abline(aes(intercept = modelo_erro_quadratico_medio$par[[1]], slope = modelo_erro_quadratico_medio$par[[2]]))  
  geom_abline(aes(intercept = modelo_distancia_media_absoluta$par[[1]], slope = modelo_distancia_media_absoluta$par[[2]]))
```



A diferença é visível no gráfico, contudo ainda nos parece difícil dizer qual das duas curvas se adequa melhor. Poderíamos tentar comparar a variável `value` em cada modelo, mas vale ressaltar que, como foram construídas com base em cálculos diferentes de distância, os valores não nos parecem comparáveis.

Exercício 18.2.3

Um desafio ao realizar otimização numérica é que ela só garante encontrar um ótimo local. Qual é o problema com a otimização de um modelo de três parâmetros como este?

```
model1 <- function(a, data) {
  a[1] + data$x * a[2] + a[3]
}
```

Solução. x

18.3 Visualizando modelos

Exercício 18.3.1

Em vez de usar `lm()` para ajustar uma linha reta, você pode usar `loess()` para ajustar uma curva suave. Repita o processo de ajuste de modelos, geração de grade, previsões e visualização em `sim1` usando `loess()`, em vez de `lm()`. Como o resultado se compara a `geom_smooth()`?

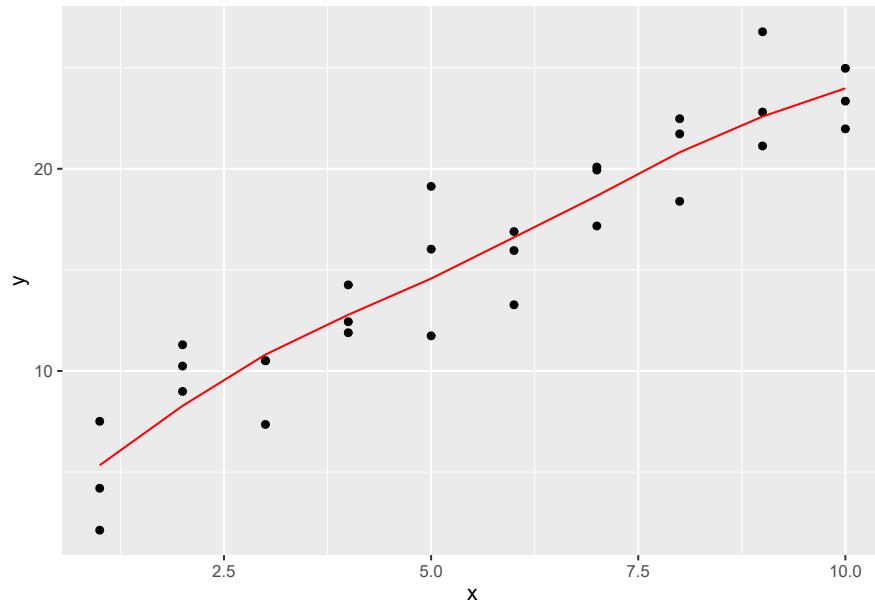
Solução. Inicialmente vamos ajustar o modelo e construir um novo data frame contendo, além dos dados, a predição e o resíduo.

```
modelo_loess <- loess(y ~ x, sim1)

dados_ajustados <- sim1 %>%
  add_predictions(modelo_loess) %>%
  add_residuals(modelo_loess)
```

Na sequência vamos plotar o gráfico com os dados e o resultado da predição:

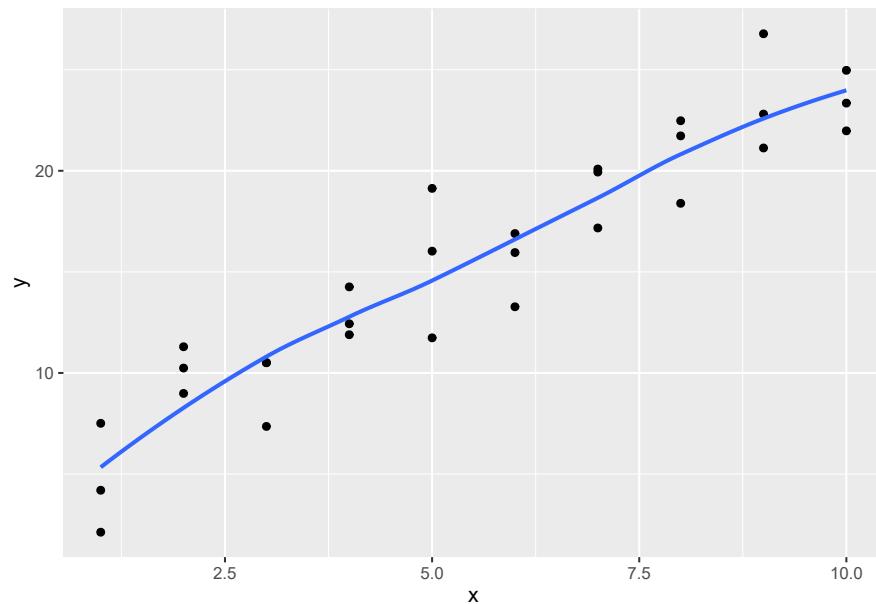
```
ggplot(dados_ajustados, aes(x)) +
  geom_point(aes(y = y)) +
  geom_line(aes(y = pred), color = "red")
```



E também os dados junto à `geom_smooth()`:

```
ggplot(dados_ajustados, aes(x, y)) +  
  geom_point() +  
  geom_smooth(method = "loess", se = FALSE)
```

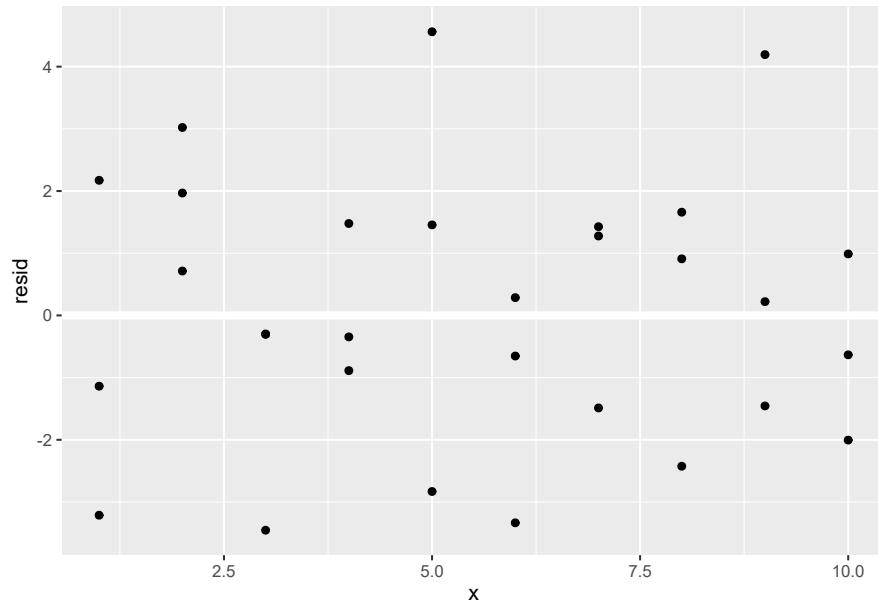
```
## `geom_smooth()` using formula = 'y ~ x'
```



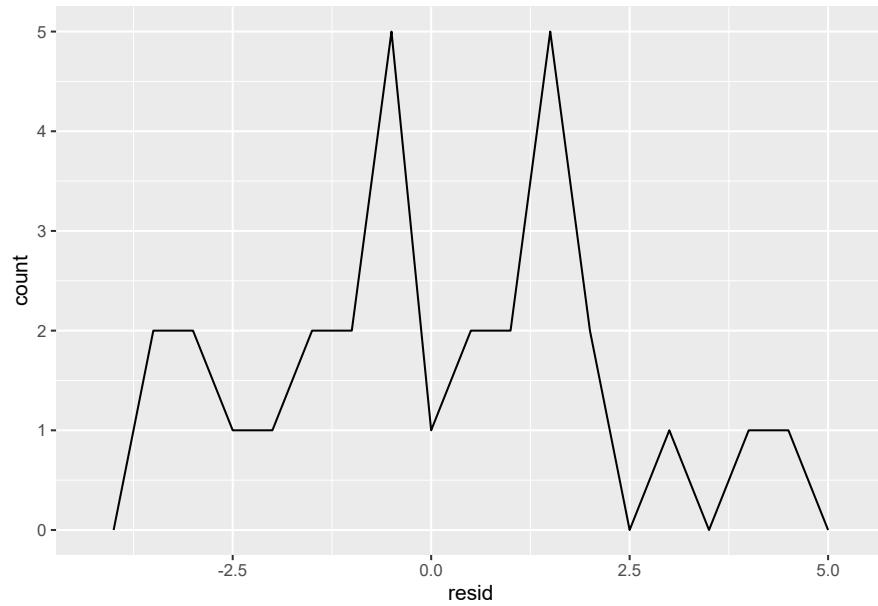
Notamos que não há diferença entre o modelo ajustado pela função `loess()` e o gerado pela `geom_smooth()`.

Agora plotaremos os resíduos:

```
ggplot(dados_ajustados, aes(x)) +  
  geom_point(aes(y = resid)) +  
  geom_ref_line(h = 0)
```



```
ggplot(dados_ajustados, aes(resid)) +  
  geom_freqpoly(binwidth = 0.5)
```



Como não identificamos um padrão para os resíduos, podemos entender que o

modelo é útil para explicar o comportamento dos dados, isto é, parece explicar o máximo possível dos dados, deixando o menor resíduo possível.

Exercício 18.3.2

`add_predictions()` é pareado com `gather_predictions()` e `spread_predictions()`. Como essas três funções diferem?

Solução. Enquanto a função `add_predictions()` trabalha apenas com um modelo por vez, as funções `spread_predictions()` e `gather_predictions()` conseguem trabalhar com uma lista de modelos, facilitando a comparação entre diversos modelos.

Exercício 18.3.3

O que `geom_ref_line()` faz? De qual pacote ela vem? Por que exibir uma linha de referência em gráficos mostrando resíduos é útil e importante?

Solução. Esta função é uma otimização das funções `geom_hline()` e `geom_vline()` que desenham linhas horizontais e verticais, respectivamente, num gráfico. Ela está definida no pacote `mdeLR` e é bastante útil porque nos permite desenhar uma referência para os resíduos.

Exercício 18.3.4

Por que você pode querer observar o polígono de frequência dos resíduos absolutos? Quais são os prós e os contras de comparados a observar os resíduos brutos?

Solução. x

18.4 Fórmulas e famílias de modelos

Exercício 18.4.1

O que acontece se você repetir a análise de `sim2` usando um modelo sem uma intersecção? O que acontece com a equação do modelo? E com as previsões?

Solução. Para executar o modelo sem a intersecção, precisamos adicionar `- 1` ou `+ 0` à nossa fórmula.

```
mod2 <- lm(y ~ x, data = sim2)
mod2a <- lm(y ~ x - 1, data = sim2)

sim2 %>%
  data_grid(x) %>%
  spread_predictions(mod2, mod2a)
```

```
## # A tibble: 4 x 3
##   x     mod2 mod2a
##   <chr> <dbl> <dbl>
## 1 a     1.15  1.15
## 2 b     8.12  8.12
## 3 c     6.13  6.13
## 4 d     1.91  1.91
```

Neste caso, não temos mudanças nos valores das previsões. (Por quê?)

Exercício 18.4.2

Use `model_matrix()` para explorar as equações geradas para os modelos que eu ajustei em `sim3` e `sim4`. Por que `*` é um bom atalho para cada interação?

Solução. Para começar, vamos gerar as equações para esses dois datasets:

```
x3 <- model_matrix(y ~ x1 * x2, data = sim3)
x4 <- model_matrix(y ~ x1 * x2, data = sim4)
```

Agora vamos visualizar `x3`, lembrando que o conjunto `sim3` contém uma variável categórica e uma contínua:

```
x3
```

```
## # A tibble: 120 x 8
##   `"(Intercept)"` x1    x2b
##   <dbl> <dbl> <dbl>
## 1 1       1       0
## 2 2       1       0
## 3 3       1       0
## 4 4       1       1
## 5 5       1       1
```

```

## 6          1      1      1
## 7          1      1      0
## 8          1      1      0
## 9          1      1      0
## 10         1      1      0
## # i 110 more rows
## # i 5 more variables:
## #   x2c <dbl>, x2d <dbl>,
## #   `x1:x2b` <dbl>,
## #   `x1:x2c` <dbl>, ...

```

Podemos notar que foram criadas as variáveis x_{2*} correspondentes aos valores de x_2 (exceto para a categoria a , que é o complemento do conjunto). Também foram criadas as colunas $x_{1:x_{2*}}$ que corresponde ao produto das colunas x_1 e x_{2*} . Podemos confirmar isso usando o seguinte:

```
all(x3[["x1:x2b"]] == x3[["x1"]] * x3[["x2b"]])
```

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

```
all(x3[["x1:x2c"]] == x3[["x1"]] * x3[["x2c"]])
```

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

```
all(x3[["x1:x2d"]] == x3[["x1"]] * x3[["x2d"]])
```

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

Agora avaliaremos o caso do conjunto `sim4`:

```
x4
```

```

## # A tibble: 300 x 4
##   `(Intercept)`     x1      x2
##   <dbl> <dbl> <dbl>
## 1       1     -1    -1

```

```

## 2      1   -1 -1
## 3      1   -1 -1
## 4      1   -1 -0.778
## 5      1   -1 -0.778
## 6      1   -1 -0.778
## 7      1   -1 -0.556
## 8      1   -1 -0.556
## 9      1   -1 -0.556
## 10     1   -1 -0.333
## # i 290 more rows
## # i 1 more variable:
## #   `x1:x2` <dbl>

```

Como ambas as variáveis são contínuas, nenhuma delas precisou ser quebrada. O modelo gerou a coluna `x1:x2` como o produto de `x1` por `x2`. O que confirmamos assim:

```
all(x4[["x1:x2"]] == x4[["x1"]] * x4[["x2"]])
```

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

Utilizar o `*` é útil quando desejamos considerar no modelo o efeito da interação entre duas ou mais variáveis.

Exercício 18.4.3

Usando os princípios básicos, converta em funções as fórmulas nos dois modelos a seguir. (Dica: comece convertendo a variável categórica em variáveis 0-1)

```

mod1 <- lm(y ~ x1 + x2, data = sim3)
mod2 <- lm(y ~ x1 * x2, data = sim3)

```

Solução. x

Exercício 18.4.4

Para `sim4`, qual é melhor, `mod1` ou `mod2`? Acredito que `mod2` faz um trabalho levemente melhor removendo padrões, mas é bem sutil. Você consegue criar um gráfico que suporte minha afirmação?

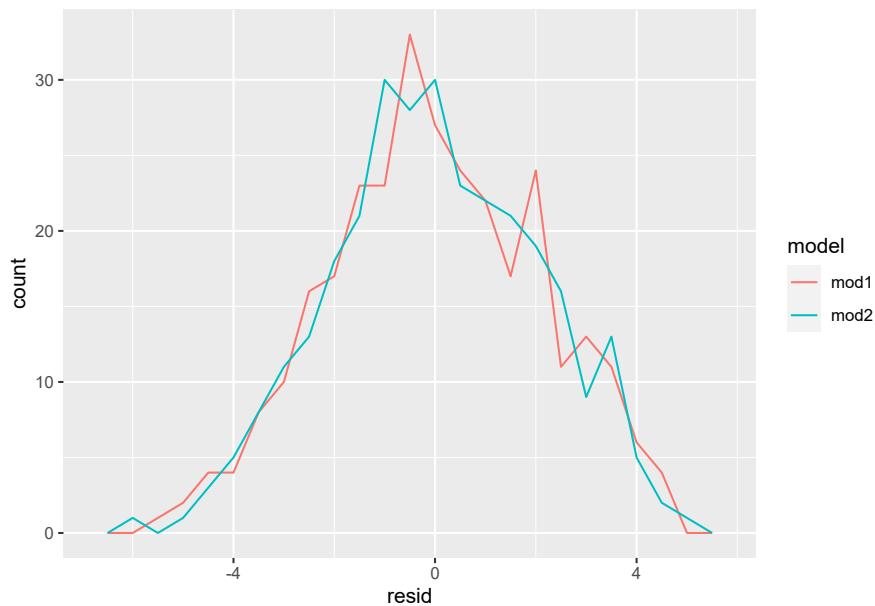
Solução. Inicialmente vamos construir os dois modelos (note que ambos usam a mesma métrica para cálculo do resíduo, dessa forma, poderemos compará-los).

```
mod1 <- lm(y ~ x1 + x2, sim4)
mod2 <- lm(y ~ x1 * x2, sim4)
```

Agora, para avaliar os modelos, vamos dar uma olhada nos resíduos.

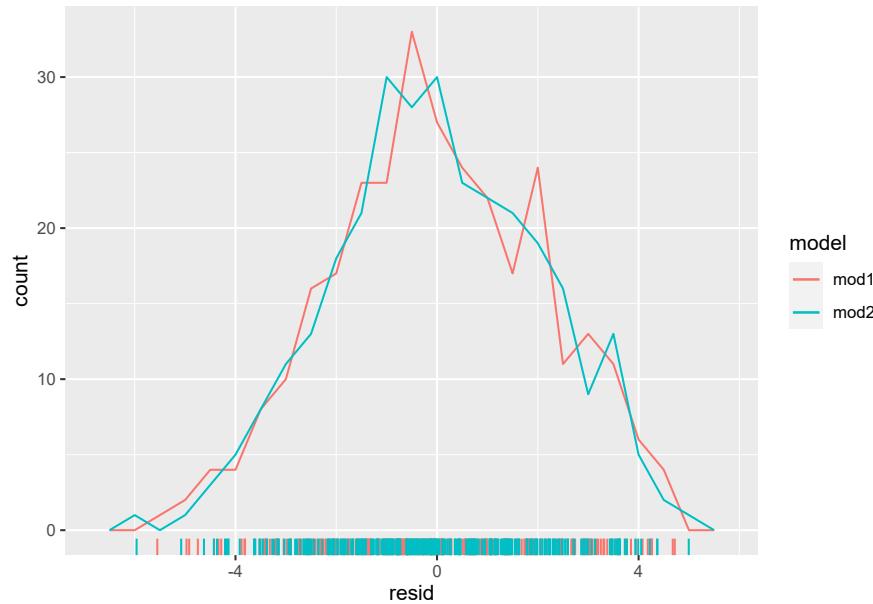
```
residuos <- sim4 %>%
  gather_residuals(mod1, mod2)

residuos %>%
  ggplot(aes(resid, color = model)) +
  geom_freqpoly(binwidth = 0.5)
```



Para ambos os modelos, vemos que os resíduos se concentram em torno de zero. Isso nos ajuda a ver que ambos os modelos são úteis, mas não nos ajuda na comparação entre eles. Vamos adicionar mais uma camada ao gráfico para tentar visualizar melhor.

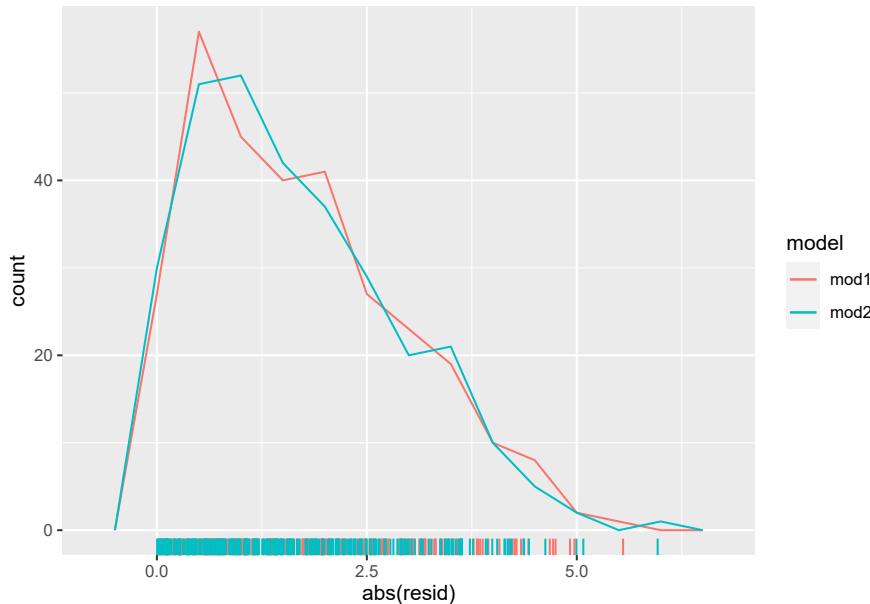
```
residuos %>%
  ggplot(aes(resid, color = model)) +
  geom_freqpoly(binwidth = 0.5) +
  geom_rug()
```



O gráfico ainda não é muito explicativo, mas conseguimos visualizar uma pequena diferença: a calda de `mod1` parece ser um pouco mais longa, isto é, o modelo possui resíduos mais extremos, se comparado ao `mod2`.

Vamos visualizar também o valor absoluto dos resíduos:

```
residuos %>%
  ggplot(aes(abs(resid), color = model)) +
  geom_freqpoly(binwidth = 0.5) +
  geom_rug()
```



A tendência ainda se mantém. Indicando que `mod2` é levemente melhor do que `mod1`. Vamos verificar ainda o desvio padrão dos resíduos de cada modelo:

```
residuos %>%
  group_by(model) %>%
  summarise(
    mean = mean(resid),
    sd = sd(resid)
  )
```

```
## # A tibble: 2 x 3
##   model     mean     sd
##   <chr>     <dbl>  <dbl>
## 1 mod1  1.85e-16  2.10
## 2 mod2  1.32e-16  2.07
```

Notamos que, apesar de pequena, existe uma diferença entre os modelos e `mod2` realmente gera um resíduo menor, ou seja, ele é um pouco melhor do que seu “adversário”.

18.5 Valores faltantes

Não temos exercícios para esta seção.

18.6 Outras famílias de modelos

Não temos exercícios para esta seção.

19

Construção de modelos

19.1 Introdução

Não temos exercícios para esta seção.

19.2 Por que diamantes de baixa qualidade são mais caros?

Para esta seção, necessitamos do seguinte código:

```
diamonds2 <- diamonds %>%
  filter(carat <= 2.5) %>%
  mutate(
    lprice = log2(price),
    lcarat = log2(carat)
  )

mod_diamond2 <- lm(lprice ~ lcarat + color + cut + clarity, data = diamonds2)

diamonds2 <- add_residuals(diamonds2, mod_diamond2, "lresid2")
```

Exercício 19.2.1

No gráfico `lcarat` versus `lprice` há algumas listras verticais claras. O que elas representam?

Solução. Eles representam a concentração de diamantes com valores arredondados para números “human friendly”, “bonitos”.

Exercício 19.2.2

Se $\log(\text{price}) = a_0 + a_1 * \log(\text{carat})$, o que isso diz sobre o relacionamento entre `price` e `carat`?

Solução. De modo bastante simplificado, se há uma relação linear entre $\log_2(\text{carat})$ e $\log_2(\text{price})$, podemos afirmar que há uma relação exponencial entre `carat` e `price`. Isso significa que o aumento de x unidades em `carat`, corresponde a um crescimento da ordem de 2^x em `price`.

Exercício 19.2.3

Extraia os diamantes que tenham resíduos muito altos ou muito baixos. Há alguma coisa estranha sobre esses diamantes? Eles são particularmente ruins ou bons, ou você acha que são erros de especificação?

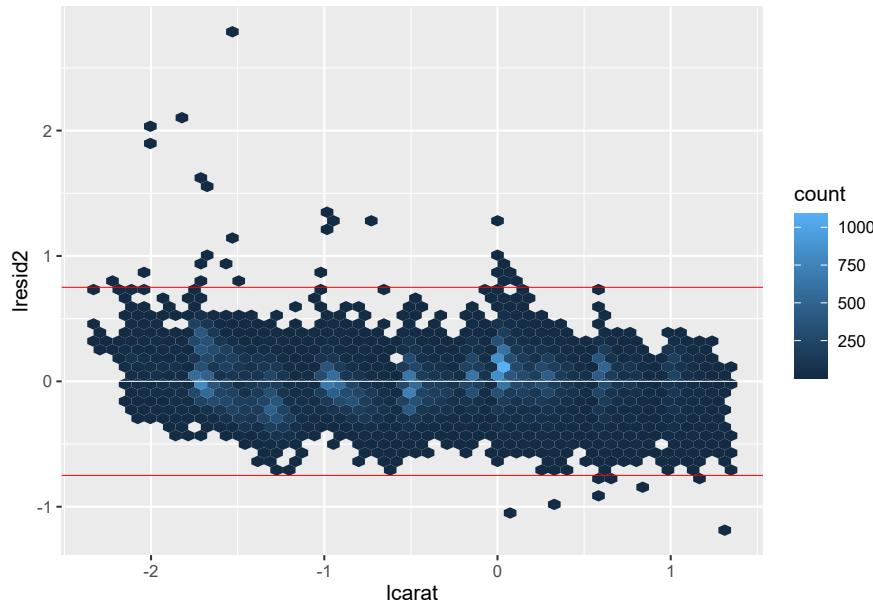
Solução. Não encontrei nada que possa indicar uma relação escondida. Acredito que as diferenças se devem a erros de especificação ou de entrada de dados.

Exercício 19.2.4

O modelo final, `mod_diamonds2`, faz um bom trabalho na previsão dos preços de diamantes? Você confiaria nele para dizer quanto gastar se fosse comprar um diamante?

Solução. Antes de responder a questão, vamos dar uma olhada na distribuição dos resíduos, no desvio padrão e mais algumas estatísticas sobre os erros.

```
ggplot(diamonds2, aes(lcarat, lresid2)) +
  geom_hex(bins = 50) +
  geom_ref_line(h = 0, size = 0.25) +
  geom_ref_line(h = 0.75, size = 0.25, colour = "red") +
  geom_ref_line(h = -0.75, size = 0.25, colour = "red")
```



```
diamonds2 %>%
  summarise(
    sd = sd(lresid2),
    rmse = sqrt(mean(lresid2^2)),
    mae = mean(abs(lresid2)),
    p025 = quantile(2^lresid2, 0.025),
    p975 = quantile(2^lresid2, 0.975)
  )
```

```
## # A tibble: 1 x 5
##       sd     rmse     mae   p025   p975
##     <dbl>    <dbl>    <dbl>    <dbl>
## 1 0.192 0.192 0.149 0.774
## # i 1 more variable:
## #   p975 <dbl>
```

Avaliando os dados, vemos que o erro quadrático médio é de aproximadamente 0.19 o que corresponderia a um erro em torno de 14%. Se usarmos o erro médio (em torno de 0.15), teríamos um erro na casa de 11%. Ademais, temos que 95% das previsões erram entre 23% e 31%.

Concluímos que, para uma ideia geral de preços, o modelo parece ser muito bom, contudo para uma empresa que trabalhe com compra e/ou venda de diamantes, um modelo mais robusto seria necessário.

19.3 O que afeta o número de voos diários?

Para solução dos exercícios desta seção, precisaremos do seguinte código:

```
daily <- flights %>%
  mutate(date = make_date(year, month, day)) %>%
  group_by(date) %>%
  summarise(n = n())

daily <- daily %>%
  mutate(wday = wday(date, label = TRUE))

term <- function(date) {
  cut(date,
    breaks = ymd(20130101, 20130605, 20130825, 20140101),
    labels = c("spring", "summer", "fall")
  )
}

daily <- daily %>%
  mutate(term = term(date))

mod <- lm(n ~ wday, data = daily)

daily <- daily %>%
  add_residuals(mod)

mod1 <- lm(n ~ wday, data = daily)
mod2 <- lm(n ~ wday * term, data = daily)
```

Exercício 19.3.1

Use suas habilidades de detetive no Google para pensar o porquê de haver menos voos do que o esperado para os dias 20 de janeiro, 26 de maio e 1º de setembro. (Dida: todos têm a mesma explicação.) Como esses dias seriam generalizados para outro ano?

Solução. Os três dias antecedem feriados nacionais nos EUA. Como a maioria dos voos são a trabalho, é esperada um queda no número de voos no dia anterior (similar à queda que ocorre nos sábados).

A regra geral para os feriados seria: - Dia de Martin Luther King Jr.: terceira segunda-feira de janeiro; - Memorial Day: última segunda-feira de maio; - Dia do trabalho: primeira segunda-feira de setembro.

Exercício 19.3.2

O que os três dias com resíduos positivos altos representam? Como esses dias seriam generalizados para outro ano?

```
daily %>%
  top_n(3, resid)
```

```
## # A tibble: 3 x 5
##   date           n wday term
##   <date>     <int> <ord> <fct>
## 1 2013-11-30    857 sáb  fall
## 2 2013-12-01    987 dom  fall
## 3 2013-12-28    814 sáb  fall
## # i 1 more variable:
## #   resid <dbl>
```

Solução. Esses dias correspondem aos finais de semana imediatamente após o dia de ação de graças e o natal. Podemos generalizar para os demais anos considerando as datas desses feriados.

Exercício 19.3.3

Crie uma nova variável que separe a variável `wday` em períodos, mas somente para os sábados. Por exemplo, ela deve ter `Thurs`, `Fri`, menos `Sat-summer`, `Sat-spring`, `Sat-fall`. Como esse modelo se compara com o modelo com cada combinação de `wday` e `term`?

Solução. Iniciaremos usando a função `case_when()` para criar a nova variável.

```
daily <- daily %>%
  mutate(
    wday2 =
      case_when(
        wday == "sáb" & term == "summer" ~ "sáb_summer",
        wday == "sáb" & term == "fall" ~ "sáb_fall",
        wday == "sáb" & term == "spring" ~ "sáb_spring",
```

```
TRUE ~ as.character(wday)
)
)
```

Na sequência, criaremos o modelo de predição:

```
mod3 <- lm(n ~ wday2, daily)
```

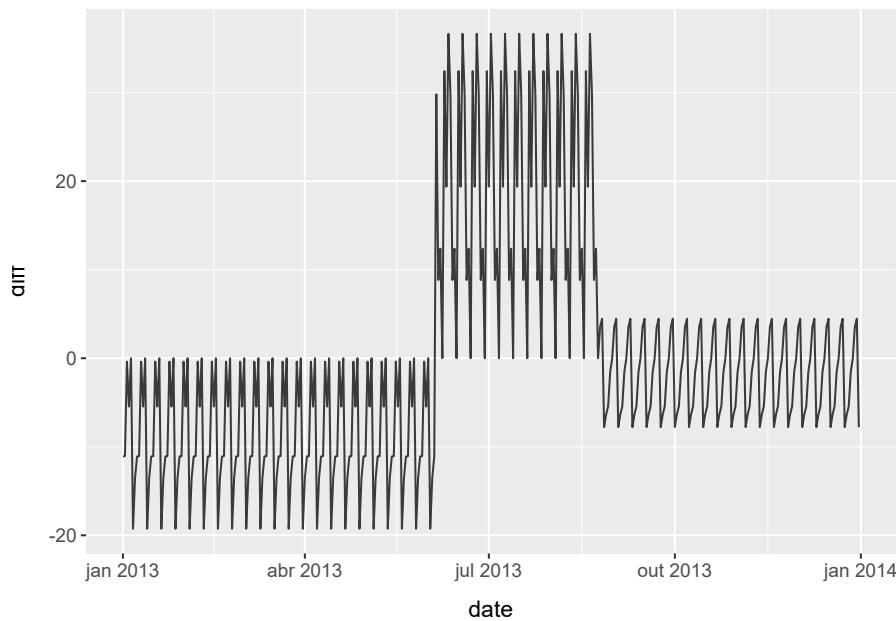
E exibiremos um gráfico comparando os modelos:

```
daily %>%
  gather_residuals(sab = mod3, all = mod2) %>%
  ggplot(aes(date, resid, color = model)) +
  geom_line(alpha = 0.75) +
  tema
```



Devido à quantidade de dados, fica difícil visualizar alguma coisa. Vamos construir um novo gráfico com base apenas na diferença entre os resíduos.

```
daily %>%
  spread_residuals(sab = mod3, all = mod2) %>%
  mutate(diff = sab - all) %>%
  ggplot(aes(date, diff)) +
  geom_line(alpha = 0.75) +
  tema
```



Notamos que o modelo que contém a separação conforme período escolar apresenta maiores diferenças durante o outono e menores na primavera.

Precisamo encontrar ainda uma maneira mais adequada de comparar modelos! Esta questão será marcada para revisão!

Exercício 19.3.4

Crie uma nova variável `wday` que combine o dia da semana, período (para sábados) e feriados nacionais. Como se parecem os resíduos desse modelo?

Solução. x

Exercício 19.3.5

O que acontece se você encaixar um efeito dia da semana que varie por mês (por exemplo, `n ~ wday * month`)? Por que isso não é útil?

Solução. x

Exercício 19.3.6

Como você esperaria que o modelo `n ~ wday + ns(date, 5)` fosse? A partir de seus conhecimentos sobre dados, por que você esperaria que ele não fosse particularmente eficaz?

Solução. x

Exercício 19.3.7

Nós formulamos a hipótese de que pessoas viajando aos domingos são mais propensas a serem viajantes a negócios, que precisam estar em algum lugar na segunda-feira. Explore essa hipótese observando como ela se desmembra com base na distância e no tempo: se for verdade, você esperaria ver mais voos nos domingos à noite para lugares distantes.

Solução. x

Exercício 19.3.8

É um pouco frustrante que domingo e sábado estejam em pontas separadas do gráfico. Escreva uma pequena função para estabelecer os níveis do fator para que a semana comece na segunda-feira.

Solução. x

19.4 Aprendendo mais sobre modelos

Não temos exercícios para esta seção.

20

Muitos modelos com purrr e broom

20.1 Introdução

20.2 gapminder

20.3 List-columns

20.4 Criando list-columns

20.5 Simplificando list-columns

20.6 Criando dados tidy com broom



Parte V

Comunicar



21

R Markdown

21.1 Introdução

21.2 O Básico de R Markdown

21.3 Formatação de texto com markdown

21.4 Trechos de código

21.5 Resolução de problemas

21.6 Header YAML

21.7 Aprendendo mais



22

Gráficos para comunicação com ggplot2

22.1 Introdução

22.2 Rótulo

22.3 Anotações

22.4 Escalas

22.5 Dando zoom

22.6 Temas

22.7 Salvando seus gráficos

22.8 Aprendendo mais



23

Formatos R Markdown

23.1 Introdução

23.2 Opções de saída

23.3 Documentos

23.4 Notebooks

23.5 Apresentações

23.6 Dashboards

23.7 Interatividade

23.8 Sites

23.9 Outros formatos

23.10 Aprendendo mais

24

Fluxo de trabalho de R Markdown



A

Apêndice A - Estudo de caso sobre a arrumação de dados com tidyverse

Introdução

Neste apêndice, reproduzimos o estudo de caso conduzido pelos autores para exemplificar o uso do `tidyverse` na arrumação de dados no capítulo 9 do livro.

Para este estudo de caso, foi utilizado o conjunto de dados `tidyverse::who`, que contém casos de tuberculose (TB) separados por ano, país, gênero e método e diagnose. Os dados são oriundos do *2014 World Health Organization Global Tuberculosis Report* disponível em: <http://www.who.int/tb/country/data/download/en/>.

Estudo de caso

Como dito na introdução, o conjunto de dados contém informações sobre os casos de tuberculose no mundo. As informações epidemiológicas contidas nesta base de dados são riquíssimas, porém é desafiador trabalhar com os dados na forma em que são fornecidos:

who

```
## # A tibble: 7,240 x 60
##   country iso2 iso3 year
##   <chr>    <chr> <chr> <dbl>
## 1 Afghanis~ AF    AFG    1980
## 2 Afghanis~ AF    AFG    1981
## 3 Afghanis~ AF    AFG    1982
## 4 Afghanis~ AF    AFG    1983
## 5 Afghanis~ AF    AFG    1984
```

```

## 6 Afghanis~ AF    AFG    1985
## 7 Afghanis~ AF    AFG    1986
## 8 Afghanis~ AF    AFG    1987
## 9 Afghanis~ AF    AFG    1988
## 10 Afghanis~ AF   AFG    1989
## # i 7,230 more rows
## # i 56 more variables:
## #   new_sp_m014 <dbl>,
## #   new_sp_m1524 <dbl>,
## #   new_sp_m2534 <dbl>,
## #   new_sp_m3544 <dbl>, ...

```

Esse é um conjunto de dados real bem típico. Ele contém colunas redundantes, códigos estranhos de variáveis e muitos valores faltantes. Resumindo, `who` é bagunçado, e precisaremos de vários passos para arrumá-lo. Como `dplyr`, o `tidyverse` é projetado para que cada função faça uma única coisa muito bem. Em situações reais, isso significa que você normalmente precisa juntar vários verbos em um pipeline.

O melhor lugar para começar é quase sempre reunindo as colunas que não são variáveis. Vamos dar uma olhada no que temos:

- Parece que `country`, `iso2` e `iso3` são três variáveis que redundantemente especificam o país.
- `year` também é claramente uma variável.
- Nós ainda não sabemos o que são todas as outras colunas, mas dada a estrutura dos nomes das variáveis, provavelmente são valores, não variáveis.

Então precisamos reunir todas as colunas de `new_sp_m014` até `newrel_f65`. Não sabemos, contudo, o que esses valores representam, então lhe daremos o nome genérico de “key”. Nós sabemos que as células representam a contagem de casos, então usaremos a variável `cases`. Há vários valores faltantes na representação atual, então, por enquanto, usaremos `na.rm = TRUE` para podermos focar nos valores que são apresentados.

```

(who1 <- who %>%
  gather(
    new_sp_m014:newrel_f65,
    key = "key",
    value = "cases",
    na.rm = TRUE
  ))

## # A tibble: 76,046 x 6
##       country   iso2   iso3   year cases
##       <fct>     <fct>  <fct> <dbl> <dbl>
## 1      AFG      AFG      AFG 1985  1.00
## 2      AFG      AFG      AFG 1986  1.00
## 3      AFG      AFG      AFG 1987  1.00
## 4      AFG      AFG      AFG 1988  1.00
## 5      AFG      AFG      AFG 1989  1.00
## # ... with 76,041 more rows
## # ... and 56 more variables:
## #   new_sp_m014 <dbl>,
## #   new_sp_m1524 <dbl>,
## #   new_sp_m2534 <dbl>,
## #   new_sp_m3544 <dbl>, ...

```

```

##   <chr>     <chr> <chr> <dbl>
## 1 Afghanis~ AF    AFG    1997
## 2 Afghanis~ AF    AFG    1998
## 3 Afghanis~ AF    AFG    1999
## 4 Afghanis~ AF    AFG    2000
## 5 Afghanis~ AF    AFG    2001
## 6 Afghanis~ AF    AFG    2002
## 7 Afghanis~ AF    AFG    2003
## 8 Afghanis~ AF    AFG    2004
## 9 Afghanis~ AF    AFG    2005
## 10 Afghanis~ AF   AFG    2006
## # i 76,036 more rows
## # i 2 more variables:
## #   key <chr>, cases <dbl>

```

Podemos conseguir algumas dicas da estrutura dos valores na nova coluna `key` ao contá-los:

```

who1 %>%
  count(key)

```

```

## # A tibble: 56 x 2
##   key           n
##   <chr>     <int>
## 1 new_ep_f014 1032
## 2 new_ep_f1524 1021
## 3 new_ep_f2534 1021
## 4 new_ep_f3544 1021
## 5 new_ep_f4554 1017
## 6 new_ep_f5564 1017
## 7 new_ep_f65   1014
## 8 new_ep_m014  1038
## 9 new_ep_m1524 1026
## 10 new_ep_m2534 1020
## # i 46 more rows

```

Você pode até ser capaz de analisar isso sozinho com um pouco de raciocínio e experimentação, mas felizmente nós temos o dicionário de dados por perto. Ele nos diz:

1. As primeiras três letras de cada coluna denotam se a coluna contém casos novos ou antigos de TB. Nesse conjunto de dados, cada uma delas contém novos casos.

2. As das letras seguintes descrevem o tipo de TB:

- *rel* é para casos de relapsidade.
- *ep* é para casos de TB extrapulmonar.
- *sn* é para casos de TB pulmonar que não poderiam ser diagnosticados por uma amostra pulmonar (amostra negativa).
- *sp* é para casos de TB pulmonar que poderiam ser diagnosticadas por amostra pulmonar (amostra positiva).

3. A sexta letra dá o gênero dos pacientes de TB. O conjunto de dados agrupa casos de homens (*m*) e mulheres (*f*).
4. O restante dos números dá a faixa etária. o conjunto de dados agrupa os casos em sete faixas etárias:

- 014 = 0 a 14 anos
- 1524 = 15 a 34 anos
- 2534 = 25 a 34 anos
- 3544 = 35 a 44 anos
- 4554 = 45 a 54 anos
- 5564 = 55 a 64 anos
- 65 = 65 ou mais

Precisamos fazer uma pequena correção no formato dos nomes e colunas: infelizmente os nomes são levemente inconsistentes porque, em vez de *new_rel*, temos *newrel* (é difícil identificar isso aqui, mas se você não fizer a correção, teremos erros nos passos subsequentes).

```
(who2 <- who1 %>%
  mutate(key = stringr::str_replace(key, "newrel", "new_rel")))
```

```
## # A tibble: 76,046 x 6
##   country iso2 iso3 year
##   <chr>    <chr> <chr> <dbl>
## 1 Afghanis~ AF    AFG    1997
## 2 Afghanis~ AF    AFG    1998
## 3 Afghanis~ AF    AFG    1999
## 4 Afghanis~ AF    AFG    2000
## 5 Afghanis~ AF    AFG    2001
## 6 Afghanis~ AF    AFG    2002
## 7 Afghanis~ AF    AFG    2003
## 8 Afghanis~ AF    AFG    2004
## 9 Afghanis~ AF    AFG    2005
```

```
## 10 Afghanis~ AF      AFG      2006
## # i 76,036 more rows
## # i 2 more variables:
## #   key <chr>, cases <dbl>
```

Podemos agora separar os valores em cada código com duas passagens de `separate()`. A primeira passagem separará os códigos em cada underscore:

```
(who3 <- who2 %>%
  separate(key, c("new", "type", "sexage"), sep = "_"))
```

```
## # A tibble: 76,046 x 8
##       country iso2 iso3    year
##       <chr>   <chr> <chr> <dbl>
## 1 Afghanis~ AF     AFG    1997
## 2 Afghanis~ AF     AFG    1998
## 3 Afghanis~ AF     AFG    1999
## 4 Afghanis~ AF     AFG    2000
## 5 Afghanis~ AF     AFG    2001
## 6 Afghanis~ AF     AFG    2002
## 7 Afghanis~ AF     AFG    2003
## 8 Afghanis~ AF     AFG    2004
## 9 Afghanis~ AF     AFG    2005
## 10 Afghanis~ AF     AFG   2006
## # i 76,036 more rows
## # i 4 more variables:
## #   new <chr>, type <chr>,
## #   sexage <chr>, cases <dbl>
```

Depois podemos deixar de lado a coluna `new`, porque ela é constante neste conjunto de dados. Enquanto estamos deixando colunas de lado, vamos deixar de lado também `iso2` e `iso3`, já que são redundantes.

```
who3 %>%
  count(new)
```

```
## # A tibble: 1 x 2
##   new      n
##   <chr> <int>
## 1 new    76046
```

```
(who4 <- who3 %>%
  select(-new, -iso2, -iso3))
```

```
## # A tibble: 76,046 x 5
##   country year type sexage
##   <chr>    <dbl> <chr> <chr>
## 1 Afghani~ 1997 sp    m014
## 2 Afghani~ 1998 sp    m014
## 3 Afghani~ 1999 sp    m014
## 4 Afghani~ 2000 sp    m014
## 5 Afghani~ 2001 sp    m014
## 6 Afghani~ 2002 sp    m014
## 7 Afghani~ 2003 sp    m014
## 8 Afghani~ 2004 sp    m014
## 9 Afghani~ 2005 sp    m014
## 10 Afghani~ 2006 sp   m014
## # i 76,036 more rows
## # i 1 more variable:
## #   cases <dbl>
```

Em seguida vamos separar sexage em sex e age ao separar depois do primeiro caractere:

```
(who5 <- who4 %>%
  separate(sexage, c("sex", "age"), sep = 1))
```

```
## # A tibble: 76,046 x 6
##   country year type sex   age
##   <chr>    <dbl> <chr> <chr> <dbl>
## 1 Afghani~ 1997 sp    m     0
## 2 Afghani~ 1998 sp    m     0
## 3 Afghani~ 1999 sp    m     0
## 4 Afghani~ 2000 sp    m     0
## 5 Afghani~ 2001 sp    m     0
## 6 Afghani~ 2002 sp    m     0
## 7 Afghani~ 2003 sp    m     0
## 8 Afghani~ 2004 sp    m     0
## 9 Afghani~ 2005 sp    m     0
## 10 Afghani~ 2006 sp   m     0
## # i 76,036 more rows
## # i 2 more variables:
## #   age <chr>, cases <dbl>
```

O conjunto de dados who agora está arrumado!

Mostrei um pedaço de código de cada vez, atribuindo cada resultado provisório a uma nova variável. Normalmente não é assim que você trabalha interativamente. Em vez disso, você construiria gradualmente um pipe complexo:

```
who %>%
  gather(
    new_sp_m014:newrel_f65,
    key = "code",
    value = "cases",
    na.rm = TRUE
  ) %>%
  mutate(
    code = stringr::str_replace(code, "newrel", "new_rel")
  ) %>%
  separate(code, c("new", "type", "sexage")) %>%
  select(-new, -iso2, -iso3) %>%
  separate(sexage, c("sex", "age"), sep = 1)

## # A tibble: 76,046 x 6
##   country     year type sex
##   <chr>      <dbl> <chr> <chr>
## 1 Afghanis~  1997 sp    m
## 2 Afghanis~  1998 sp    m
## 3 Afghanis~  1999 sp    m
## 4 Afghanis~  2000 sp    m
## 5 Afghanis~  2001 sp    m
## 6 Afghanis~  2002 sp    m
## 7 Afghanis~  2003 sp    m
## 8 Afghanis~  2004 sp    m
## 9 Afghanis~  2005 sp    m
## 10 Afghanis~ 2006 sp    m
## # i 76,036 more rows
## # i 2 more variables:
## #   age <chr>, cases <dbl>
```



Bibliografia

Hadley Wickham and Garrett Grolemund. *R para Data Science*. Alta Books, Rio de Janeiro, 2019.