R para Data Science

Importe, Arrume, Transforme, Visualize e Modele Dados

Hadley Wickham e Garrett Grolemund

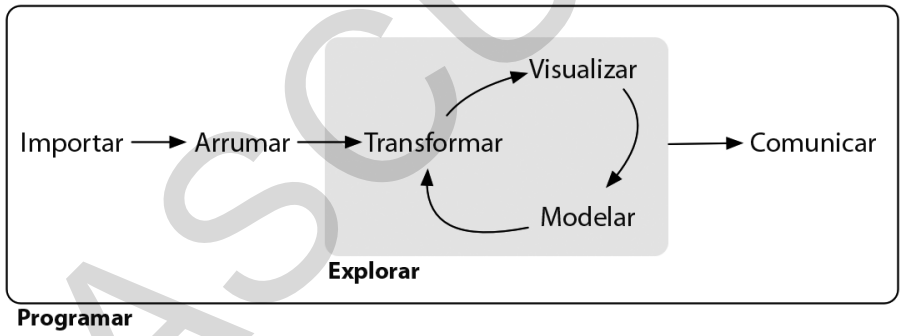
ALTA BOOKS
EDITORA
Rio de Janeiro, 2018

Sumário

Parte II. Wrangle

PARTE I
Explorar

O objetivo da primeira parte deste livro é o de que você fique atualizado com as ferramentas básicas de *exploração de dados* o mais rápido possível. A exploração de dados é a arte de observar seus dados, gerar hipóteses e testá-las com rapidez e então repetir, repetir, repetir, e o objetivo dessa exploração é gerar muitos leads promissores que você pode explorar mais tarde com mais profundidade.



Nesta parte do livro você aprenderá algumas ferramentas úteis que apresentam uma recompensa imediata:

- A visualização é um ótimo lugar para começar com a programação R, porque a recompensa é muito clara: você pode fazer gráficos elegantes e informativos que o ajudam a entender os dados. No Capítulo 1 você mergulha na visualização, aprendendo a estrutura básica de um gráfico **ggplot2** e técnicas poderosas para transformar dados em gráficos.

- Só a visualização normalmente não é o suficiente, então, no Capítulo 3, você aprenderá os verbos-chave que lhe permitem selecionar variáveis importantes, filtrar observações-chave, criar novas variáveis e calcular resumos.
- Finalmente, no Capítulo 5, você combinará visualização e transformação com sua curiosidade e ceticismo para fazer e responder perguntas interessantes sobre dados.

A modelagem é uma parte importante do processo exploratório, mas você não tem as habilidades para aprendê-la ou aplicá-la eficazmente ainda. Nós voltaremos a ela na Parte IV, assim que você estiver melhor equipado com mais ferramentas de data wrangling e de programação.

Aninhados entre esses três capítulos que lhe ensinam as ferramentas de exploração, há outros três capítulos que focam no fluxo de trabalho R. Nos Capítulos 2, 4 e 6, você aprenderá boas práticas para escrever e organizar seu código R. Eles o prepararão para o sucesso no longo prazo, enquanto lhe darão as ferramentas para se organizar quando atacar projetos de verdade.

CAPÍTULO 1

Visualização de Dados com ggplot2

Introdução

> O gráfico simples trouxe mais informações à mente dos analistas de dados do que qualquer outro dispositivo.
>
> — John Tukey

Este capítulo lhe ensinará como visualizar seus dados usando o **ggplot2**. O R tem vários sistemas para fazer gráficos, mas o **ggplot2** é um dos mais elegantes e versáteis. O **ggplot2** implementa a *gramática dos gráficos*, um sistema coerente para descrever e construir gráficos. Com **ggplot2** você pode fazer mais rápido, ao aprender um sistema e aplicá-lo em muitos lugares.

Se quiser conhecer mais sobre os fundamentos teóricos de **ggplot2** antes de começar, eu recomendaria a leitura de "A Layered Grammar of Graphics" (*http://vita.had.co.nz/papers/layered-grammar.pdf* — conteúdo em inglês).

Pré-requisitos

Este capítulo foca no **ggplot2**, um dos membros centrais do tidyverse. Para acessar os conjuntos de dados, páginas de ajuda e funções que usaremos neste capítulo, carregue o tidyverse ao executar este código:

```
library(tidyverse)
#> Loading tidyverse: ggplot2
#> Loading tidyverse: tibble
#> Loading tidyverse: tidyr
#> Loading tidyverse: readr
#> Loading tidyverse: purrr
```

```
#> Loading tidyverse: dplyr
#> Conflicts with tidy packages ----------------------------
#> filter(): dplyr, stats
#> lag():    dplyr, stats
```

Esta linha de código carrega o núcleo do tidyverse, pacotes que você usará em quase todas as análises de dados. Ela também lhe diz quais funções do tidyverse entram em conflito com funções do R básico (ou de outros pacotes que você possa ter carregado).

Se você executar esse código e obtiver a mensagem de erro "there is no package called 'tidyverse'", precisará primeiro instalá-lo e depois executar `library()` novamente:

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

Você só precisa instalar o pacote uma vez, mas precisa recarregá-lo sempre que iniciar uma nova sessão.

Se precisarmos ser explícitos sobre de onde vem uma função (ou conjunto de dados), usaremos a forma especial `package::function()`. Por exemplo, `ggplot2::ggplot()` lhe diz explicitamente que estamos usando a função `ggplot()` do pacote **ggplot2**.

Primeiros Passos

Usaremos nosso primeiro gráfico para responder a uma pergunta: carros com motores maiores usam mais combustível que carros com motores menores? Você provavelmente já tem uma resposta, mas tente torná-la precisa. Com o que a relação entre tamanho de motor e eficácia do combustível se parece? É positivo? Negativo? Linear? Não linear?

O Data Frame mpg

Você pode testar sua resposta com o data frame `mpg` encontrado em **ggplot2** (também conhecido como `ggplot2::mpg`). Um *data frame* é uma coleção retangular de variáveis (nas colunas) e observações (nas linhas). O `mpg` contém observações coletadas pela Agência de Proteção Ambiental dos Estados Unidos sobre 38 modelos de carros:

```
mpg
#> # A tibble: 234 × 11
#>   manufacturer model displ  year   cyl      trans   drv
#>          <chr> <chr> <dbl> <int> <int>      <chr> <chr>
#> 1         audi    a4   1.8  1999     4   auto(l5)     f
#> 2         audi    a4   1.8  1999     4 manual(m5)     f
```

```
#> 3          audi    a4   2.0  2008     4 manual(m6)     f
#> 4          audi    a4   2.0  2008     4   auto(av)     f
#> 5          audi    a4   2.8  1999     6   auto(l5)     f
#> 6          audi    a4   2.8  1999     6 manual(m5)     f
#> # ... with 228 more rows, and 4 more variables:
#> # cty <int>, hwy <int>, fl <chr>, class <chr>
```

Entre as variáveis em `mpg` estão:

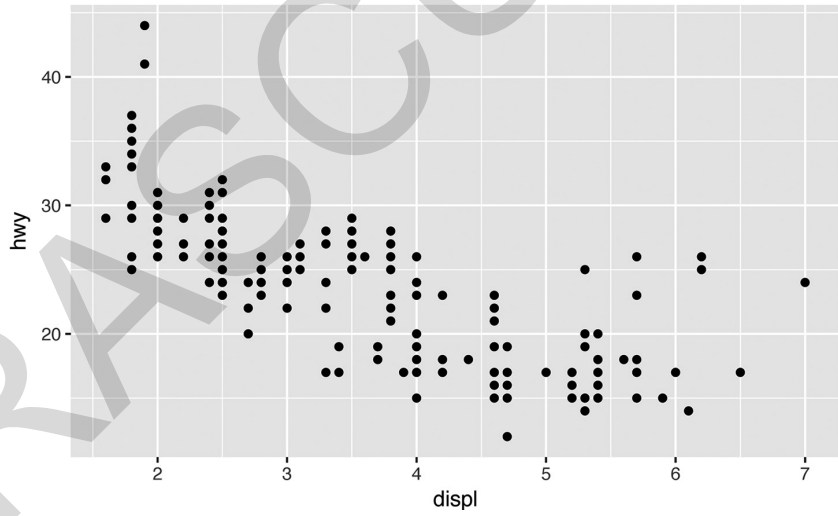
- `displ`, o tamanho do motor de um carro, em litros.
- `hwy`, a eficiência do combustível de um carro na estrada, em milhas por galão (mpg). Um carro com uma eficiência de combustível baixa consome mais combustível do que um carro com eficiência de combustível alta quando viajam a mesma distância.

Para aprender mais sobre `mpg`, abra sua página de ajuda executando `?mpg`.

Criando um ggplot

Para fazer o gráfico de `mpg`, execute o código para colocar `displ` no eixo x e `hwy` no eixo y:

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



O gráfico mostra uma relação negativa entre o tamanho do motor (`displ`) e a eficiência do combustível (`hwy`). Em outras palavras, carros com motores grandes usam mais combustível. Isso confirma ou refuta sua hipótese sobre eficiência do combustível e tamanho do motor?

Com **ggplot2** você começa um gráfico com a função `ggplot()`, que cria um sistema de coordenadas ao qual você pode adicionar camadas. O primeiro argumento de `ggplot()` é o conjunto de dados para usar no gráfico. Então `ggplot(data = mpg)` cria um gráfico em branco, mas não é muito interessante, por isso não o mostrarei aqui.

Você completa seu gráfico adicionando uma ou mais camadas a `ggplot()`. A função `geom_point()` adiciona uma camada de pontos ao seu gráfico, que cria um gráfico de dispersão. O **ggplot2** vem com muitas funções geom que adicionam um tipo diferente de camada a um gráfico. Você aprenderá várias delas ao longo deste capítulo.

Cada função geom no **ggplot2** recebe um argumento `mapping`. Isso define como as variáveis de seu conjunto de dados são mapeadas para propriedades visuais. O argumento `mapping` é sempre combinado com `aes()`, e os argumentos `x` e `y` de `aes()` especificam quais variáveis mapear para os eixos x e y. O **ggplot2** procura a variável mapeada no argumento `data`, neste caso, `mpg`.

Um Template de Gráfico

Vamos transformar este código em um template reutilizável para fazer gráficos com **ggplot2**. Para fazer um gráfico, substitua as seções entre colchetes angulares por um conjunto de dados, uma função geom ou uma coleção de mapeamentos:

```
ggplot(data = <DATA>) +
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

A continuação deste capítulo lhe mostrará como completar e ampliar este template para fazer tipos diferentes de gráficos. Começaremos com o componente `<MAPPINGS>`.

Exercícios

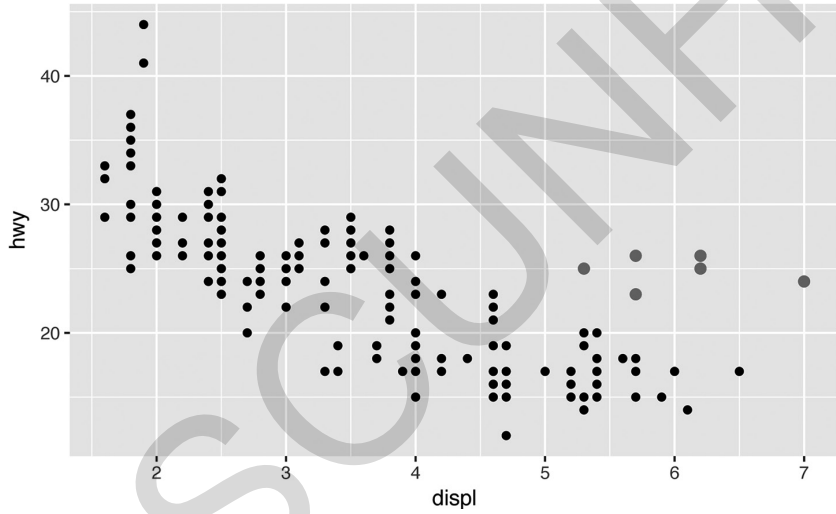
1. Execute `ggplot(data = mpg)`. O que você vê?
2. Quantas linhas existem em `mtcars`? Quantas colunas?
3. O que a variável `drv` descreve? Leia a ajuda de `?mpg` para descobrir.
4. Faça um gráfico de dispersão de `hwy` versus `cyl`.

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

Mapeamentos Estéticos

> O maior valor de uma imagem é quando ela nos força a notar o que nunca esperávamos ver.
>
> — John Tukey

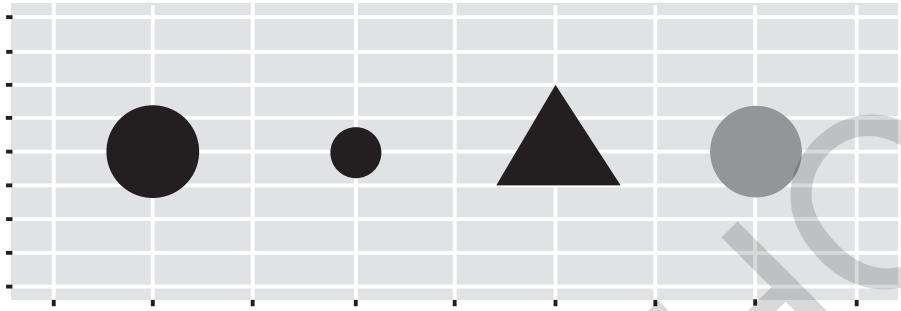
No gráfico a seguir, um grupo de pontos (destacados em cinza) parece ficar fora da tendência linear. Esses carros têm uma milhagem mais alta do que o esperado. Como você explica esse fato?



Vamos supor que esses carros sejam híbridos. Uma maneira de testar esta hipótese é ver o valor `class` de cada carro. A variável `class` do conjunto de dados `mpg` classifica os carros em grupos como compacto, tamanho médio e SUV. Se os pontos afastados são híbridos, eles deveriam ser classificados como carros compactos ou, talvez, sub-compactos (lembre-se de que esses dados foram coletados antes dos caminhões e SUVs híbridos se tornarem populares).

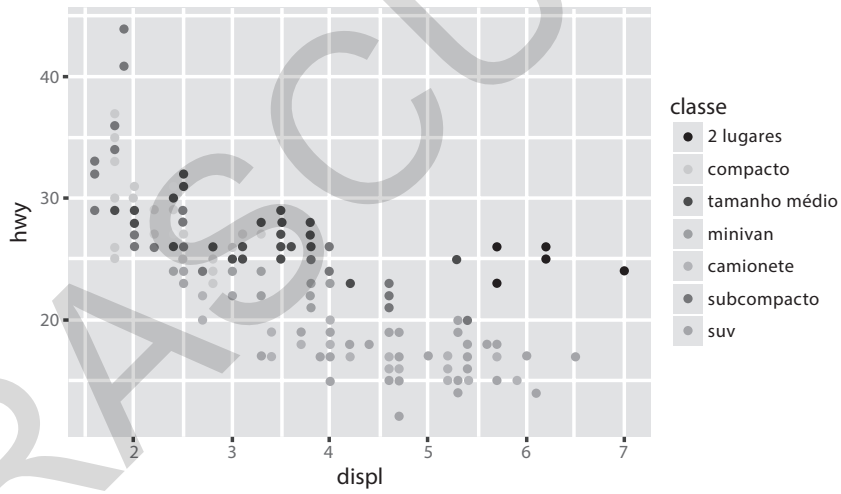
Você pode adicionar uma terceira variável, como `class`, a um gráfico de dispersão bidimensional ao mapeá-lo a um *estético* (*aesthetic*). Um estético é uma propriedade visual dos objetos em seu gráfico. Estéticos incluem coisas como tamanho, forma ou cor dos seus pontos. Você pode exibir um ponto (como o mostrado a seguir) de diferentes maneiras ao mudar os valores de suas propriedades estéticas. Visto que já usamos a

palavra "valor" para descrever dados, usaremos a palavra "nível" para descrever propriedades estéticas. Aqui nós mudamos os níveis do tamanho, da forma e da cor de um ponto para torná-lo pequeno, triangular ou cinza:



Você pode transmitir informações sobre seus dados ao mapear a estética em seu gráfico para as variáveis em seu conjunto de dados. Por exemplo, você pode mapear as cores de seus pontos para a variável `class` para revelar a classe de cada carro:

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



(Se você prefere o inglês britânico, como Hadley, pode usar *colour*, em vez de *color*.)

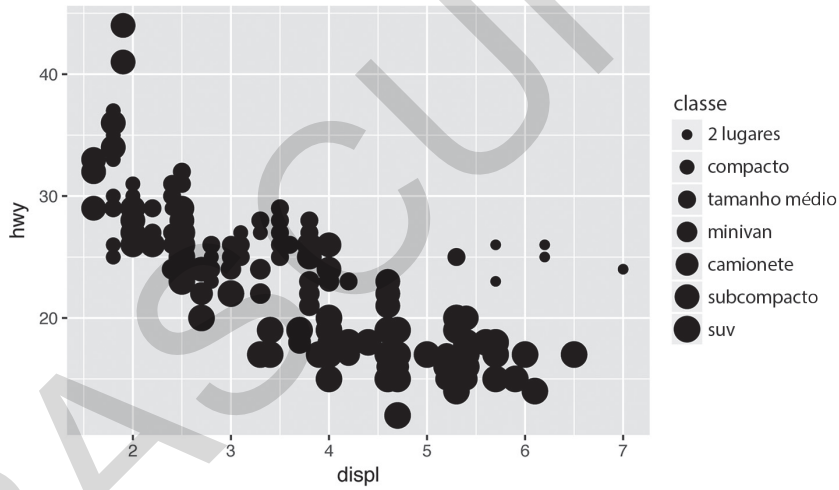
Para mapear uma estética a uma variável, associe o nome da estética ao nome da variável dentro de `aes()`. O **ggplot2** atribuirá automaticamente um nível singular de estética (aqui uma cor singular) para cada valor singular da variável, um processo conhecido

como *escalar* (*scaling*). O **ggplot2** também adicionará uma legenda que explica quais níveis correspondem a quais valores.

As cores revelam que muitos dos pontos estranhos são carros de dois lugares. Eles não parecem ser híbridos. São, de fato, esportivos! Carros esportivos têm motores grandes, como SUVs e caminhonetes, mas estrutura pequena, como carros de tamanho médio ou compactos, o que melhora sua milhagem de combustível. Em retrospecto, era improvável que esses carros fossem híbridos, já que têm motores grandes.

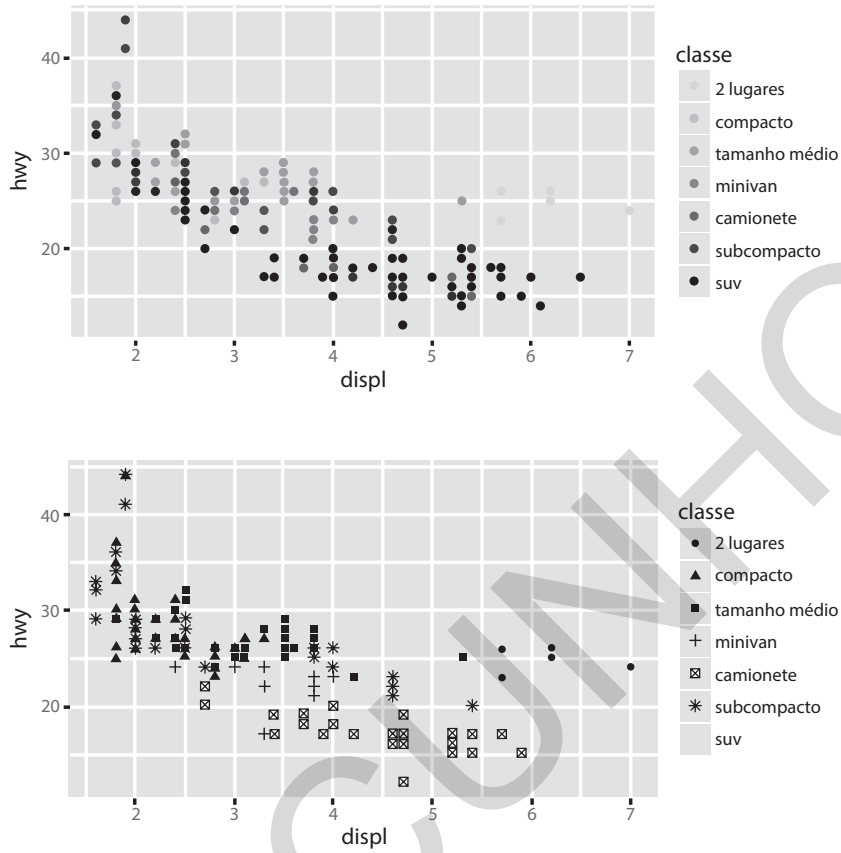
No exemplo anterior, mapeamos `class` à estética de cor, mas poderíamos ter mapeado `class` à estética de tamanho da mesma maneira. Neste caso, o tamanho exato de cada ponto revelaria sua afiliação de classe. Nós recebemos um *aviso* (*warning*) aqui, porque mapear uma variável não ordenada (`class`) à uma estética ordenada (`size`) não é uma boa ideia:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, size = class))
#> Warning: Using size for a discrete variable is not advised.
```



Ou poderíamos ter mapeado `class` à estética *alpha*, que controla a transparência dos pontos, ou a forma dos pontos:

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

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



O que acontece com os SUVs? O **ggplot2** só utilizará seis formas de cada vez. Por padrão, grupos adicionais ficam de fora do gráfico quando você usa essa estética.

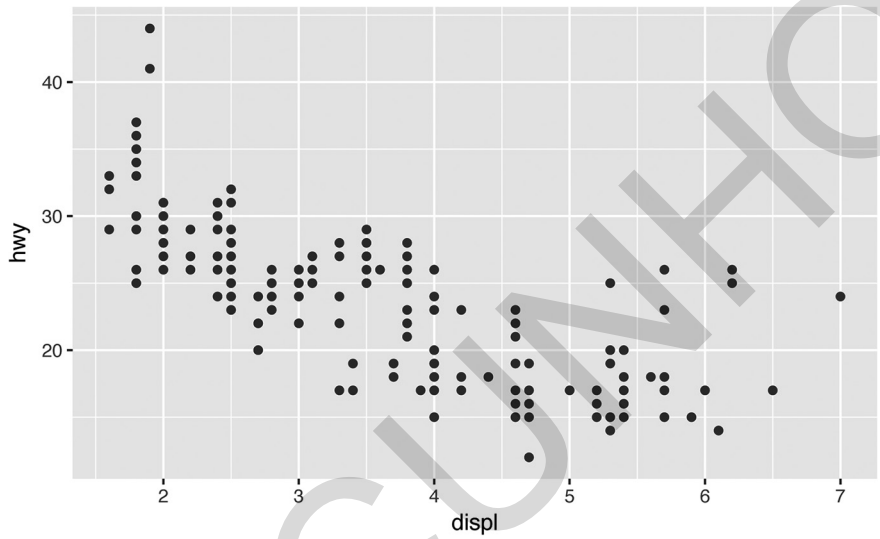
Para cada estética você usa o `aes()` para associar o nome da estética à variável a ser exibida. A função `aes()` reúne cada um dos mapeamentos estéticos usados por uma camada e os passa para o argumento de mapeamento da camada. A sintaxe destaca um insight útil sobre x e y: as localizações x e y de um ponto são, por si, propriedades visuais estéticas que você pode mapear às variáveis para exibir informações sobre os dados.

Uma vez mapeada uma estética, o **ggplot2** cuida do resto. Ele seleciona uma escala razoável para usar com a estética e constrói uma legenda que explica o mapeamento entre níveis e valores. Para estéticas x e y, o **ggplot2** não cria uma legenda, mas uma

linha de eixo com marcas e um rótulo. Essa linha age como uma legenda, que explica o mapeamento entre localizações e valores.

Você também pode *configurar* as propriedades de sua geom manualmente. Por exemplo, nós podemos deixar todos os pontos em nosso gráfico blue (azul)[1]:

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



Aqui a cor não transmite informações sobre uma variável, só muda a aparência do gráfico. Para configurar uma estética manualmente, configure-a por nome como um argumento da sua função geom, isto é, *fora* de `aes()`. Você precisará escolher um valor que faça sentido para essa estética:

- O nome de uma cor como uma string de caracteres.
- O tamanho de um ponto em mm.
- A forma de um ponto como um número, como mostrado na Figura 1-1. Há algumas duplicatas aparentes: por exemplo, 0, 15 e 22 são quadrados. A diferença vem da interação das estéticas `color` e `fill`. As formas ocas (0–14) têm uma borda determinada por `color`, as formas sólidas (15–18) são preenchidas com `color`, e as formas preenchidas (21–24) têm uma borda de `color` e são preenchidas por `fill`.

[1] N.E.: As imagens apresentadas nesta obra em sua versão colorida estão disponíveis no site da editora. Para visualiza-las, acesse www.altabook.com.br.

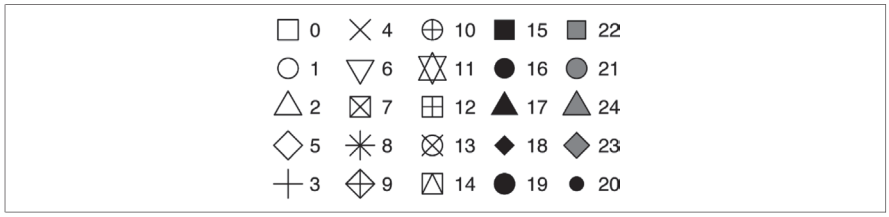
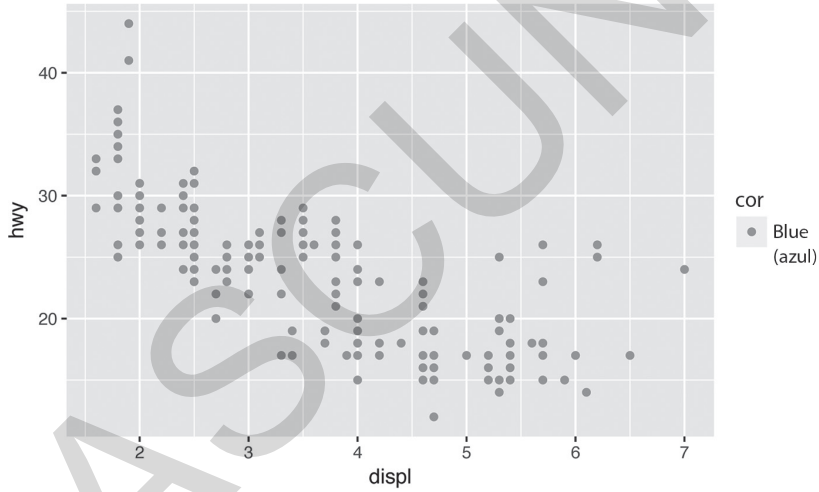


Figura 1-1. R tem 25 formas incorporadas que são identificadas por números

Exercícios

1. O que há de errado com este código? Por que os pontos não estão pretos?

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



2. Quais variáveis em `mpg` são categóricas? Quais variáveis são contínuas? (Dica: digite `?mpg` para ler a documentação do conjunto de dados.) Como você pode ver essa informação quando executa `mpg`?
3. Mapeie uma variável contínua para `color`, `size` e `shape`. Como essas estéticas se comportam de maneira diferente para variáveis categóricas e contínuas?
4. O que acontece se você mapear a mesma variável a várias estéticas?
5. O que a estética `stroke` faz? Com que formas ela trabalha? (Dica: use `?geom_point`.)

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

Problemas Comuns

Quando você começar a executar código R, provavelmente encontrará problemas. Não se preocupe — isso acontece com todo mundo. Eu escrevo código R há anos, e todos os dias ainda escrevo código que não funciona!

Comece comparando cuidadosamente o código que você está executando com o código do livro. R é extremamente exigente, e um caractere no lugar errado pode fazer toda a diferença. Certifique-se de que todo (esteja combinado com um) e todo " esteja combinado com outro ". Às vezes você executará o código e não acontecerá nada. Verifique o lado esquerdo de seu console: se for um +, significa que o R não acha que você tenha digitado uma expressão completa e está esperando que você a termine. Neste caso, normalmente é fácil começar do zero de novo ao pressionar Esc para abortar o processamento do comando atual.

Um problema comum ao criar gráficos **ggplot2** é colocar o + no lugar errado: ele precisa ficar no final da linha, não no começo. Em outras palavras, certifique-se de não ter escrito acidentalmente o código deste jeito:

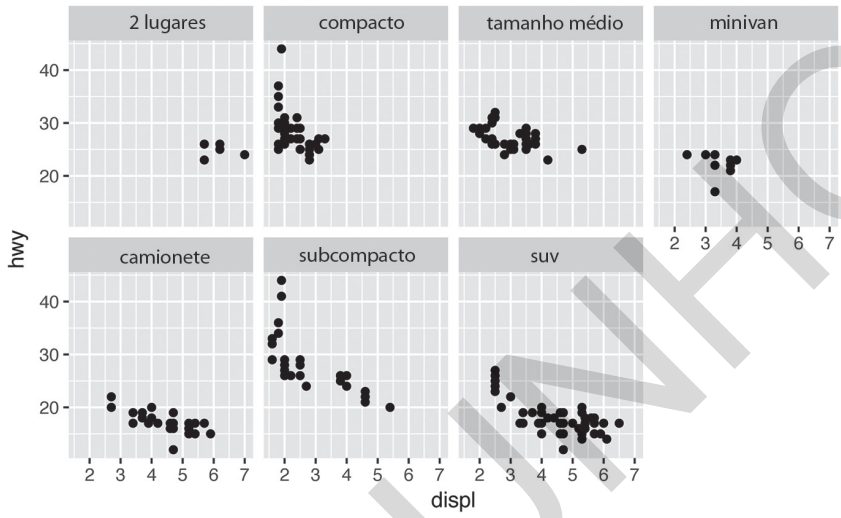
```
ggplot(data = mpg)
+ geom_point(mapping = aes(x = displ, y = hwy))
```

Se ainda estiver preso, tente a ajuda. Você pode obter ajuda sobre qualquer função R executando `?function_name` no console ou selecionando o nome da função e pressionando F1 no RStudio. Não se preocupe se a ajuda não parecer tão útil — pule para os exemplos e busque o código que combine com o que você está tentando fazer.

Se isso não ajudar, leia atentamente a mensagem de erro. Às vezes a resposta estará escondida lá! A resposta pode estar na mensagem de erro, mas se você for novato em R, ainda não saberá como compreendê-la. Outra ótima ferramenta é o Google: tente fazer uma busca pela mensagem de erro, já que é provável que outra pessoa tenha tido o mesmo problema e recebeu ajuda online.

Facetas

Uma maneira de adicionar mais variáveis é com estéticas. Outra maneira, particularmente útil para variáveis categóricas, é dividir seu gráfico em *facetas* — subgráficos que exibem um subconjunto dos dados.

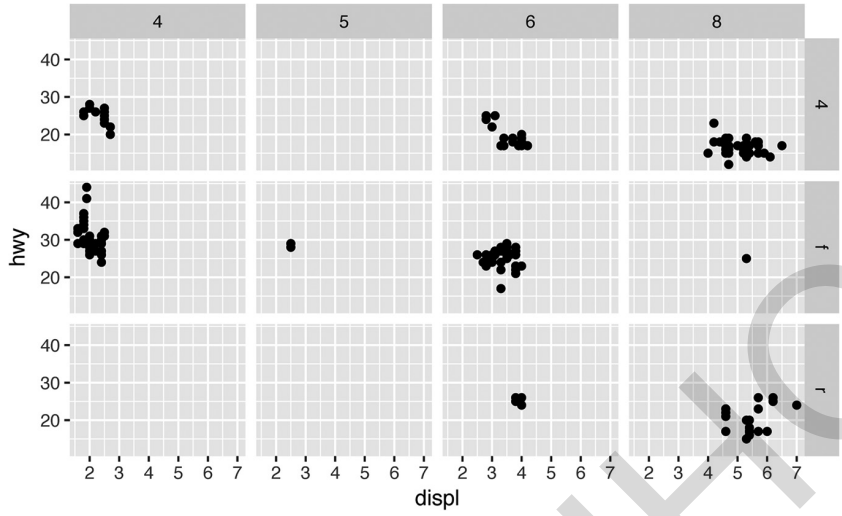


Para criar facetas de seu gráfico a partir de uma única variável, use `facet_wrap()`. O primeiro argumento de `facet_wrap()` deve ser uma fórmula, que você cria com `~` seguido de um nome de variável (aqui "fórmula" é o nome de uma estrutura de dados em R, não um sinônimo para "equação"). A variável que você passa para `facet_wrap()` deve ser discreta:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~ class, nrow = 2)
```

Para criar facetas de seu gráfico a partir de uma combinação de duas variáveis, adicione `facet_grid()` à sua chamada de gráfico. O primeiro argumento de `facet_grid()` também é uma fórmula. Desta vez a fórmula deve conter dois nomes de variáveis separados por um `~`:

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



Se preferir não criar facetas nas dimensões de linhas ou colunas, use um `.`, em vez do nome de uma variável. Por exemplo, `+ facet_grid(. ~ cyl)`.

Exercícios

1. O que acontece se você criar facetas em uma variável contínua?
2. O que significam 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 = drv, y = cyl))
```

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 ~ .)

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

4. Pegue o primeiro gráfico em facetas desta seção:

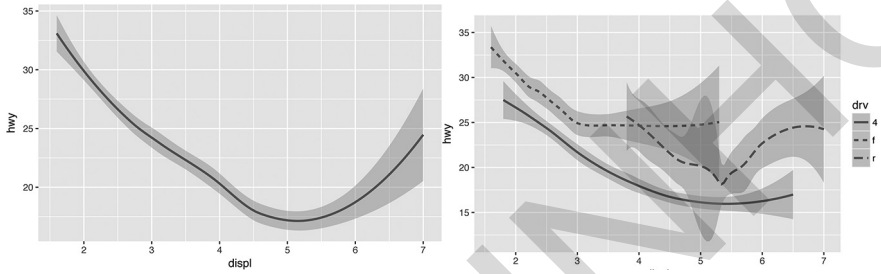
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  facet_wrap(~ class, nrow = 2)
```

   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?

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

Objetos Geométricos

Quais as similaridades desses dois gráficos?



Ambos os gráficos contêm a mesma variável `x` e a mesma variável `y`, e ambos descrevem os mesmos dados. Mas os gráficos não são idênticos. Cada gráfico usa um objeto visual diferente para representar os dados. Na sintaxe **ggplot2**, dizemos que eles usam *geoms* diferentes.

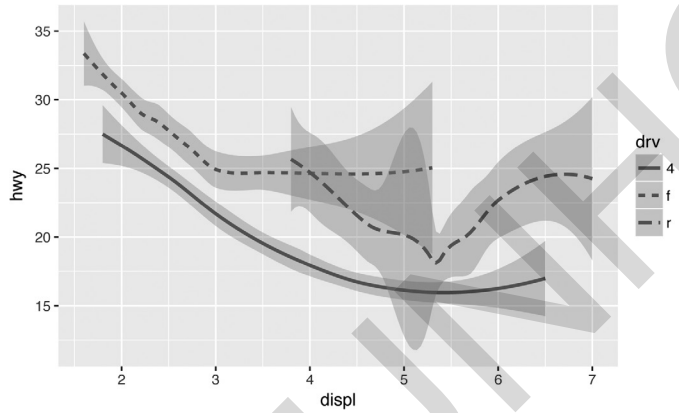
Um *geom* é o objeto geométrico que um gráfico usa para representar dados. As pessoas frequentemente descrevem gráficos pelo tipo de geom que ele usa. Por exemplo, gráficos de barra usam geoms de barra, gráficos de linha usam geoms de linha, diagramas de caixa usam geoms de caixa, e assim por diante. Gráficos de dispersão quebram a tendência, eles usam geom de ponto. Como vemos nos gráficos anteriores, você pode usar geoms diferentes para fazer gráficos dos mesmos dados. O gráfico à esquerda usa o geom de ponto, e o da direita usa o geom smooth, uma linha suave ajustada aos dados.

Para mudar o geom de seu gráfico, altere a função geom que você adiciona a `ggplot()`. Por exemplo, para gerar os gráficos anteriores, você pode usar este código:

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

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

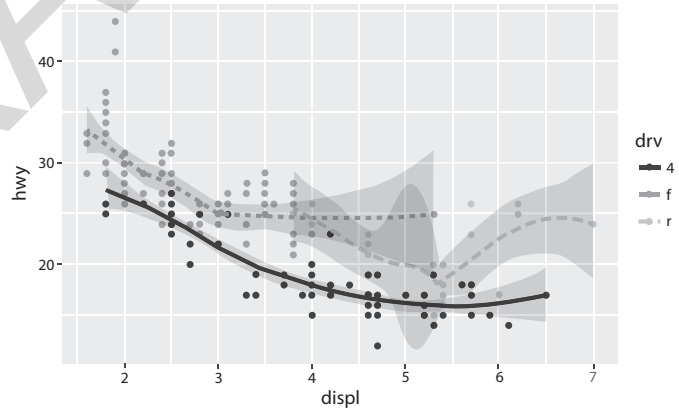
Cada função geom em **ggplot2** recebe um argumento `mapping`. Entretanto, nem toda estética funciona com todo geom. Você poderia configurar a forma de um ponto, mas não poderia configurar a "forma" de uma linha. Por outro lado, você *poderia* configurar o tipo de uma linha (`linetype`). O `geom_smooth()` desenhará uma linha diferente, com um tipo diferente, para cada valor singular da variável que você mapeia ao tipo de linha:

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



Aqui, `geom_smooth()` separa os carros em três linhas baseadas em seus valores `drv`, que descreve a transmissão de um carro. Uma linha descreve todos os pontos com um valor `4`, uma linha descreve todos os pontos com um valor `f`, e a outra descreve todos os pontos com um valor `r`. Aqui, `4` quer dizer tração nas quatro rodas, `f` é tração dianteira, e `r` é tração traseira.

Se isso parece estranho, podemos deixar mais claro sobrepondo as linhas sobre os dados brutos e então colorindo tudo de acordo com `drv`.

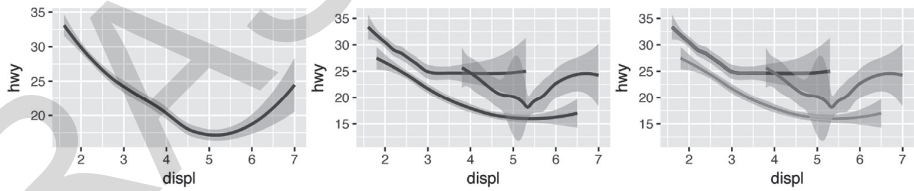


Note que esse gráfico contém dois geoms no mesmo espaço! Se isso o deixa animado, prepare-se. Na próxima seção aprenderemos como colocar vários geoms no mesmo gráfico.

O **ggplot2** fornece mais de 30 geoms, e pacotes de extensão fornecem ainda mais (veja uma amostra em *https://www.ggplot2-exts.org* — conteúdo em inglês). A melhor maneira de obter uma visão geral ampla é consultando a folha de cola do **ggplot2**, que você pode encontrar em *http://rstudio.com/cheatsheets* (conteúdo em inglês). Para aprender mais sobre qualquer geom único, use a ajuda: `?geom_smooth`.

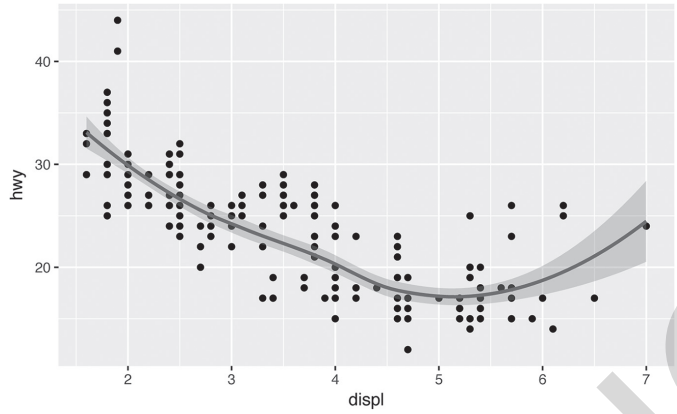
Muitos geoms, como o `geom_smooth()`, usam um único objeto geométrico para exibir várias linhas de dados. Para esses geoms, você pode configurar a estética `group` com uma variável categórica para desenhar vários objetos. O **ggplot2** desenhará um objeto separado para cada valor único da variável de agrupamento. Na prática, o **ggplot2** agrupará automaticamente os dados para esses geoms sempre que você mapear uma estética a uma variável discreta (como no exemplo `linetype`). É conveniente depender desse recurso, porque a estética de grupo por si não adiciona uma legenda ou características distintas aos geoms:

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

ggplot(data = mpg) +
  geom_smooth(mapping = aes(x = displ, y = hwy, group = drv))

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



Para exibir vários geoms no mesmo gráfico, adicione várias funções geom ao `ggplot()`:

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

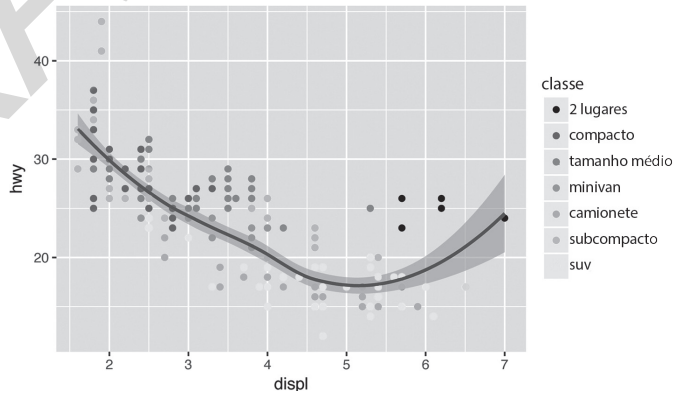


Isso, no entanto, introduz alguma duplicação ao seu código. Imagine que você quisesse mudar o eixo y para que exiba `cty` em vez de `hwy`. Você precisaria mudar a variável em dois lugares, e poderia esquecer de atualizar uma. É possível evitar esse tipo de repetição passando um conjunto de mapeamentos para `ggplot()`. O **ggplot2** tratará esses mapeamentos como mapeamentos globais que se aplicam a cada geom no gráfico. Em outras palavras, esse código produzirá o mesmo gráfico que o código anterior:

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

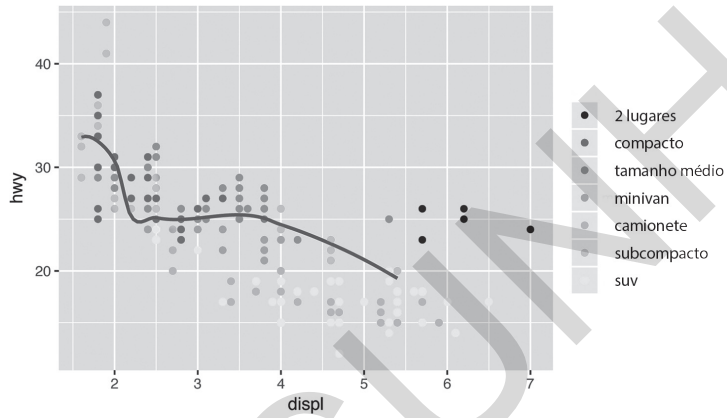
Se você colocar mapeamentos em uma função geom, o **ggplot2** os tratará como mapeamentos locais para a camada. Ele usará esses mapeamentos para ampliar ou sobrescrever os mapeamentos globais *apenas para aquela camada*. Isso possibilita exibir estéticas diferentes em camadas diferentes:

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



Você pode usar a mesma ideia para especificar um conjunto de dados diferente para cada camada. Aqui, nossa linha suave exibe apenas um subconjunto do conjunto de dados `mpg`, os carros subcompactos. O argumento de dados local em `geom_smooth()` desconsidera o argumento de dados global em `ggplot()` apenas para aquela camada:

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(color = class)) +
  geom_smooth(
    data = filter(mpg, class == "subcompact"),
    se = FALSE
  )
```



(Você aprenderá como `filter()` funciona no próximo capítulo. Por enquanto, saiba que esse comando seleciona apenas os carros subcompactos.)

Exercícios

1. Que geom você usaria para desenhar um gráfico de linha? Um diagrama de caixa (boxplot)? Um histograma? Um gráfico de área?
2. Execute este código em sua cabeça e preveja como será o resultado. Depois execute o código em R e confira suas previsões:

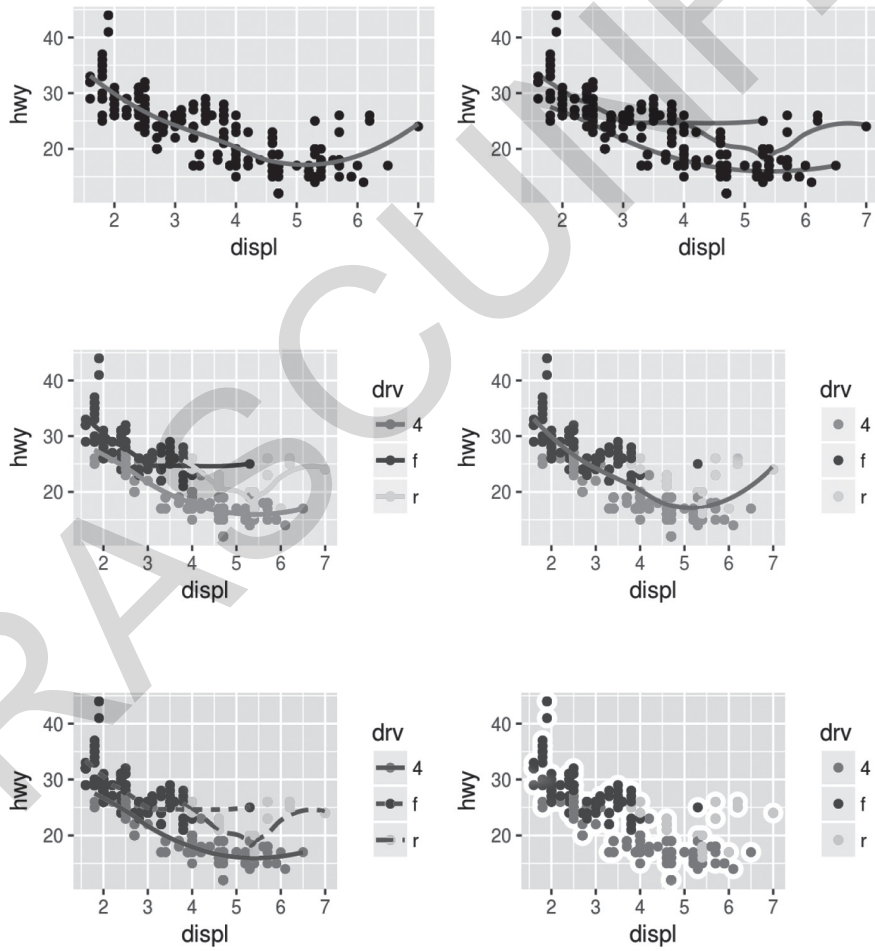
```
ggplot(
  data = mpg,
  mapping = aes(x = displ, y = hwy, color = drv)
) +
  geom_point() +
  geom_smooth(se = FALSE)
```

3. O que `show.legend = FALSE` faz? O que acontece se você removê-lo? Por que você acha que usei isso anteriormente no capítulo?
4. O que o argumento `se` para `geom_smooth()` faz?

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

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

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

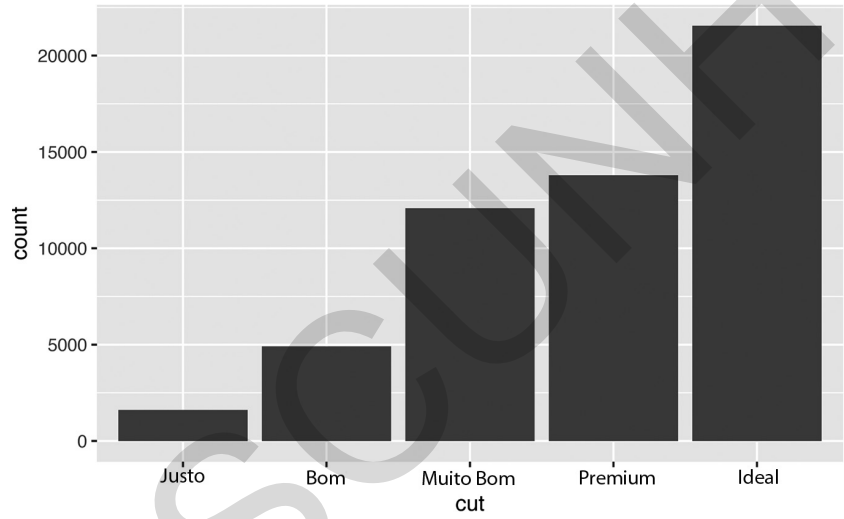
6. Recrie o código R necessário para gerar os seguintes gráficos:



Transformações Estatísticas

Gráficos de barra parecem simples, mas são interessantes, pois revelam algo sutil sobre os gráficos. Considere um gráfico de barra básico, como o desenhado com `geom_bar()`. O gráfico a seguir exibe o número total de diamantes no conjunto de dados `diamonds`, agrupado por `cut`. O conjunto de dados `diamonds` vem no **ggplot2** e contém informações sobre ~54.000 diamantes, incluindo `price`, `carat`, `color`, `clarity` e `cut` de cada um. A seguir vemos que há mais diamantes disponíveis com cortes de alta qualidade do que com cortes de baixa qualidade:

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



No eixo x, o gráfico exibe `cut`, uma variável de `diamonds`. No eixo y, exibe `count`, apesar de não ser uma variável em `diamonds`! De onde vem `count`? Muitos gráficos, como os de dispersão, plotam os valores brutos de seu conjunto de dados. Outros gráficos, como os de barra, calculam novos valores para usar:

- Gráficos de barra, histogramas e polígonos de frequência armazenam seus dados e então plotam as contagens de armazenamento, o número de pontos que cai em cada espaço.
- Smoothers encaixam um modelo em seus dados e então plotam as previsões do modelo.